DNLP Notes by Alessandro De Marco

Text Preprocessing

Un testo può essere diviso in:

- Character: Smallest text unit
- Word: Series of letters between spaces
- N-Gram: a set of words
- Multi-word expressions: textual form made up of at least two lexical items, like Artificial Intellingence
- Sentence: Text snippet separated by punctuation
- Phrase: part of a sentence consisting of a group of word without subject and verb
  e.g. The quick brown fox
- Clause: part of a sentence including a subject and a predicate
  e.g. She walks to school
- Lemma: canonical form of a words or a multi-word expression
  e.g. “run” is the lemma of “runs”, “running”, “ran” and so on
- Lexeme: set of all single words or multi-word forms that have the same meaning
  e.g. “run” “runs” “running”... made a lexeme, for example the “run” lexeme
- Stem: reduce a word to its root (stem)
  e.g. “chang” is the stem of “change”, “changing”, “changer”, “changed” and so on
- Bag-Of-Word: unordered set of words

Inoltre un testo può essere suddiviso in:

- Paragraph: portion of text consisting of a sequence of sentences

- Section: portion of text consisting of consecutive paragraphs

Preprocessing steps

- Cleaning: filtering steps to remove noise, errors and redundant content
- Tokenization: divide the raw text into units and sub-units
  e.g. from “the cat sat on the mat.” to “the”, ”cat”, ”sat”, ”on”, ”the”, ”mat”, “.”
  Strongly language-dependent
- Stop-word elimination: remove too frequent words with little semantic meaning
  Language-dependent
  Some models do not require stop word elimination anymore
- Part of speech tagging: annotate the text words with the corresponding role in the sentence
  Context and language dependent, relies on morphological word analyses
  e.g. “The cat sat on the mat” → “The(article) cat(noun) sat(verb) on(preposition) the(article) mat(noun)”
- Lemmatization and stemming: map words to the lemma or the stem version
  - Lemmatization require analyzing the sentence or document level context, it is context and language dependent and relies on morphological word analyses
  - Stemming is the simplified version of the lemmatization process, it requires lower computational complexity with an approximated result which is acceptable in most of the cases
- Remove special characters and extra spaces
- Case normalization: handle upper and lower cases
- Data format conversion
- Handle non-textual content

An example of text preprocessing pipeline is:

1. Basic filtering
2. Tokenization

3. Multiwords grouping
4. Stopword filtering

Syntactic parsing

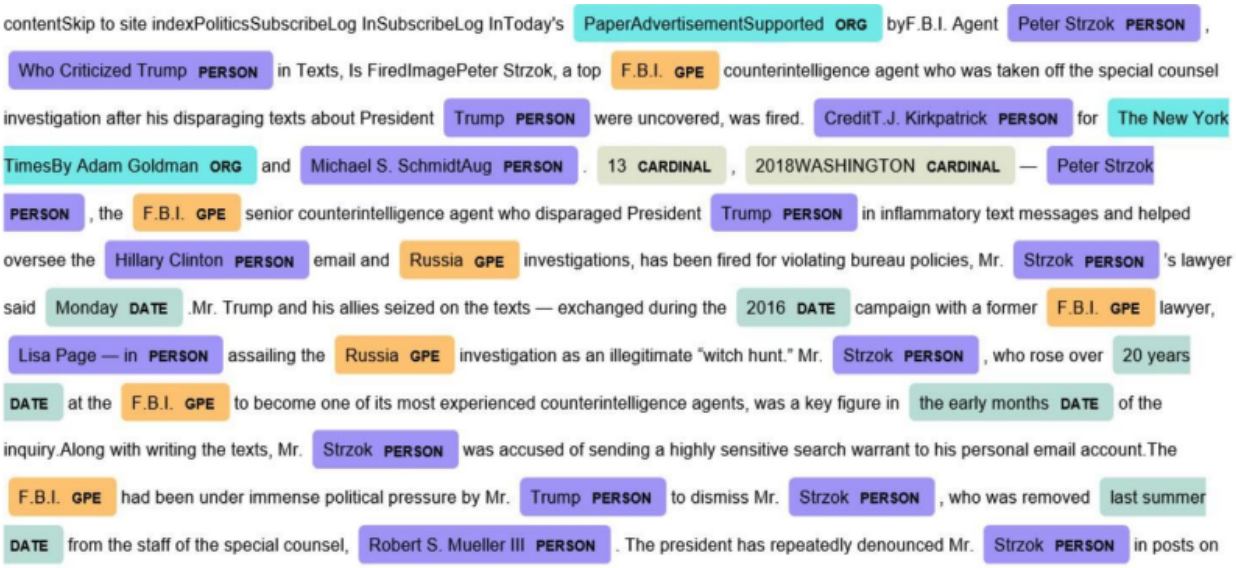
Syntactic analysis of the text according to certain grammar theory, it requires high computational complexity with a low quality in many real contexts

Shallow parsing

It is a simplified text parsing which syntactical analyzes only the parts of a text that are unambiguous

Named Entity Recognition

Locate and classify names in text, identify references to information units called entities



Named Entity recognition is usually combined with Word sense disambiguation that identifies the sense of a word with multiple meanings used in a sentence. NERD stands for “Named Entity Recognition and Disambiguation”

Text representation

- Fully unstructured document: raw text without sectioning and paragraphs
- Weakly structured document: text organized in sections paragraphs according to a predefined format
- Semi structured document: annotated with tags or with a overlay structure based on markup (like html)
- Structured model: for example a set of features summarizing the key properties of the text

Feature engineering

define a set of features an collect the corresponding values

- Dimensionality reduction: reduce the number of features to alleviate the issues related to high-dimensional data (like PCA)
- Feature selection: reduce the number of features to improve the performance of Machine Learning models
  - Unsupervised methods
  - Supervised methods

Topic Modeling

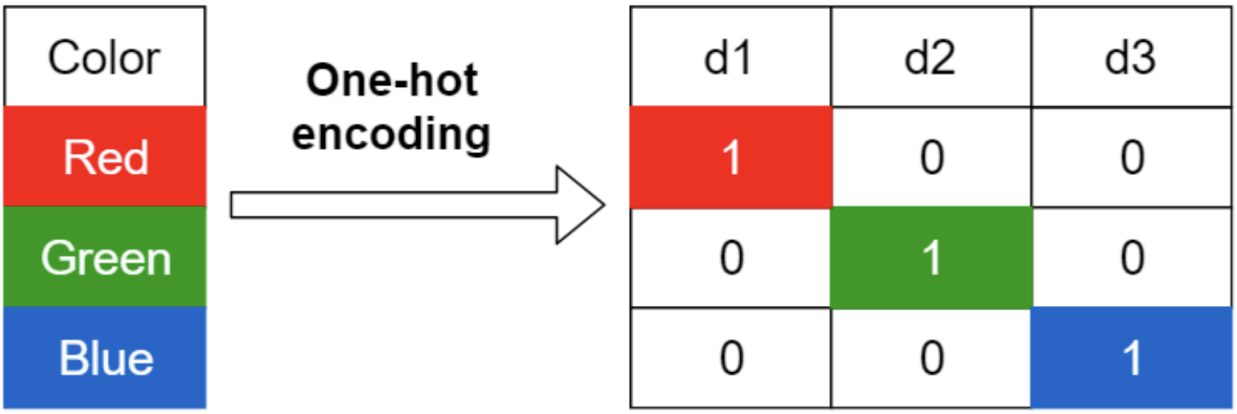
For the topic modeling we need a feature-based text representation. There are two different strategies:

- Occurrence-based representation: Textual features are computed on the frequencies of occurrences of the main textual units in larger text snippets
- Distributed vector representations of text: high-dimensional text representation extracted by means of neural network training

Here we focus on occurrence-based text representations

One hot encoding

One-hot encoding is a process that converts categorical variables (or word) into a binary (0 and 1) matrix representation. Each category value (word) is transformed into a unique binary vector.



- High Dimensionality: With large vocabularies, the resulting vectors become very high-dimensional, leading to computational inefficiencies
  Countermeasures:
  - Feature selection: Discard the less relevant features
  - Dimensionality reduction: Derive a lower-dimensional representation
- Sparsity: The vectors are sparse (mostly zeros), which can be memory inefficient.

One hot encoding is suitable for evaluating syntactical text similarities, but it is unsuitable for capturing semantic text similarities

A common way to define a weight for the importance of a word in a give document is Term-frequency - inverse document frequency

Term-frequency - Inverse document frequency (tf-idf)

Term-frequency

Term frequency measures how frequently a term appears in a document. The assumption is that the more a word appears in a document, the more important it is to that document.

$$\mathrm{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in document}}{\text{Total number of terms in document } d}$$

Inverse Document Frequency

Inverse document frequency measures how important a term is in the entire corpus. It help to diminish the weight of terms that appear very frequently across all documents, as these are less informative (like stopwords)

$$\text{IDF}(t, D) = \log \frac{\text{Total number of documents in the corpus}}{\text{Number of documents containing the term } t}$$

TF-IDF

TF-IDF is the product of TF and IDF. It balances the frequency of a term in a document with how common the term is across the entire corpus, giving higher weights to term that are frequent in a document but rare across the corpus

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

TF-IDF has some limitation as unbounded term frequency and in general is a very simple approach

Best Matching 25

The idea is to introduce the corpus-level stats in the TF-IDF formulation as:

- $Avgdl$: average document length over all the documents in the collection
- k_1: term frequency saturation (free parameter)
- b: penalty score associated with the document length (free parameter)

$$\text{BM25}_{i,j} = \text{IDF}_i \cdot \frac{\text{TF}_{i,j} \cdot (k_1 + 1)}{\text{TF}_{i,j} + k_1 \cdot (1 - b + b \cdot \frac{d_j}{Avgdl})}$$

Alternative scoring function

- Log normalization
- Frequency max
- Double normalization

Topic Modeling

Topic modeling is a methods used in NLP to discover hidden or latent topics in a large document. It helps to identify patterns and underlying structures within large volumes of text.

The number of topics covered by a document is likely small

Traditional approaches

The traditional approaches are based on occurrence-based text representation. The distribution of words in each document is expressed as weighted combination of concepts derived from the occurrence-based text representation

- Latent Semantic Indexing LSI: the underlying concepts and the corresponding weights are derived from the Singular Value Decomposition (SVD).
- Latent Dirichlet Allocation LDA: Derive document-topics and topics-terms probabilities distributions using generative model

LSI

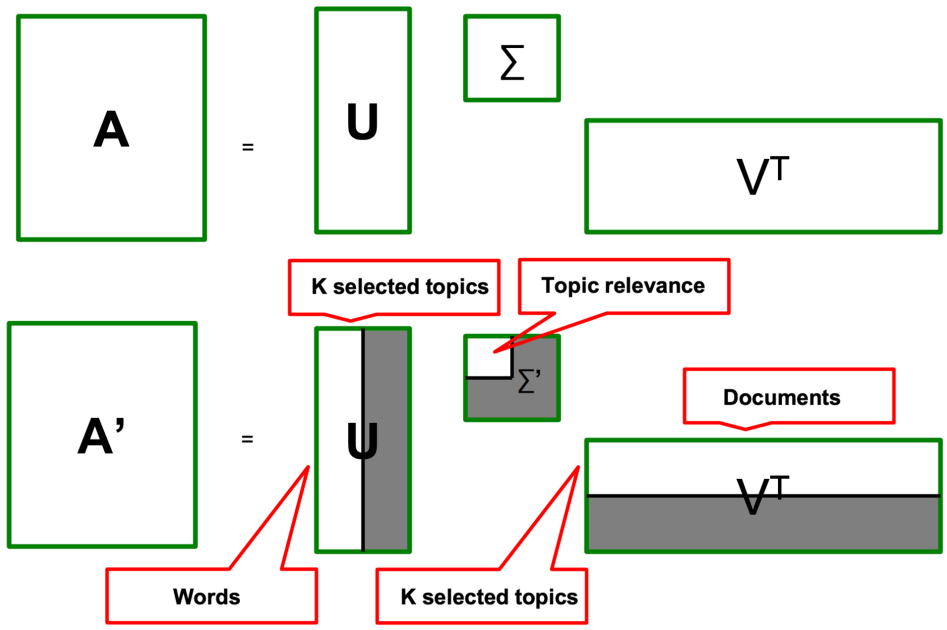
It's a mathematical approach used to represent occurrence-based text representation in lower dimensional latent space. it is instrumental for topic detection

It's based on reduced SVD. compute a low-rank approximation of the document-word matrix

Each eigenvalue in A′ is representative of a salient document concept

The first k rows of V^{T} represent the relevance of each document with respect to the shortlisted top-k concepts

The first k columns of U represent the relevance of each word with respect to the shortlisted top-k concepts



The k value for reduced SVD is analyst-provided. The heuristic approach define σ_k as the smallest singular value that is above the half of the highest one($\sigma_k > 2\sigma_1$)

$$
\underset{R}{\begin{bmatrix} 5 & 5 & 0 & 0 & 1 \\ 4 & 5 & 1 & 1 & 0 \\ 5 & 4 & 1 & 1 & 0 \\ 0 & 0 & 4 & 4 & 4 \\ 0 & 0 & 5 & 5 & 5 \\ 1 & 1 & 4 & 4 & 4 \end{bmatrix}} \approx \underset{U}{\begin{bmatrix} -0{,}27 & 0{,}55 & -0{,}78 & 0 \\ -0{,}29 & 0{,}47 & 0{,}44 & -0{,}71 \\ -0{,}29 & 0{,}47 & 0{,}44 & 0{,}71 \\ -0{,}45 & -0{,}29 & -0{,}01 & 0 \\ -0{,}56 & -0{,}36 & -0{,}02 & 0 \\ -0{,}50 & -0{,}18 & -0{,}05 & 0 \end{bmatrix}} \underset{\Sigma}{\begin{bmatrix} 13{,}74 & 0 & 0 & 0 \\ 0 & 10.88 & 0 & 0 \\ 0 & 0 & 1.36 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}} \underset{V^T}{\begin{bmatrix} -0{,}32 & -0{,}32 & -0{,}52 & -0{,}52 & -0{,}5 \\ 0{,}63 & 0{,}63 & -0{,}25 & -0{,}25 & -0{,}29 \\ -0{,}02 & -0{,}02 & 0{,}41 & 0{,}41 & -0{,}82 \\ 0{,}71 & -0{,}71 & 0 & 0 & 0 \end{bmatrix}}
$$

For example in the image above k is 2 because $\sigma_1 = 10.88 > 2 \cdot \sigma_2 = 2 \cdot 1.36$. The document-level topic relevance scores for the document 1 is equal to the rows of the topic score matrix Σ moltiplicated by the columns of the document 1. So for the topic 1 we have $\mathrm{r1} \times \mathrm{c1} = 13.74 \times -0.32$ and for the topic 2 $\mathrm{r2} \times \mathrm{c1} = 10.88 \times 0.63$

LDA

It's a probabilistic topic model. The idea is that the documents are mixtures of multiple topics and a topic is a distribution over a fixed vocabulary. Each topic is

characterized by a set of words that have a higher probability of occurring in that topic.

LDA introduces two dirichlet distributions:

- Document-topic distribution where each document is assumed to have a distribution over topics, which is drawn from a Dirichlet distribution
- Topic-word Distribution: Each topic has a distribution over words, which is also drawn from a Dirichlet distribution

The Generative Process

LDA assumes the following generative process for each document in the corpus:

- Choose the Number of Words (N):
  - For each document, choose the number of words N (this is usually observed from the data).
- Choose a Topic Distribution:
  - For each document d, choose a topic distribution θ_d from a Dirichlet distribution with parameter α .
- Choose a Topic for Each Word:
  - For each word w in document d:
    - Choose a topic $z_{d,n}$ from the topic distribution θ_d.
    - Choose a word $w_{d,n}$ from the topic's word distribution $\beta_{z_{d,n}}$.

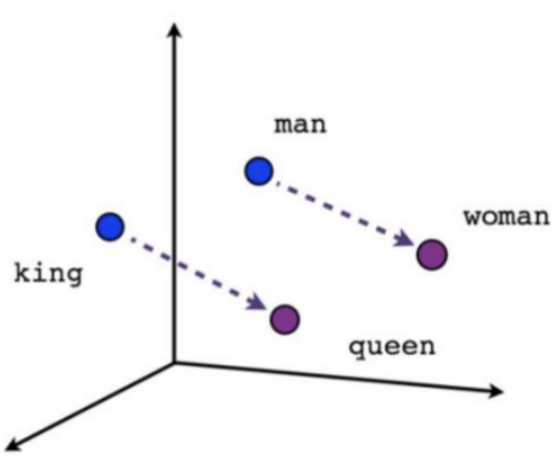
The Parameters:

- α : The Dirichlet prior on the per-document topic distribution. It influences the distribution of topics within documents. A higher α means documents are likely to contain a mixture of most topics.
- β : The Dirichlet prior on the per-topic word distribution. It influences the distribution of words within topics. A higher β means topics are likely to contain a mixture of most words.

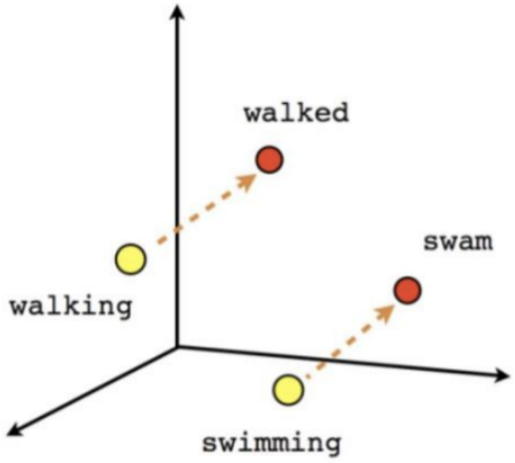
Word Embeddings

Word embeddings are a type of word representation that allows words to be represented as vectors in a continuous vector space. These vectors capture semantic meaning by placing semantically similar words close to each other in this space.

The vectors contain set of real number indicating the position of the word in the space, in order to achieve a representation where the words with some semantic relation are closet to each other in the space



Male-Female

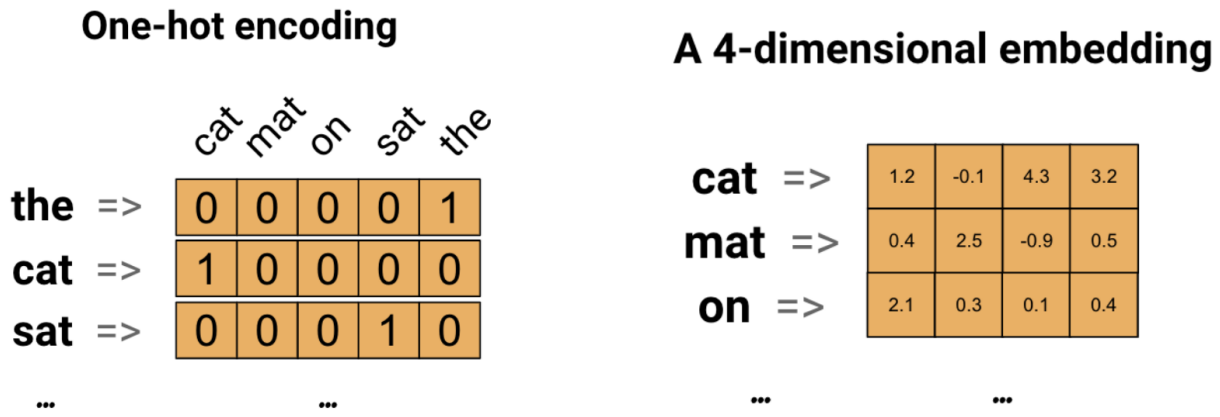


Verb tense

Each textual unit is mapped to a dense vector of real numbers. The vector representation of text embeds textual similarities into the vector representation, so semantically related units are mapped to similar vectors

For word embeddings self-supervised text representation enable better supervised learning on large datasets.

Multiple architectures have been proposed to embed single words into a fixed-size vector representation like one-hot encoding and 4-dimensional embedding



Key Idea

The key idea behind feature extraction in embedding models is that given a target word we look for the semantically related words among those appearing nearby

A fixed sliding window denotes the context of a target word, typically from 3 to 5 preceding or subsequent to the target word

the key steps:

1. Collect a large document corpus (like wikipedia)
2. Define a word-level vocabulary
3. Slide over the text to build training samples
4. Train an ad hoc network
5. Learn word vectors from the network

Word2Vec

Word2Vec is a pioneering approach to efficiently learn word embeddings. W2V transforms words into numerical vectors, known as embeddings. These embeddings aim to capture semantic and syntactic relationships between words based on the idea that words used in similar contexts have similar meanings. W2V can be implemented using two main Architectures:

- **Skip-Gram**: predicts the surrounding context words for a given target word, which effectively means that for each word in your training corpus, you use it as an input to predict the words around it.
  Process:

  1. **Input**: For a given target word, the model inputs a one-hot encoded vector representing this word.
  2. **Output**: The model outputs multiple predictions, one for each context position. For example, if the context window is 2, it predicts the two words before and the two words after the target word.

  Positive and negative example:

  1. **Positive Examples**: These are the actual words that appear in the context of the target word within the specified window size. For instance, if the target word is “bank” and the window includes “river” on the left and “money” on the right, these are treated as positive examples.
  2. **Negative Sampling**: To improve training efficiency and effectiveness, negative examples are introduced. There are random words selected from the vocabulary that are not part of the context of the target word. The idea is to teach the model what is not a suitable prediction, which sharpens its ability to understand contexts.

  Training: The skip-gram model uses a technique called negative sampling, where for each positive example, several negative examples are randomly chosen. The model updates its weights not only to increase the probability of the positive words but also to decrease the probability of these negative samples.

- **Continuous Bag Of Words CBOW**: uses the surrounding context words to predict the target word. It’s essentially the reverse of Skip-gram
  Process:

  1. **Input**: The input is the average (or another type of combination) of the one-hot encoded vectors of context words
  2. **Output**: A single prediction for the target word

  Positive and negative example:

1. **Positive examples**: In CBOW, the positive example is the target word that the context words are trying to predict.
2. Negative sampling: As with skip-gram, negative sampling is used in CBOW to provide negative examples which the model learns to not predict in the given context.

Training: During training, the model learns by adjusting the weights to maximize the probability of the actual target word while minimizing the probability of the negatively sampled words

Negative Sampling

Negative Sampling is an approach that transforms the problem of predicting the distribution of context words into a more manageable problem of binary classification:

1. Positive vs negatives:
    a. Positive Examples: Fro a given target word and a real context word within the specified context window , the desired output is “true”, indicating that this context word is indeed correct for the target word.
    b. Negative Examples: Conversely, for the same target word, several examples of the words that are not in the context are randomly generated. For These, the desired output is “false”, indicating that these words are not appropriate contexts for the target word.
2. Model Operation:
    a. For each target-context pair (positive), the model also generates several negative pairs
    b. The model then trains on these pairs, adjusting its weights not only to correctly predict the positive context but also to recognize and reject the negative contexts.

During training with Negative Sampling, W2V essentially trains a binary classifier for each pair of words. For each pair:

- The model calculates the probability that the context word is indeed a valid context for the target word.
- This probability is compared against the “true” o “false” label, ad the model updates its weights to reduce the error in this binary prediction

Model

W2V has a simple architecture:

1. Input layer:
    - Input Encoding: The input is a one-hot vector representing a word in the vocabulary. The dimension of this vector matches the size of the vocabulary; for example, if the vocabulary has 10.000 words, the one-hot vector will be 10.000 elements long, with a 1 in the position of the input word and 0s everywhere else.
    - For skip-gram: the input is the one-hot vector of the target word.
    - For CBOW: The input is the average (or another combination) of the one-hot vectors of the context words.
2. Hidden layer:
    - No activation functions: Different from traditional neural networks, there are no activation functions applied to the neurons in the hidden layer. The role of this layer is simply to transform the input vector into a smaller output vector, corresponding to the desired embedding size.
    - Weights of the hidden layer: This layer has a weight matrix W with dimensions *vocabulary_size x embeddings_size*
3. Output layer:
    - **Softmax**: The output layer uses the softmax functions to transform the logits, produced by multiplying the output of the hidden layer with a second weight matrix W’, into probabilities. The dimension of matrix w’ is *embedding_size x vocabulary_size*
    - **Probabilities**: The output is a vector the length of the vocabulary, where each element represents the probability that a specific word from the vocabulary is the context word (skip-gram) or the target word(CBOW)

Extracting Embeddings

After the model is trained:

- In skip-gram, you can obtain embeddings directly from the weights associated with the input layer
- in CBOW, the embedding salso come from the input layer's weights, even though the training process involves predicting the target word from the context.

Fast Text

FastText is a word embedding model developed by Facebook, it is similar to W2V but use subword n-grams instead of word to generate word embeddings. This means that instead of using only the whole word as input, FastText also considers sequential groups of characters that make up each word.

Key Details

1. Character N-grams: A word is split into multiple character n-grams. For example, for the word "cat" with 3-gram n-grams, the n-grams might be <ca, cat, at>, where < and > indicate the beginning and end of a word. This helps handle morphological variations like plurals, verb tenses, etc.
2. Model Training: Similar to W2V, FastText can use approaches like Skip-gram and CBOW, but instead of using just the whole word, it uses a representation that is the aggregate of the embedding of its n-grams. This allows FastText to be particularly effective for inflection-rich language and to handle out-of-vocabulary words during testing, as it can assemble embeddings for unseen words during training using known n-grams.
3. Skip-gram and CBOW: In FastText, both skip-gram and CBOW models are modified to work with character n-grams.
    a. Skip-gram: the goal is to predict the context n-grams given an n-gram of the target word

b. CBOW: the character n-grams of the context words are used to predict the target word.

Advantages

- Handling of Rare and Out of vocabulary Words: Since it can break down into n-grams, FastText can create reasonable embeddings for rare or unseen words during training
- Morphological Richness:It is particularly useful for languages with rich morphology like Turkish, Finnish or Arabic, where words can have many different forms.

GloVe

GloVe constructs a global co.occurrence matrix that counts how often words appear together in a specific context within an entire corpus. This matrix provides a global view of word frequencies that can reveal semantic and syntactic relationships.

1. Co-occurrence matrix: A matrix is constructed where each element X_{ij} represents the number of times word j appears in the context of word i across the entire corpus. The “context” of a word can be defined ad a specific window of words around it.
2. Model Objective: The goal of GloVe is to learn word vectors w_i and w_j such that their dot product $w_i^T \cdot w_j$ equals the logarithm of their co-occurrence probability, $\log(X_{ij})$.
3. Loss Function: The loss function used to optimize the vectors is a weighted loss that minimizes the difference between the dot product of the word vectors and the logarithm of the co-occurrence frequencies. It is weighted to give less importance to very frequent or very rare co-occurrences.

Differences

- Global vs local Analysis: While Word2Vec and FastText generate embeddings based on local learning from specific contexts, GloVe uses a global approach,

aggregating statistical information from across the corpus.

- Data Dependency: GloVe requires the construction of a large co-occurrence matrix and works best with a large corpus that provides sufficient statistical information, whereas W2V and FastTest can be more flexible with smaller corpora
- Preprocessing: GloVe requires initial processing to build the co-occurrence matrix, which can be memory and time-intensive, especially with large vocabularies.

Advantages

GloVe is capable of capturing both global statistical relationships and local relationships between words, making it effective at grasping subtle semantic nuances and usage patterns of words. Moreover, it has shown to perform well on analogy tasks and to be competitive with other embedding models in various NLP applications.

Sentence Embeddings

Doc2Vec

Doc2Vec is an extension of the W2V model that allows for generating embeddings for entire documents, which can be sentences, paragraphs, or longer texts. The main idea behind D2V is to learn a vector representation that captures the context of whole block if text, going beyond the simple aggregation of word embeddings.

Two main architectures:

1. Distributed Memory (DM): This architecture is similar to CBOW in W2V. The model predicts a target word based on a context of surrounding words and a single paragraph id. The paragraph id acts as a memory that remembers the context or content of the entire document. In practice, both the words and the paragraph id are used to predict the next word.
2. Distributed Bag of Words (DBOW): This architecture is similar to Skip-Gram in W2V, but with a key difference. Instead of using the paragraph id to predict

words from a surrounding context, DBOW uses the paragraph id to directly predict words from the document, disregarding the surrounding word context. This means that during training, random words are selected from the document, and the paragraph id is used to predict them

Training Doc2Vec

Training Doc2Vec follows a process similar to that of W2V, with the addiction that the model must also learn the paragraph ids alongside word embeddings. Here are the key steps:

1. Initialization: The paragraph id and word embeddings are initialized randomly.
2. Sliding Window: Depending on the architecture (DB or DBOW) the model scans through the documents using a context window to predict words or using the document id to predict random words from the document
3. Optimization: Stochastic gradient descent is used to minimize the prediction error, adjusting both the word embeddings and the paragraph

Sent2Vec

Sent2Vec is a methods for generating embeddings for entire sentences, it can be seen an extension of CBOW.

How it works

1. Input: Sent2Vec takes entire sentences as input rather than individual words.
2. Words and n-gram model: Sent2Vec consider not only words but also n-grams to capture local context within the sentence. This approach helps retain some of the grammatical structure and contextual meaning that might be lost when only considering single words.
3. Sentente Embeddings: the model generates embeddings by aggregating the embeddings of words and n-grams. The aggregation function can vary; a common approach is the weighted average of the embeddings of all words and n-grams present in the sentence.

InferSent

InferSent is a model for learning universal sentence representations. It's distinct from the other models in that it leverages the power of supervised learning.

InferSent uses a supervised approach to sentence embedding, training on labeled dataset where pairs of sentences are given along with annotations on whether the sentences are entailing, contradictory, or neutral with respect to each other. This approach is based on the hypothesis the understanding whether one sentence can infer another is a strong indicator of semantic understanding

Text: "If you help the needy, God will help you"

- Positive TE (text entails hypothesis) hypothesis: "Giving money to a poor man has good consequences"
- Negative TE (text contradicts hypothesis) hypothesis: "Giving money to a poor man has no consequences"
- Neutral TE (text does not entail nor contradict hypothesis) hypothesis: "Giving money to a poor man will make you a better person"

InferSent typically uses BiLSTM (Bidirectional Long Short-Term Memory) network as its core architecture to encode sentences. The choice of BiLSTM allows the model to capture information from both the beginning and the end of the sentence, integrating context effectively

The model is primarily trained on the Stanford Natural Language Inference(SNLI) dataset, which consists of pairs of sentences labeled with their relationship. By learning to predict these relationships, InferSent develops an ability to encapsulate complex semantic properties of sentences

After processing sentences through the BiLSMT, InferSent applies a max pooling strategy to derive a fixed-size sentence embedding from the variable-length outputs of LSTM. This embedding captures the essential semantic features of the sentence.

The embedding learned by inferSent are designed to be universal, meaning they can be successfully applied to many different NLP tasks without the need for task-

specific tuning

Attention Mechanism

Attention is a mechanism that enables language processing models, like Transformers, to handle and interpret sequences of words mode effectively.

Attention helps the model focus on specific parts of a sentence while processing or generating text, thereby improving language understanding and production.

The attention mechanism assigns a **weight**, or "attention score", to each element of an input sequence. These weights determine how much part of the input should be considered when processing a particular part of the output. Practically, this allows the model to decide which information is most relevant in a given context, making the processing more similar to how humans read and understand texts.

Basic Attention

Basic attention in NLP consist of three main components:

1. Query: A vector what represents the current element for which the output is being calculated.
2. Key: A vector associated with each input element that will help determine how important each element is to the query.
3. Value: A vector associated with each input element that contributes to the final output based on the calculated attention weights

Computing Basic Attention

1. Transformation into Query, Key, Value: Input elements are transformed into query, key, and value vectors using specific weight matrices learned during the model's training
2. Calculation of Attention Scores: Attention scores are calculated by comparing every query with all keys. The most common method is the dot product, but other functions can be used (such as additive or concatenation)
3. Normalization of Scores: The calculated scores are normalized using softmax function to turn them into a probability distribution. This step ensures that the

scores are positive and sum to one, making the attention mechanism a kind of “weighted average”

4. Output Based on Value: The value vectors are combined based on the calculated attention weights, resulting in an output vector that is weighted sum of the values

Goals of Attention

- Performance improvement
  - No recurrence → facilitate parallelization
  - Focus on specific text portions
  - Few training steps
- Overcome the vanishing gradient and explosion
  - Shortcut to faraway states
  - Facilitate long range dependencies
- Model explainability
- Attention distribution indicates on which sentence portion the decoder is focusing on

Self Attention

Self attention is a mechanism that allows each position in a sequence to attend to all positions within the same sequence. This is particularly useful in understanding the entire context of a sentence or document, as it captures dependencies without regard to their distance in the text.

- Query(Q): Represents the element that is attempting to find relevant information in other parts of the sequence
- Key(K): Represents elements that are being compared against the query to establish relevance

- Value(V): Represents elements that will be used to construct the output based on the calculated attention weights

Process of Self-Attention

The self-attention process can be broken down into the following steps:

1. Transformation into Queries, Keys, and Values: Each input token is transformed into the sets of Q, K, and V vectors through multiplication by learned matrices specific to Q, K, and V
2. Scoring: The attention scores are computed by taking the dot product of the query with all keys. The score reflects how much focus to place on other parts of the input sequence for each position. Essentially, it measures the compatibility or relevance between different position in the input sequence.

   $$\text{score}_{ij} = \text{Query}_i \cdot \text{Key}_j^T$$
3. Normalization: The scores for each query are then normalized using the softmax function, which converts them into probabilities that sum to one. This step ensures that the scores are comparable across different position and stabilize the gradients in the networks
4. Weighted Sum: The output for each position is calculated as a weighted sum of the value vectors, weighted by the attention scores. This allows the model to focus more on the most relevant parts of the input

   $$\text{output}_i = \sum_j (\text{weights}_{ij} \times \text{Value}_j)$$
5. Output representation: the resulting outputs are then passed on, often through further layers of processing to refine and utilize the gathered information

Cross attention

Cross-attention allows a model to integrate information from an input sequence into an output sequence. Practically, it enables each element of the output sequence to “look at” and “consider” all elements of the input sequence to determine which are the most relevant for producing the correct output

- Query(Q): These are derived from the sequence that is currently being generated
- Key(K) and Value(V): These come from the input sequence. Keys help determine which parts of the input are relevant to the output, while values are the actual information used to construct the output

Process of Cross-Attention

Similar to basic and self-attention:

1. Generation of Query, Key, and Value: Keys and values are generated from the input sequence, while queries are generated from the output sequence or its intermediate representation
2. Calculation of Attention Scores: The score are calculated by taking the dot product between the queries and the keys. This score determines the relative importance of different parts of the input for a given element of the output

   $$\text{score}_{ij} = \text{Query}_i \cdot \text{Ket}_j^T$$

3. Normalization of Scores: The scores are normalized using a softmax function to turn them into a probability distribution

   $$\text{weights}_{ij} = \text{softmax}(\text{score}_{ij})$$

4. Weighted Sum for output: The output is calculated as a weighted sum of the value vectors, based on the weights derived from the attention scores. This result directly influence the generation of the output or the next step

   $$\text{output}_i = \sum_j (\text{weights}_{ij} \times \text{Value}_j)$$

Cross-Attention is particularly useful in scenarios where information need to be transferred or adapted from one sequence to another like machine translation or question answering

Multi-Head Attention

Multi-head attention involves running several attention mechanisms (or “heads”) in parallel. Each head learn different aspects of the data, allowing the model to pay attention to different parts of the sequence for different reasons, thereby capturing a richer diversity of information.

It has the same component of basic attention: Query (Q), Key (K), Value (V). These components are replicated across multiple heads, each with its own set of weight matrices.

Process of Multi-Head Attention

1. Linear Transformations: Each head applies its own set of linear transformations to the input vectors to produce separate sets of Qs, Ks and Vs. This means that for each head, the input vectors are transformed by different, learned weight matrices.
2. Attention Calculation: Each head calculates attention independently using its Qs, Ks and Vs. The attention calculation in each head follows the usual steps:
   a. Calculate dot products of Qs with Ks to get raw scores
   b. Apply softmax to normalize the scores to probabilities
   c. use these probabilities to get a weighted sum of the values

   $$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

   Here
   d_k is the dimension of the key vectors, and the division $\sqrt{d_k}$ is a scaling factor that helps stabilize training
3. Concatenation of heads: The outputs from all heads are concatenated. This concatenated vector contains information from different representational subspaces, as each head might be focusing on different parts of the input.
4. Final Linear Transformation: The concatenated output then goes through one final linear transformation to combine the diverse information captured by all the heads into a single output vector

Contextualized embeddings

Classic word and sequence embeddings lack of Context sensitivity, they cannot capture changes in meaning based on the context in which words are used. They also do not handle well the polysemy where a word has multiple meanings based on context and also ignore the positional information

Contextualized embeddings overcome many of the limitations of traditional word embeddings by generating representations of words that dynamically vary based on their specific context.

They better understand context, they can distinguish between different uses of word in the same sentence or in different sentences, the are also particularly useful in tasks like named entity recognition, text comprehension and machine translation, where context plays a fundamental role

Additionally, the position of words within a sentence also carries significant information that is essential for understanding the syntax and structure of sentences. Traditional embeddings do not inherently account for the order of words, which can lead to ambiguities in parsing and generating sentences. In contrast, models like Transformer incorporate position embeddings or mechanisms like attention that inherently consider the order of words, thereby capturing more natural and syntactically coherent representations of text.

Language Model

Forward Model

The forward LM predicts the next word in a sequence based on the previous words alone. For instance, got the phrase “The cat eats”, a forward Lm would predict the next word “eats” after seeing “The cat”

How it works:

- Training: During training, the model learns to estimate the probability of a subsequent word given a sequence of preceding words. This is done by maximizing the likelihood of the following words in sentences from the training dataset.

- Usage: Once trained, the model can generate text by continuing a given phrase or be used to calculate the probability of a word sequence

Applications:

- Automatic text generation
- Auto-completion in writing systems and chatbots

Backward Model

The Backward ML operates in reverse to the forward LM, predicting the preceding word of a sequence based on the subsequent words.

How it works:

- Training: similar to the forward LM, but instead of learning to predict the next word, it learns to predict the previous word.
- Usage: This type od model can be useful in text analysis tasks where future context is informative

Applications:

- Enhancing context understanding in text analysis when the subsequent context is relevant
- Retrospective text analysis applications, where following context can provide clarification on previous elements

Bidirectional Language Model

Bidirectional LMs combine the strengths of both forward LMs and backward LMs to have a more comprehensive view of a word's context within a sequence. They use information from both preceding and following words to make more accurate predictions.

How it works:

- Training: The model is exposed to both, the preceding and following context during training, thus learning to predict words based on a dull understanding of the sentence.

- Usage: It can be used for tasks that require a deep understanding of text, such as part-of-speech tagging, named entity recognition, and sentiment analysis

Applications:

- BERT and other Transformer-based models use bidirectional attention to analyze all available context, significantly improving performance across numerous NLP tasks
- Deep semantic analysis, speech recognition and question-answering system

ELMo

ELMo (Embeddings from Language Models) is a context-based word embedding model. Unlike earlier models like W2V or GloVe, which generate a single static representation for each word in the vocabulary, ELMo considers the context to generate dynamic embeddings. This means that for every word, ELMo produces different representation depending on its specific usage in sentence.

Architecture

ELMo uses a bidirectional LSTM neural network to process text. The network is trained as a language prediction model, meaning it learns to predict the next word in a sentence based on the previous and following words.

1. Bidirectional Language Model (BiLM):
    a. Input Layer: The text is initially preprocessed with tokenization, and each token is mapped to a initial embedding. These initial embeddings are generally pretrained and static, similar to those in W2V
    b. LSMT Layer: Following the initial embeddings, the text is processed by two separated LSTM network:
        i. Forward LSTM: Processes the text from left to right, processing each word based on the information from previous words.
        ii. Backward LSTM: Processes the text from right to left, processing each word based in the information from following words.

c. Output Layer: Each LSTM produces a series of hidden states for each word, capturing contextual information.

2. Contextual Embeddings Extraction:
3. For each word in the text, ELMo extracts and combines the hidden states from LSTM layers at various levels. For example, it considers hidden states from the last layer of each direction and also from intermediate layers, if present.
4. The combination of these states is typically done through a weighted sum, where the weights are learned specifically for the NLP task to which the model is applied.

ELMo can be fine-tuned to specialize the word embeddings to the task under analysis:

- Question answering
- Sentiment analysis
- Named entity extraction
- Semantic role labeling
- Conflict resolution

ELMo utilizes a self-supervised learning method for its training.

Sentence encoding

Seq2Seq models are designed to transform an input sequence into an output sequence.

Applications:

- Machine translation
- Question answering
- Summarization
- Text paraphrasing

Architecture

- Encoder:
  - The encoder processes the input sequence and produces a context vector that encapsulates everything needed from the input sequence to generate the output.
  - Typically, a RNN, LSTM or GRU is used to capture temporal dependencies in the input
- Decoder:
  - The decoder receives the context vector from the encoder and begins generating the output sequence, step by step. At each step, the decoder relies on the current hidden state and the previous output value to produce the next element of the output sequence
  - The decoder is also often implemented using RNN, LSTM or GRU.

Directional Embeddings

LSTM networks take into account variable sequence lengths, similar to a human readers they process the input from left to right, so the preceding context is taken into account to create sequence embedding.

They provide contextualized word-level representation that could be easily combined to create sentence embeddings

Backward

- Sentences must be processed word by word
  - This hinders parallel training
- In directional sequence to sequence models each state (word) is assumed to be dependent only on the previously seen states
  - This does not allow contextual information to be effectively retained for long sequences
- Words are processed in a single direction

Transformer

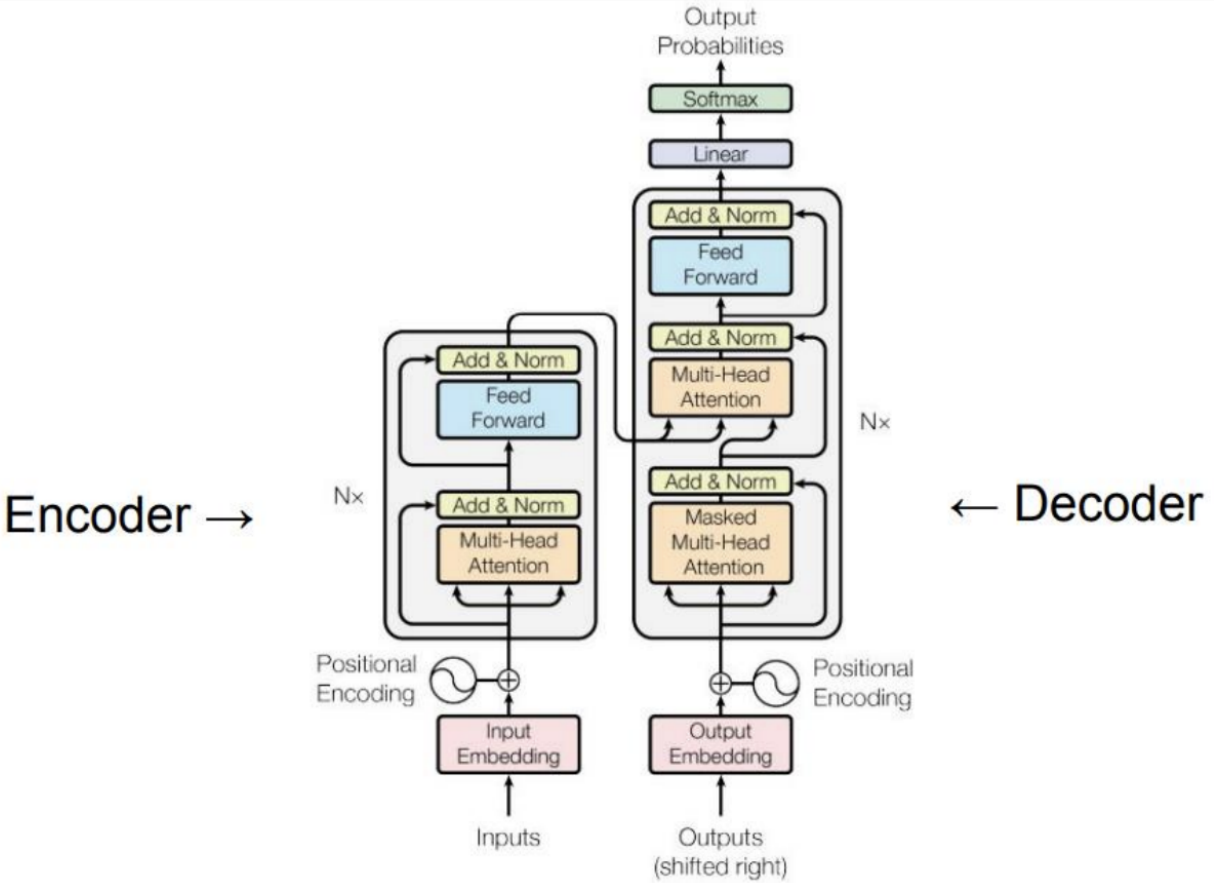
Transformers are a type of deep learning model architecture that has revolutionized NLP due to their ability to handle sequences of data without the need for a recursive or convolutional structure, relying entirely on attention mechanisms instead.

The goal is to minimize recurrence and maximize parallelization. It is an encoder decoder architecture based on attention with no recurrence. The key properties are:

- Feed the entire sequence of words in input to the encoder
- Compute the corresponding embeddings
- Process the whole input text in parallel
- With positional encoding it considers the relative word positioning

Architecture

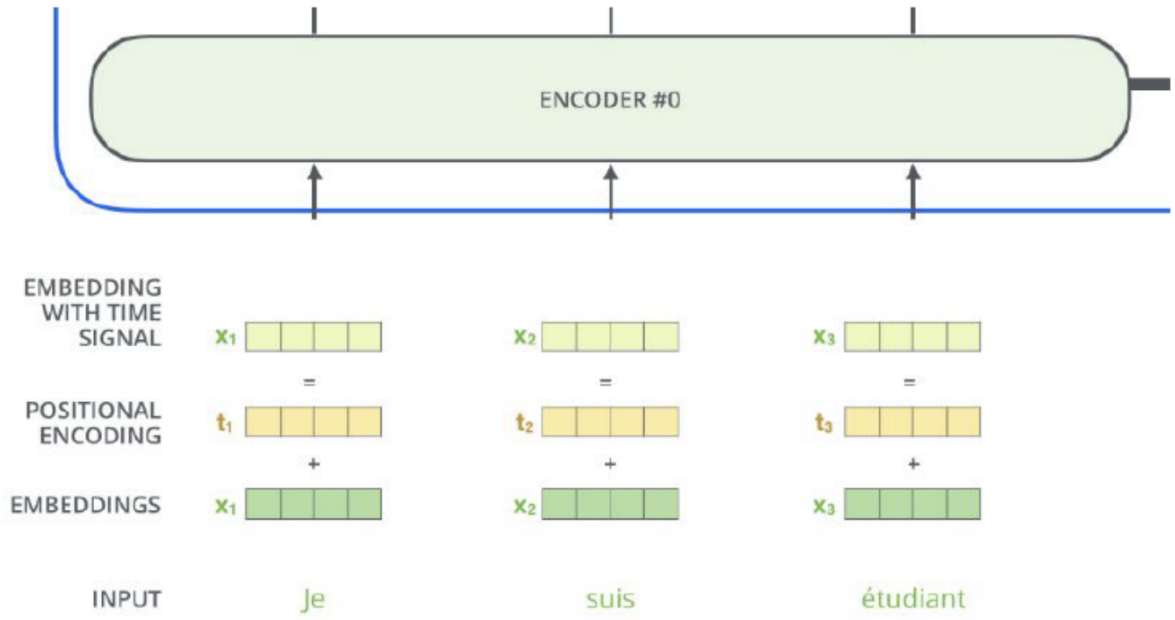
The architecture of the Transformer is based on an encoder-decoder model but with significant differences from classic seq2seq models that use RNN or LSTM



Encoder

The encoder in the Transformer model transforms the input text into a series of vector representations that contain all the necessary information for the decoder to produce the desired output

1. Input Embeddings: Initially, each token of the input text is converted into a vector via an embedding layer. This vector captures the semantic properties of the token
2. Positional Encoding: These embeddings are augmented with positional encoding to incorporate information about the token's position in the sequence. This helps the model understand the order and relationship between words, which is essential given the lack of recurrence in the network.
    a. $\text{PE}_{\text{position},2i} = \sin(\text{position}/10000^{2i/d})$
    b. $\text{PE}_{\text{position},2i+1} = \cos(\text{position}/10000^{2i/d})$



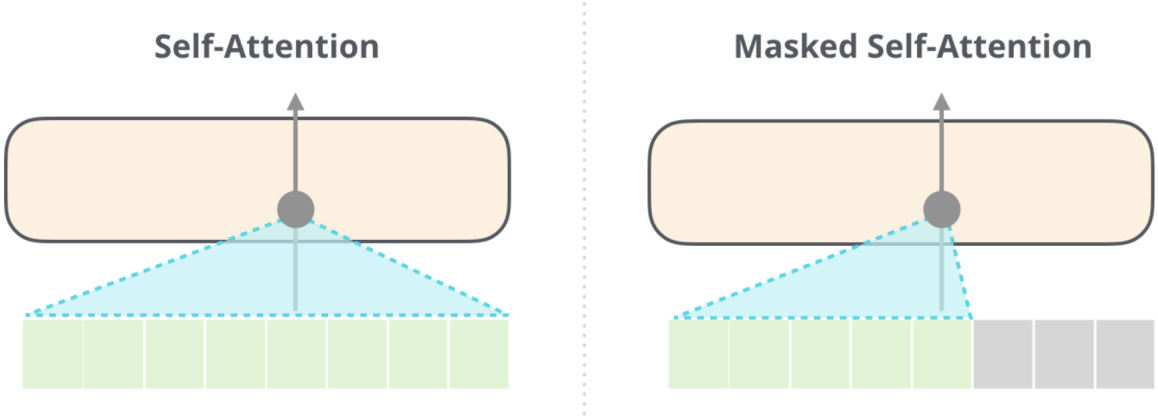
3. Multi-Head Attention: The encoder uses the multi-head attention mechanism to allow the model to focus on different subsets of other words in the sentence while processing a specific word. This enables capturing diverse contexts for each word and enhances the model's ability to handle dependencies between distant words in the sequence.
4. Feed-Forward Networks: Each layer of the encoder also contains a feed-forward network that applies the same transformation to each position separately and in parallel, contributing to the model's efficiency
5. Normalization and Residual Connections: Each sub-layer (such as attention and feed-forward layers) has a residual connection around it, followed by a layer of normalization. This design helps to prevent the vanishing gradient problem during training in very deep networks.

Decoder

The decoder in the Transformer generates the output based on the representations provided by the encoder

1. Output Embedding and positional Encoding: Similarly to the encoder, the decoder processes the output embeddings that are summed with positional encoding

2. Masked Multi-Head Attention: This version of attention in the decoder is masker to prevent the decoder from “seeing” future output during the prediction of a word. This ensures that the prediction of each word is based only on previous words and the representation provided by the encoder



3. Encoder-Decoder Attention: This layer of attention allows the decoder to focus on the entire output of the encoder. Using the encoder’s hidden states, the decoder can select parts of the input text that are most relevant for predicting the next token.
4. Feed-Forward Networks: As in the encoder, the decoder also contains feed-forward networks that further transform the output of the attention layers
5. Normalization and Residual Connections: Also in the decoder, each sub-layer has residual connections and normalization, which help maintain the effectiveness of learning through multiple layers

In summary, the encoder of the Transformer encodes the input text into a series of context-rich vectors, while the decoder uses this information, along with its own attention mechanisms, to generate the output text step-by-step.

BERT

BERT is based solely on the encoder part of the Transformer. This means it uses only the encoding part of the Transformer architecture, which is designed to process input sequences and generate rich, contextualized vector representations

of words. BERT is deeply bidirectional, meaning the context from both the left and right sides of a word is considered simultaneously to generate that word's embedding

Pretraining

BERT is pre-trained on two main unsupervised tasks:

1. Masked Language Model (MLM): During pre-training, 15% of the words in each sentence of the dataset are randomly replaced with a special `[MASK]` token. BERT's task is to predict the original words based on the context provided by the other non-masked words in the sentence. This allows BERT to learn a language representation that is deeply bidirectional, integrating context from both sides.
2. Next Sentence Prediction(NSP): BERT is also trained to predict whether a sentence B logically follows a sentence A. During training, the model is given a pair of sentences, and it must predict whether the second sentence in the pair is the logical follow-up to the first or if it is a random sentence taken from the corpus. This helps the model understand the relationships between sentences.

With just pre-training BERT can be used as a powerful sentence encoding model. The vector representations (sentence embeddings) generated can be used for many applications such as:

- Calculating semantic similarity between sentences
- Content-based and context-aware text retrieval

Pretraining requires 2.5ish billions sentences

Fine-tuning

Fine-tuning involves adapting the pre-trained model to specific NLP tasks. BERT can be fine-tuned relatively easily on a new task with a specific dataset

1. Adding Task-Specific Layers: Depending on the task, a specific output layer can be added to BERT. For example, for classification, a softmax layer might be added to categorize entries. For question answering task, layers that can predict the start and end of the answer in the text are added.

2. Training on the Task Dataset: During fine-tuning, the entire model(both transformer layers and task-specific layers) is trained on the task dataset. Thanks to the rich linguistic representation learned during pre-training, BERT require relatively few task-specific data and fewer epochs of training to achieve high performance

Fine-tuning requires ~1K sentences (depending on the task)

Tasks

- Sentence pair classification
- Single sentence classification/regression
- Question Answering
- Sentence Tagging
- ...

Encoder models

LongFormer

LongFormer is an improved version of the Transformer model optimized for processing long documents. While the standard Transformer model uses self-Attention, calculating relationship between all tokens in a sequence, this approach becomes computationally expensive as the document length increases.

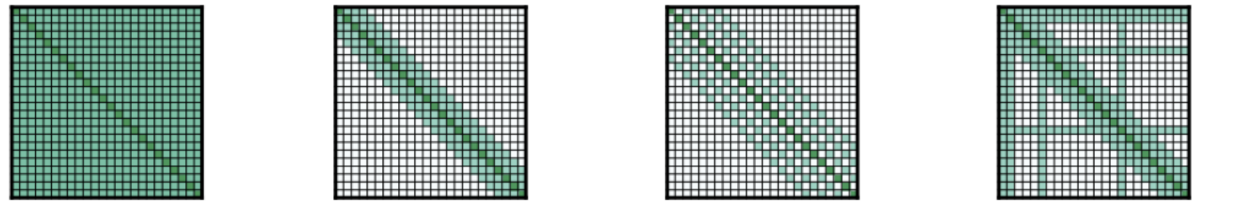
LongFormer introduces an attention mechanism that combines local attention and global attention:

- Local Attention: restricts the attention calculation to nearby tokens, reducing computational complexity.
- Global attention: selects key tokens (e.g. tokens representing important entities in the text) to participate in attention with all other tokens in the sequence. This allows the model to capture critical information without the computational cost of full-text attention

Global attention

Global attention is a strategy used in LongFormer and other models to reduce the complexity of full self-attention. Specific tokens are selected for “global attention”, meaning these tokens are attended to by every other token in the document, while the rest of the tokens only use local attention. This approach is particularly useful for maintaining computational efficiency while handling large-scale inputs.

- Key idea
  - sparsify the full self-attention matrix according to an “attention pattern” specifying pairs of input locations attending to one another

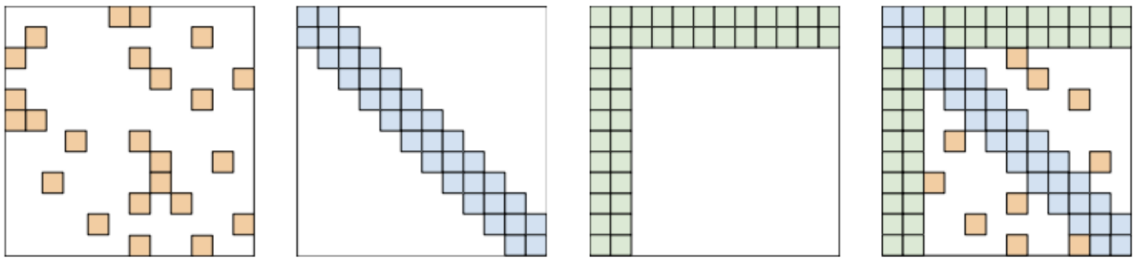


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

BigBird

BigBird is another Transformer based model designed to handle very long sequences. Similar to LongFormer, BigBird uses a mix of local attention, global attention and random attention to improve computational efficiency and document handling capability. Random attention adds a layer of flexibility, allowing the model to capture dependencies from unexpected parts of the document

- Transformer-based encoder that extends LongFormer using a sparse attention mechanism
  - Linear with the number of sequence tokens
- It considers
  - A set of g global tokens attending on all parts of the sequence.
  - All tokens attending to a set of w local neighboring tokens.
  - All tokens attending to a set of r random tokens.

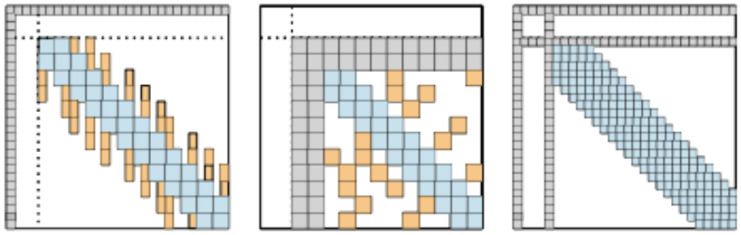


(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

LSG

Local Sliding window Global attention (LSG) is a variant that further extends the LongFormer model by integrating a sliding window mechanism for global attention. This approach allows the model to more effectively handle even longer documents and sequences, improving long-range context capture and further reducing computational complexity.

- **Local attention**: capture local context using a fixed length sliding window
  - Adopted also by LongFormer together with global attention
- **Sparse connections**: capture extended context by selecting an additional set of tokens following a set of rules
  - Not using random attention as in BigBird
- **Global attention**: it attends to every tokens across the sequence and all tokens attend to them
  - Like BERT and LongFormer



GPT

Architecture

GPT models are based solely on the decoder component of the Transformer architecture. Unlike the decoder in seq2seq model that has both attention and connections to an encoder, the decoders in GPT receive no external inputs other than the context they have already generated. They use a method of self-masked attention to prevent looking ahead to the parts of the text not yet generated, allowing the model to generate text one token at a time

Self-Masked Attention

In GPT architecture, self-masked attention is a type of attention that prevents the model from accessing future information in the sequence during text generation. This is essential to ensure that text generation is autoregressive, meaning each token generated depends only on the previous tokens, maintaining coherence and causality in the generated text

Training

GPT is trained in a self supervised learning. The model is trained to predict the next token in a sequence given all previous tokens. this process is similar to that of a traditional Language Model but on a much larger scale and with a far more powerful neural network. The ability to predict the next word allows GPT to generate coherent and surprisingly human-like text from a given input prompt.

Byte Pair Encoding

GPT utilizes Byte Pair Encoding, a form of tokenization that allows the model to handle vast vocabulary without needing to keep it entirely in memory. BPE works by identifying the most frequent pairs of bytes (or character) in the text and creating new tokens that represent them, reducing the vocabulary size and allowing the model to handle new or rare words as they appear in texts.

Versions

1. GPT-1: The first model introduced the idea of pretraining on a vast corpus of text followed by fine-tuning on specific tasks
2. GPT-2: Expanded on this concept using more data and a larger network, often foregoing fine-tuning on to show how a model can generalize from extensive pretraining alone
3. GPT-3: Took these ideas further with a 175 billion parameter architecture, demonstrating extraordinary capabilities across multiple tasks without specific fine-tuning, relying only on verbal instructions or “few-shot learning”.
4. GPT-4: This version has an higher memory limit, it can process up to 25.000 words, it is 10 times more advance than the previous version and combining both language and vision model to understand images

Tasks

- Text Generation
- Machine translation
- Question Answering
- Sentiment Analysis

- Content Summarization
- ...

Information retrieval

Information Retrieval (IR) is concerned with locating information within large volumes of unstructured data, typically text, in response to specific queries provided by users. The primary aim is to retrieve relevant information without retrieving excessive non-relevant data. IR is the process through which a system identifies and presents documents or text segments that meet a user's search criteria, evaluating their relevance relative to the query.

The next need to be preprocessed, common steps:

- Word tokenization
- Lowercasing
- Lemmatization/Stemming
- Stopwords removal

Boolean Retrieval

Boolean Retrieval model relies on using logical operators—mainly AND OR and NOT— to combine search terms. These operators allow users to refine and specify their searches by combining various terms and conditions.

- AND: This operator is used to ensure that every document in the search results contains all specified terms
- OR: The Or operator broadens the search to include documents that contain any of the specified terms
- NOT: This operator excludes documents that contain the specified term from the search

Indexing

Indexing in the context of IR is the process of creating a data structure that allows for efficient retrieval of information. This involves taking a large collection of text

documents and processing it to extract important or useful terms. During indexing, each document is analyzed to identify the words is contains.

The purpose of an index is to organize the information in a way that makes it quick and easy to locate. It acts as a map between the user's queries and the documents containing the terms that match these queries

Inverted Indexing

An inverted index is a specific type of index that is particularly useful in the field of IR. Unlike a traditional index, which might list pages on which a term appears, an inverted index turns this approach around. It lists for each term the documents in which it appears. this structure is highly efficient for search operations in large databases

Here's a simple example of how an inverted index works:

Consider three documents:

- Document 1: "Apple and orange are fruits."
- Document 2: "Banana is a yellow fruit."
- Document 3: "Apple pie and banana bread are delicious."

An inverted index for these documents might look like this:

- Apple: {1, 3}
- Orange: {1}
- Are: {1, 3}
- Fruits: {1}
- Banana: {2, 3}
- Is: {2}
- Yellow: {2}
- Fruit: {2}
- Pie: {3}
- Bread: {3}
- Delicious: {3}

Each word is followed by a set of document IDs in which that word appears. When query is processed, the system simply looks at the inverted index to quickly find out which documents contain the terms from the query, making the search process much faster and more efficient.

Benefits of Inverted Indexing

- Speed: Retrieving information becomes much faster because the system doesn't need to scan every document for the query terms; it directly accesses the list of relevant documents from the index.
- Scalability: Inverted indexes are highly scalable and can handle large volumes of data without a significant loss in performance
- Support for complex queries: they can support complex search queries including boolean queries and phrase queries efficiently

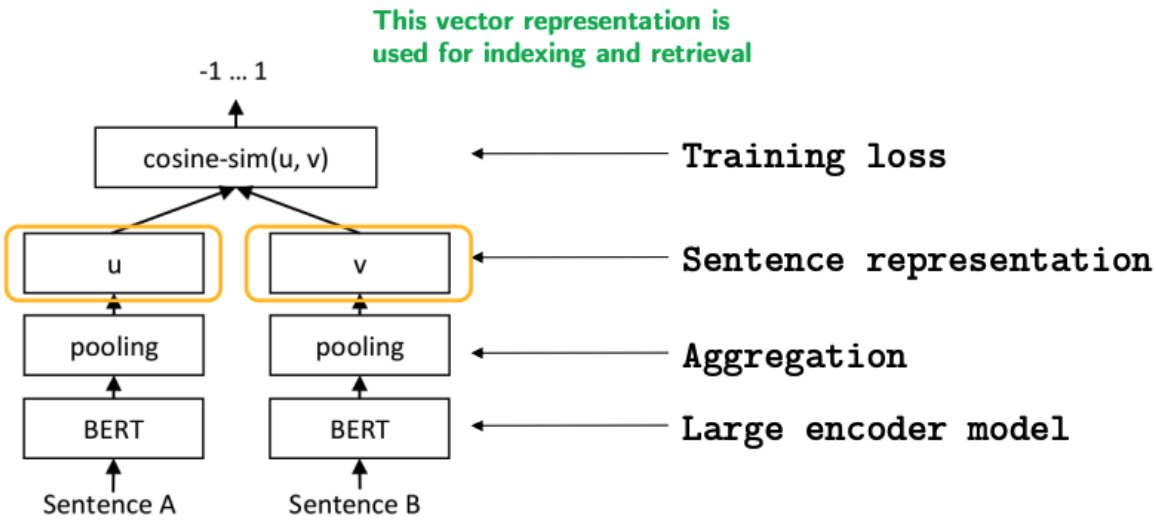
Neural IR

Standard IR models use the syntax of both the query and the document to compute the relevance score, here semantic information is lost and the order of the words is not taken into account.

Neural IR utilizes AI models to improve how systems understand and match user queries to documents. Neural IR models use the semantics of the query and the document to compute the relevance score. It leverages the representation given by modern neural architectures.

The idea is to have a dense vector representation:

- Not optimal: Word2Vec can be used to compute the semantic vectors. Vector representations can be computed for the query and the document computing the average of the word vectors
- Optimal: Transformers models are used to compute the semantic vectors that represent the entire sentence. The semantic similarity is computed by computing the cosine similarity between the vectors of the sentences

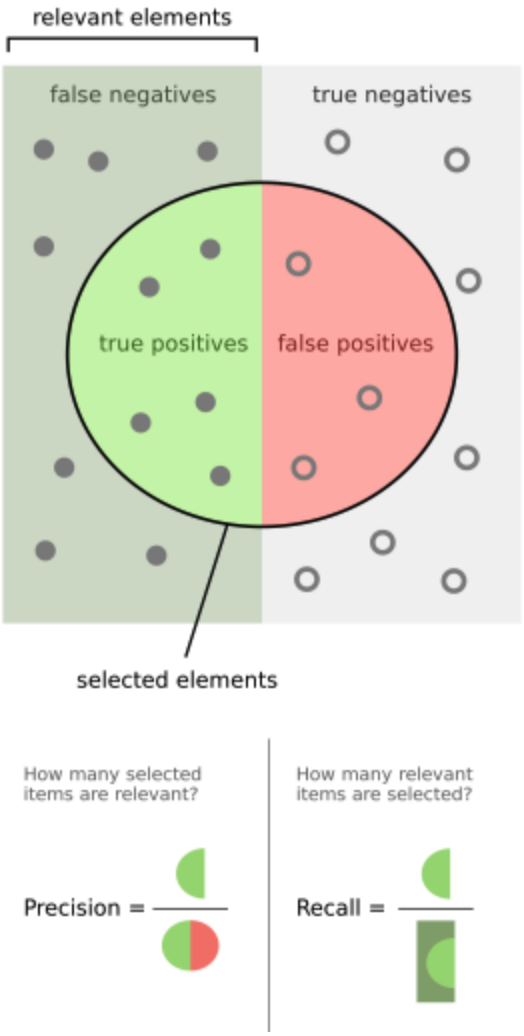


Traditional IR models use metrics like TF-IDF or similar, while Neural IR uses bert-score

Evaluation

Evaluating the effectiveness of an IR system is crucial to understanding its efficiency and improving its performance

- Precision: What fraction of the returned results are relevant to the information need?
- Recall: What fraction of the relevant documents in the collection were turned by the system?



Web Search engines

Web Search involves using search engined to locate information across the vast expanse of the internet. A search engine indexes the web using web crawlers, which follow links and gather data from web pages. This indexed data is then processed and stored in a way that allows it to be quickly retrieved through a user query

Graph modeling

Graph modeling is a powerful tool used in web search to represent the relationships between web pages. In this context, the internet can be visualized as a huge directed graph where each webpage is a node and hyperlinks between them are directed edges

- Nodes: In the web context, nodes represent web pages
- Edges: Directed edges in the graph represent links from one page to another. These are not merely connections but carry significant importance in determining the authority and relevance of a webpage

Link Analysis

Link analysis is a method used in graph theory to evaluate the relationships between nodes. in the context of web search, link analysis algorithms assess the quality and relevance of web pages based on how they are linked to each other

- Importance of Link analysis: Link analysis helps in determining which pages are authoritative and trustworthy. For instance, a page with many incoming links (inbound links) from other highly regarded sites is typically seen as more authoritative.
- Use in Search Algorithms: Algorithms like PageRank and HITS (Hyperlink-Induced Topic Search) are built on the principles of link analysis. These algorithms not only consider the content of the pages but also how pages are interlinked to determine their relevance and ranking in search results

HITS

HITS is a novel approach for analyzing web page value and authority through their network of links. The algorithm operates on the key concepts of “Hubs” and “Authority”

- Authority: A page is considered an authority on a specific topic if it has high-quality content that many other pages link to
- Hubs: A page is a good hub on a topic if it links to many relevant authority pages. Essentially, a good hub acts as a directory pointing to high-quality

resources

Algorithm

- The algorithm begins by identifying a subset of pages relevant to the search query, known as the root set. This root set is then expanded to a larger base set by including pages that link to those in the root set and pages that are linked from these
- Each page in the base set is initially assigned a hub score and an authority score.
- The scores are then iteratively updated following these rules:
  - The authority score of a page is updated to be proportional to the sum of the hub scores of the pages linking to it

$$\text{Authority}(p) = \sum_{i \in \text{ set of pages linking to } p} \text{Hub}(i)$$

  - The hub score of a page is updated to be proportional to the sum of the authority scores of the pages it links to

$$\text{Hub}(p) = \sum_{i \in \text{ set of pages } p \text{ links to}} \text{Authority}(i)$$

- After each iteration, the hub and authority scores are normalized to prevent them from growing out of control. This normalization is crucial for maintaining numerical stability in the algorithm
  - Each score is typically normalized by dividing by the norm of the vector formed by the scores:

$$\text{Authority}(p) = \frac{\text{Authority}(p)}{\sqrt{\sum_j \text{Authority}(j)^2}}$$
$$\text{Hub}(p) = \frac{\text{Hub}(p)}{\sqrt{\sum_j \text{Hub}(j)^2}}$$

Hits is query dependent, meaning that the scores resulting from the link analysis are influenced by the search terms. It is executed at Query time, with the associated hit on performance.

PageRank

The basic idea behind PageRank is that the importance of a web page can be determined by the number and quality of links pointing to it. Each link is viewed as a "vote" in the global context of the web. Not all votes carry the same weight; links from pages that are themselves considered important(with a high PageRank) have more value.

PageRank uses a probabilistic model to represent the behavior of a "random surfer" who browses the internet by clicking on links randomly:

1. Random Model: PageRank envisions a user surfing the internet by randomly clicking on links. At each step, there is a small probability (the damping factor λ, typically set to 0.85) that the user will continue to click on links. There is also a complementary probability $(1 - \lambda)$ that the user will jump to a completely new page on the web
2. Rank Distribution: The rank of a page is evenly distributed among all the pages it links to. If a high PageRank page links to many pages, the value transmitted to each destination page is diminished.
3. Iterative Calculation: the PageRank of each page is calculated iteratively based on the PageRanks of the pages linking to it. The formula is:

   $PR(x) = \frac{1-\lambda}{N} + \lambda(\sum_{y \to x} \frac{PR(y)}{out(y)})$

   Where:

   a. $PR(x)$ is the PageRank of page x
   b. N is the total number of pages on the web
   c. $out(y)$= is the number of outgoing links from page y
   d. λ is the damping factor
4. Convergence: This process is repeated until the PageRanks of all pages converge to a stable distribution

PageRank was revolutionary for several reasons:

- Quality of Search Results: It allowed Google to outperform other search engines by better assessing the authority and relevance of pages.
- Anti-Spam: PageRank is relatively resistant to spamming techniques compared to other methods based solely on textual analysis, as manipulating PageRank requires acquiring links from pages already considered to be high authority.

Elastic Search

Elastic Search is primarily used for full-text indexing and searching capabilities. It is highly scalable, allowing for near real-time search, and supports complex search queries. It is a Documented-oriented search engine, allowing complex data structures that may contain dates, geo locations, text, other objects, array of values

Elastic Search	Field	Document	Index	Cluster
SQL	Column	Row	Table	Database

Elastic Search can easily scale horizontally, providing the ability to distribute data across many nodes and manage indexing and search operations across these nodes seamlessly

Elastic Search uses a RESTful API, which means interactions with the search engine are done through standard HTTP requests. This makes it easy to integrate with a wide range of applications and programming languages

Searching

1. Structured Query on Specific Fields: Structured Queries in Elastic Search are similar to querying specific columns in SQL database. These queries focus on searching data stored within specific fields in a document, utilizing exact matches or range queries among others. The structured nature of these queries allows for precise and efficient searches when you know the exact structure and content of the data you are querying.
    - Field-specific: Targets searches to specific fields such as dates, geo-locations, status fields, numerical data, etc.

- Precise Matching: Often used for filtering where exact matches are important, e.g. `status = “active”` or price between 10 and 20
- Use Cases: Very effective for dashboards searches, reporting tools, or any application where users need to find documents based on specific criteria or ranges.

```
{
  "query": {
    "bool": {
      "must": [
        { "match": { "user": "John Doe" }}
        { "match": { "status": "active" }}
      ]
    }
  }
}
```

2. Full-Text Query: Full-text queries utilize Elastic Search powerful text analysis capabilities to perform searches that go beyond exact matches, allowing for searches on text fields that take into account synonyms, stemming and other linguistic variations. These queries are ideal searching large bodies of text, such as articles, emails or product descriptions.
   - Text Analysis: Automatically handles tokenization, applies normalization to the terms that are not exactly the same, but similar enough to be still relevant:
     - Lowercase vs uppercase
     - Stemming
     - Synonym management
   - Relevance Scoring: Results are not just about matching but also about how relevant they are to the query, using scores calculated by algorithms like TF-IDF or BM25

```
{
  "query": {
    "match": {
      "content": "quick brown fox"
    }
  }
}
```

3. A combination of above: Often, the most powerful searches in Elastic Search involve a combination of structured and full-text queries also called Compound query. This approach allows users to perform complex searches that filter documents based on specific criteria while also searching for textual matches within those filtered.

```
{
  "query": {
    "bool": {
      "must": {
        "match": { "content": "organic cotton" }
      },
      "filter": [
        { "term": { "status": "active" }},
        { "range": { "price": { "gte": 10, "lte": 50 }}}
      ]
    }
  }
}
```

Query vs Filter

- Query: Queries in Elastic Search are all about relevance. They are used not only to find documents but also to score them based on how well they match

the query criteria. This scoring is crucial for ranking documents in order of relevance, which is particularly important in full-text search scenarios

- Filter: Filters in Elastic Search are used for selectively narrowing down search results based on exact matches. They do not score documents; instead the simply include or esclude documents based on whether they meet the criteria specified in the filter

Relevance score

Relevance scoring is a calculation used to determine how well each document matches a query. This score is a numerical value that indicates the relevance of a document to the search terms entered by the user. The higher the score, the more relevant the document is considered to that query.

Relevance score is calculated with TF-IDF or BM25

Vector Space Model

In VSM, both documents and search queries are represented as vectors. Each dimension in the vector corresponds to a separate term. Each vector term is weight of one term, calculated by BM25 scoring.

BM25 can be replaced with bert score to obtain semantic similarity (Elastic Transformer)

Cosine similarity: this is a metric used to measure how similar the documents are to the search query in terms of angle in the vector space. A smaller angle between two vectors indicated a higher degree of similarity. This is crucial in the SVM as it provides a way to rank documents in a search result based on how similar they are to the query vector.

Sharding

In ES, sharding refers to the process of distributing data across different containers known as “shards”, which helps in achieving scalability and high availability

- Horizontal scaling: sharding allows ES to distribute data across multiple nodes in a cluster, facilitating horizontal scaling
- High availability: Each shard can have zero or more replicas. Replica shards provide redundancy, which ensured data availability and protection against hardware failures.
- Improved Performance: By distributing the data across multiple shards, and potentially across multiple nodes, both read and write operations can be parallelized, which anhances performance.

Versioning

Versioning refers to the mechanism that controls the updating and deletion of documents. It helps in maintaining the consistency of data as it changes over time. Each document in an index is associated with a version number, which is incremented each time the documents is updated

Recommender System

Recommender systems are specialized algorithms designed to suggest relevant items to users. These items could be anything from movies, books and articles to more specific recommendations like friends on social network or products on a shopping site.

The goal of these systems is to predict user preferences based on various factors and improve user experience by personalizing content offerings.

Three main approaches:

1. Content Based
2. Collaborative Filtering
3. Hybrid approaches

Utility Matrix

The utility matrix in RecSys is a fundamental structured used to represent the preferences of users over a set of items.Typically this matrix is known as a user-item matrix where the rows represent users and the columns represent items. The

entries in this matrix are trating that users have assigned to items, indicating their preferences.

Formalization

Given

- U: A set of users
- I: A set of items
- R: A utility matrix that includes rating users have given to items
- Known ratings in R

Goal: Predict the unknown values in R for each user-item pair where the rating is not known. This prediction task involves estimating how a user would rate items that they have not yet rated

Output: For each user $u \in U$, The recommender system aims to return a list of tok-K items that the user is likely to rate the highest. Here K is an input parameter that specifies how many recommendations to make for each user

Populating the Utility Matrix

- User Annotation: Users explicitly rate items, providing direct input on their preferences. This can range from complete, where users rate many items, to partial, where users rate only a few.
- Rate Inference (Implicit Feedback): In cases where explicit ratings are not available, the system might infer preferences from user behavior. For example, the duration a user spends watching a video or the frequency of purchases can serve as implicit feedback.
- Machine Learning Models: These models can independently predict item ratings based on patterns detected in the data, using both explicit and implicit inputs.

Content Based

Content based filtering recommends items based on the features of the items and a profile of the user's preferences. This type of system assumes that if you liked a

certain item in the past, you will also like similar items in the future.

Key characteristics:

- Item Profiles: Each item in the dataset must have a set of characteristics (features). For example, in a movie recommendation system, the features could be the genre of the movie, director, main actors, etc.
  Item profile
  V_i: vector representation of an item $i \in I$
- User profiles: The system build a profile for each user based on the features of items they have previously liked, reated, or interacted with.
- Similarity Measures: the system uses these profiles to calculate the similarity between the user and item features. Commonly used similarity metrics include cosine similarity, Euclidean distance and the Jaccard Index

  - Euclidean distance $||\mathbf{A}-\mathbf{B}||^2 = (\mathbf{A}-\mathbf{B})^T\ (\mathbf{A}-\mathbf{B})$
  - Cosine similarity $\frac{\mathbf{A}\cdot\mathbf{B}}{||\mathbf{A}||\ ||\mathbf{B}||} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$

$$Eucl(\mathbf{A},\mathbf{B}) = ||\mathbf{A}-\mathbf{B}||^2 = (\mathbf{A}-\mathbf{B})^T(\mathbf{A}-\mathbf{B}) = 2\ (1-cos(\mathbf{A},\mathbf{B}))$$

Jaccard similarity and Dissimilarity

- Jaccard Similarity: JS is defined for binary vectors, meaning vectorrs where each dimension can either be 0 or 1. This measure is used to compare the similarity between two sets or vectors based on the number of dimension in which both vectors have a value of 1.
  - a: Number of dimensions that equal 1 for both vectors V_i and V_j. This represents the intersection of the two sets

- b: Number of dimensions that equal 1 for vector V_i but equal 0 for vector V_j
- c: Number of dimensions that equal 0 for vector V_i but equal 1 for vector V_j
- d: Number of dimensions that equal 0 for both vectors V_i and V_j. This represents the union part that doesn't overlap.

Formula:

$$JS_{ij} = \frac{a}{a+b+c}$$

- Jaccard Dissimilarity: JD complements the similarity measure by quantifying how different the two vectors are. Is is calculated as one minus the JAccard similarity
  Formula:

$$JD(V_i, V_j) = \frac{b+c}{a+b+c} = 1 - JS_{ij}$$

Collaborative Filtering

Collaborative filtering is a widely used approach in recommender systems that makes automatic predictions about the interests of a user by collecting preferences or taste information from many users. The underlying assumption is that if the users agreed in the past, they will agree in the future about certain items. In essence, collaborative filtering recommends items by identifying patterns of agreement between users of a system

- Memory-based approaches: Exploit the rating in the utility matrix to compute similarities between users/items
- Model-baesd approaches: Exploit the estimate or learn a model and then apply it to forecast the user rating

Item-based neighbor models

Assign to the target item a rate similar to those assigned by other users to the most similar items

- $F(x,i)$: The predicted rating of user x for item i
- S_{ij}: The similarity score between item i and another item j.This similariti is typically calculated using methods like cosine similarity or Pearson correlation on the rating vectors of the items
- $N(x,i)$: The set of items that user x has rated that are also similar to item i. These are the neighbors in the context of item i

$$F(x,i) = \frac{\sum_{j \in N(x,i)} S_{ij} \cdot F(x,j)}{\sum_{j \in N(x,i)} S_{ij}}$$

This formula computes the predicted rating by taking a weighted average of all the rating user x has given to items similar to i, weighted by the similarity between item i and these items. This method emphasizes items that are more similar having a stronger influence on th prediction

User-based neighbor models

Assing to the target item a rate similar to those assigned by the most similar users

- $F(u,i)$: The predicted reting by user u for item i
- S_{ux}: The similarity between user u and another user x, calculated based on the similarity of their rating patterns across all items.
- $N(u,i)$: The set of users most similar to u who have rated item i. These are user u's neighbors

$$F(u,i) = \frac{\sum_{x \in N(u,i)} S_{ux} \cdot F(x,i)}{\sum_{x \in N(u,i)} S_{ux}}$$

Here, the predicted rating is calculated as the weighted average of the rating item i has received from users similar to u, with weights proportional to their similarity to u.This approach assumes that similar users have similar tastes.

Hybrid system

Combine content-baes with collaborative filtering approaches

- Ensemble methods: Build separate methods and then combine the predictions
- Nested methods: Integrate either content-based characteristics into a collborative approach or collaborative characteristics into a content-based approach
- Mixed strategy: Build a general model that integrates both content-based and collaborative characteristics.

Problems

Sparsity

In many real-world scenarios, the user-item interaction matrix that is used by recommender systems is extremely sparse. This means that most users interact with onlya small fraction of items, and most items are rated by only a small fraction of users.

First Rater

This problems occurs when an item has just been introduced and no one has rated it yet. Since collaborative filtering algorithms rely on existing ratings to make recommendations, they cannot recommend new items

Cold start

The cold start problem refers to the difficulty that recommender systems face when a new user or a new item enters the system. For new users, there is no historical data to analyze to predict their preferences. For new items, there are no previous ratings to help recommend them

Filter bubble

A filter buble is a stato of intellectual isolation that can occur when websited make use of algorithms to selectively assume the information a user would want to see, and then give information to the user according to this assumption

Popularity Bias

Recommender systems often have a tendency to recommend items that are already popular or have been rated by a large number of users. This can lead to a reinforcement loop where popular items become more popular an niche items remain obscure

Long-tail Phenomenon

This phenomenon emphasizes promoting a wide array of niche or less popular products alongside bestsellers. This approach diversifies the recommendations presented to users, enhancing personalization and user satisfaction bu exposing them to a broader range of items that might match their unique interests. *Non sono sicuro sia corretto*

Model-based

Dimensionality Reduction

Dimensionality reduction techniques are used to reduce the utility matrix to capture the most important aspects of the data, ignore the rest

- Singular Value Decompositions (SVD): This technique reduces the dimensionality of the original user-item matrix by keeping only the most significant features that explain most of the variance in the data

Probabilistic Models

Probabilistic models estimate the likelihood of a particular set of data by assuming an underlying probability distribution. These modes can use Bayesian statistics to

predict the probability of a rating given the relationships between users and items modeled as a probabilistic graph

Machine Learning Models

- Graph-based Hybrid Method:
  - Integrates content-based and collaborative signals into a graph-based model where nodes represent users and items, and edges represent interactions or similarities.
  - Techniques like Graph Neural Networks can be used to propagare information through this network, effectively capturing complex interdependencies between users and items
- Clustering-Based Hybrid Methos:
  - Groups similar users or items into clusters based on their characteristics or behaviors.
  - Predictions for user are made by averaging the preferences of other users in the same cluster or by using typical ratings of the cluster for specific items
- Association-Based Hybrid Method:
  - Utilizes association rule mining to find rules that predict the occurrence of an item based on the occurrences of other items in a user's history
  - Rules such as "if a user likes X and Y, they will likely like Z" can be used to recommend items.

Transformer Models

- RecoBERT:
  - A BERT-based model adapted for recommendation system that leverages the bidirectional nature of BERT to better understanf the context of user-item interactions
  - It can synamically incorporate the sequence of user interactions to predict the next likely interests
- Item-based Collaborative Filtering with BERT:

    - Adapts BERT to encode items based on their attrivutes and uses the encoded representation to calculate similarities between items, which are then used for making recommendations
- BERT4Rec:
    - This model specifically tailors BERT for sequential recommendation tasks
    - It treats user interactions as a sequence and uses the BERT architecture to predict the next item in the sequence, effectively capturing complex item transition dynamics

Evaluation

Results assessment

- $F_p(u, i)$: predicted rating for user u and item i
- $F_a(u, i)$: actual rating for user u and item i
- Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\sum_{u \in U, i \in I} (\mathcal{F}_p(u,i) - \mathcal{F}_a(u,i))^2}$$

Results assessment

- Precision at top K (P@K)
  - percentage of correct items in top K recommended items for a given user
- Rank correlation
  - Spearman's correlation between the vector of predicted ratings and those of actual ratings

Results assessment: binary prediction

- Boolean rating: either relevant or not relevant
- Mean of Reciprocal Error $MRR = max_{i \in I^r} \frac{1}{rank(i)}$
  - Ir is the set of relevant items
  - rank(i) is the item rank in the recommended list
- Coverage
  - Number of items/users for which the recommender can make predictions
- Precision
  - Percentage of recommended relevant items
- Receiver operating characteristic (ROC)
  - Trade-off curve between false positives and false negatives

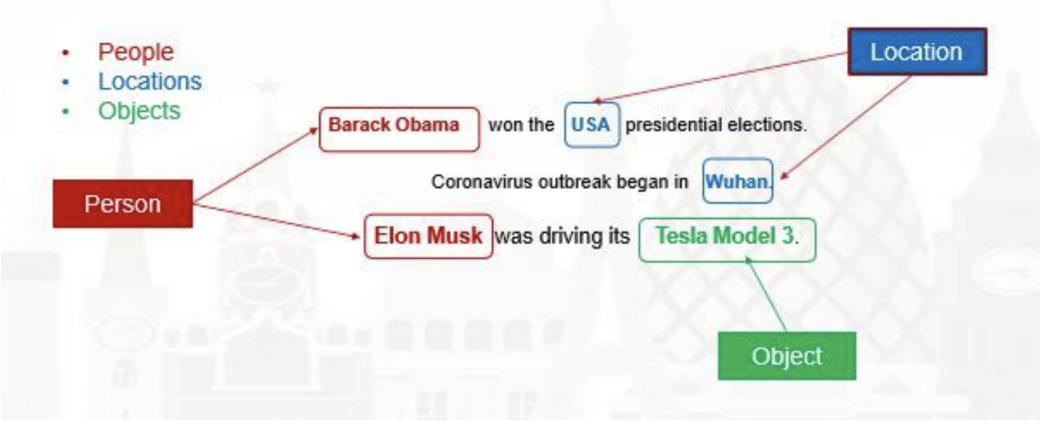
Results assessment: further aspects

- Diversity
  - diversify recommendations with the purposes of
    - Increase user satisfaction
    - Mitigate the potential effect of overfitting
- Domain adaptation
  - Tailor the recommender systems to the actual context
- Prediction order
  - refine the output rank according to domain-specific constraints
- Biased annotations
  - Ratings either do not reflect the actual service needs or miss a global viewpoint

Named Entity recognition

Named Entity Recognition is a crucial task in NLP involving the identification and classification of named entities in the text into predefined categories such as names of people, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

This process is fundamental for many applications like information retrieval, question answerting systemsm content analysis, and knowledge extraction



Named Entity Disambiguation entails connecting an entity mention to a concept in a given ontology

Semantic models

Semantic models in NLP are designed to understand the meanings of words and phrases in context, beyonf just their syntactic toles. These models captire relationships and contexts that words appear in, helping machines comprehend natural language similar to human understanding. This is particularlt useful in NER, as it helps in distinguishing between entities of the same name based on context

Semantic Web

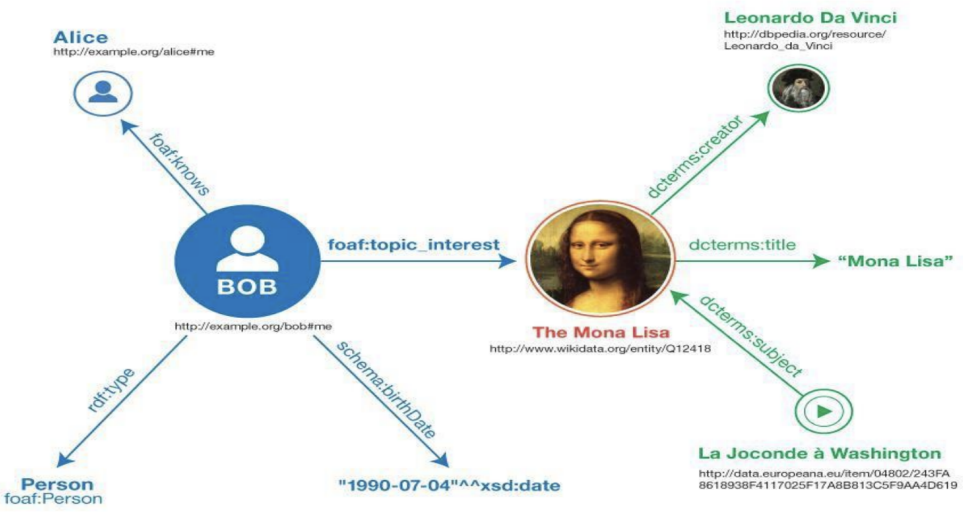
The semantic Web is an extension of the current web, where information is given well-defined meaning, anebling computers and people to work in cooperation. It uses standards, tools and languages that provide a framework allowing data to be shared and reused across application, enterprisen and community boundaries. This is closely related to NER as it involves understanding the semantics of data on the web, which often includes entities.

Resource Description Framework

RDF is a standard model for data interchange on the web. RDF exteds the linking structure of the web to use URIs to name the relationship between things as well as the two ends of the link. In the context of NER, RDF can be used to describe and link entities to a vast array of semantic web resources, enhancing the entity's semantic interpretation and usability in web contexts.

Ontologies

In NLP and semantic web, an ontology represents a formal naming and definition of the types, properties and interrelationships of the entities that really or fundamentally exist for a particular domain. In terms of NER, ontologies help in understanding the domain-specific context of entities, thereby improving the accuracy of entity classification and relationship mapping



Taxonomy

Taxonomy in NLP is the hierarchical classification of entities into categories and subcategories based on their characteristics. It helps in organizing knowledge about entities and their relationships in a structured format, which can be very useful un enhancing NER system by providing a framework to categorize and relate entities in a meaningful way.

Thesauri

A thesaurus in NLP is a resource that lists words grouped together according to similarity of meaning. In the context of NER, thesauri help in recognizing different expressions or forms of the same entity, aiding in the robustness and flexibility of the recognition process.

Rule-based NER

Rule-based NER is one of the foundamental approaches in the field if NLP for identifying and classifying entities in text. This method relies on predefined rules and patterns rather than statistical models or machine learning algorithms. It's particularly useful when you have clear, structured guidelines for what constitutes an entity within specific domains or when dealing with languages or domains for

which there isn't a large amount of annotated data available for training machine learning models

Rule-based NER systems use sets of rules that can be crafted manually by experst or generated using automated techniques. These rules are designed to identify specific patterns in text that indicate the presence of an entity. Common components of these rules include:

- A sequence of words starting with a capital letter, followed by lowercase letters
  - E.g. Barack Obama
- May contain a prefix title (Dr., Mr., Prof.)
  - E.g. Prof. Enrico Fermi
- May contain an initial in the middle
  - E.g. George W. Bush
- May contain a designation indicator prefix
  - E.g. Elon Musk CEO
- Never include special characters: %/$/=...

Regular Expressions

RegEx is a powerful tool in rule-based NER systems for pattern matching and Entity extraction. RegEx is commonly used to extract entities like dates, phone numbers and proper names based on their predictable formats. It's also useful for creating coundaries around what extacly constitutes an entity in a given domain

Challenges

- The same entity can have multiple variants of the same name
  - E.g. Luca Cagliero, Prof. Cagliero, Professor L. Cagliero
- Proper names can be ambiguous
  - E.g. Felice Buonanno, Santa Pazienza, Guido Di Rado

Enhanced Rule-based NER

- Combine regular expressions with basic NLP preprocessing steps
  - Part-of-speech (POS) tagging
    - Mark each word as a noun, verb, preposition, etc.
  - Syntactic parsing
    - Identify phrases: NP, VP, PP
  - Text-to-audio
    - Identify phonemes
  - Assign semantic word categories
    - E.g., from the WordNet lexical database (https://wordnet.princeton.edu/)
      - KILL: kill, murder, assassinate, strangle, suffocate

Ontology-based NER

- YAGO: is a large semantic knowledge base. It contains millions of entities and organizes these entities in a semantic network that includes various types of relationships between them. YAGO is especially useful for NER because it combines the broad coverage of Wikipedia with the semantic depth of WordNet, providing detailed information about entities including their types, relationships, and attributes. This makes it an excellent resource for enhancing entity recognition and linking entities to a structured semantic database.
- NERD (Named Entity Recognition and Disambiguation) is an approach that integrates multiple NER tools and combines their outputs using semantic reconciliation techniques. It is designed to improve the extraction and disambiguation of named entities from text by leveraging both statistical and semantic web technologies. NERD frameworks often utilize existing ontologies and knowledge bases (like DBpedia, YAGO) to contextualize and link identified entities to their semantic web URIs, facilitating a more robust and interconnected NER process.
- DBpedia is a crowd-sourced community effort to extract structured content from the information created as part of the Wikipedia project. This resource provides a valuable ontology and has been widely used in semantic web applications, including NER. DBpedia extracts information such as infobox

data from Wikipedia, turning it into a structured form that can be queried like a database. For NER, DBpedia is particularly useful for entity linking, where extracted entities from text are linked to their corresponding DBpedia URIs, enriching the text with detailed semantic information from the database.

- The Natural Language Toolkit (NLTK) is a Python library that provides easy-to-use interfaces to over 50 corpora and lexical resources, including WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. While NLTK itself is not an ontology-based tool, it supports the development of NER systems by providing fundamental NLP tools necessary for preprocessing text, and it can be integrated with ontology-based systems to enhance their performance by preprocessing input data effectively.
- Wikidata is a free and open knowledge base that acts as a central storage for the structured data of its Wikimedia sister projects including Wikipedia, Wikivoyage, Wikisource, and others. Similar to YAGO and DBpedia, it provides a vast amount of data about millions of entities and their relationships. Wikidata's ontology includes properties that describe relationships between items, which can be incredibly beneficial for ontology-based NER to not only recognize entities but understand the possible semantic relationships and properties associated with them.

Machine learning models

Standard Classification Models

Standards classification in NER are primaly used to determine whether a token within a sentence belongs to a named entity class or not. These models often treat the problem as a binary or multi-class classification task, where each token is independently classified into one of the predefined categories

Each token in the text is represented by a set of features, which could include lexical features, syntactic features and semantic features

The classifier is trained on these features to predict the class of each token. This approach doesn't inherently consider the context of tokens; each token is classifies independently

Sequence Labeling Models

Sequence labeling models address the contextual limitatuon of standard classifiers by treating NER as a sequence2sequence task. Here, the entire sentence or document is processed at once, and a label is assigned to each token in the context of its neighboring tokens

These models consider both the individual characteristics of tokens and their relationshiips to adjacent tokens in the sequence. This approach allows the model to capture information about entity boundaries and the interdependencies between adjacent labels

Transformer for NER

Unlike prior models that generate static embedding for words, transformer provide contextualized embedding. This means that the representation for a word will vary based on the surrounding words, allowing for more nuanced understanding and identification of entities

LUKE architecture

LUKE is a transformer model that extends the robust capabilities of transformers by incorporating entity-aware self-attention mechanisms. It was specifically designed to improve performance on NER and other entity-related tasks

Entity-aware Self-Attention: LUKE modifies the standard self-attention mechanism to differentiate between words and entities. This allows the model to learn and apply different attention patterns depending on whether it is focusing on a regular word or an entity, enriching the model's understanding of how entities interact within the text.

Pre-training with Entities: LUKE is pre-trained not only on text but also on a specially created corpus where entities are explicitly marked and linked to a knowledge base. This pre-training helps LUKE learn rich representations not only for the words but also for the entities themselves.

By focusing on entities during both pre-training and fine-tuning, LUKE can achieve more precise entity recognition and better disambiguation of entities with similar names but different contexts or meanings.

Intent detection

Text classification

Preprocessing

Text processing is crucial because it transforms raw data into a format that machine learning algorithms can understand and use efficiently.

Text is preprocessed for the subsequent steps

- Sentence splitting
- Word splitting
- Lemmatization
- Stopwords removal
- ..

We can than do feature extraction to extract relevant features that can be used for the classification

- Part-Of-Speech tagging
- Dictionary-based word count
- Relies on the Bag-Of-Word text representation
- Semantic tree parsing

Machine learning approaches

Machine learning approach involves the extraction of features from the text and labels with which to train the model. Text classification requires a supervised approach, so it is important that the labels are there and that they are correctly assigned.

Once there are labels and features, it is possible to train a DL model for text classification. Sometimes, due to a shortage of labels, it is necessary to follow a classical machine learning approach such as:

- Naive Bayes
  - Applies bayes theorem with naive independence assumptions between the features
- Logistic Regression
  - Use a logistic function to model the probability of a certain class
- Support Vector Machines
  - Maps training examples to points in space so as to maximize the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to category based on which side of the gap they fall
- K-nearest Neighbors
  - Classifies a data point by plurality vore of its neighbors, with the object being assigne to the class most common among its k nearest neighbors

Transformers can be also finetuned to classify a text, although transformers are not classifier models but sequence to sequence models, a classifier layer like softmax can be added at the end to perform text classification

Challenges

Data imbalance

Data imbalance occurs when some classes have significantly more samples than others. This is a common issue in intent detection, where some intents might be more frequent due to common user interactions, while others are rare

Solutions:

- Oversampling Minority Class: Increasing the number of samples in the minority class by replicating them. this can make the training data more balanced, but risks overfitting as the model might learn the exact patterns of the repeated samples
- Undersampling Majority Class: Reducing the number of samples in the majority class to balance the dataset. This might help in reducing bias, but at the cost of losing potentially valuable data

Multiclass Classification

Often a text can be have multiple classes. the challenge increases as the number of classes grows, requiring the model to distinguish between finer nuances of the user queries

Solution:

- One-vs-All: This involves training a single classifier per class, with the samples of that class as positive and all other samples as negatives. This method simplifies the problem into multiple binary classification problems.

Fuzzy text classification

Text data inherently noisy and ambiguous. Phrases might be vague or have multiple meanings based on context and different words or phrases might express similar intents

Solution:

- Fuzzy logic: Employing fuzzy logic can help in handling the uncertainty and vagueness in text. It allows for degrees of truth rather than a strict true/false evaluation, which is especially useful in text classification

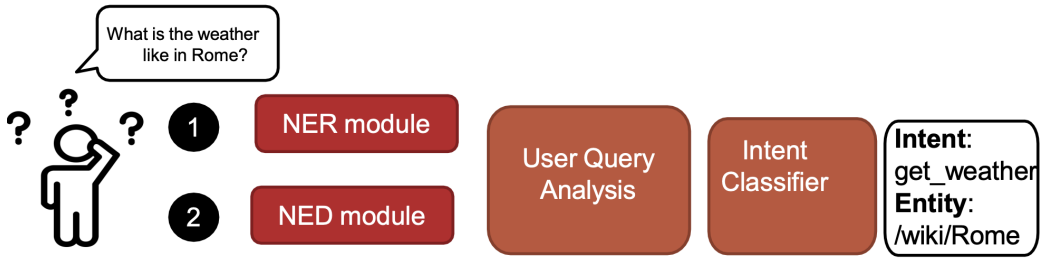
Intent detection

Intent Detection is a crucial aspect of NLP that focuses on determining the purpose or goal behind a user's input in conversational system. This could be in a text chat, a voice command, or any form of natural language communication interface. The ultimate aim is to correctly interpret what the user intends to accomplish and then trigger the appropriate action or response

Main components

1. Named Entity Recognition: This step involves identifying and classifying key information in the text into predefined categories such as names, locations, organizations, etc. NER helps in extracting structured information from unstructured text, making it easier to understand the context or specific details that are important for fulfilling the user's intent

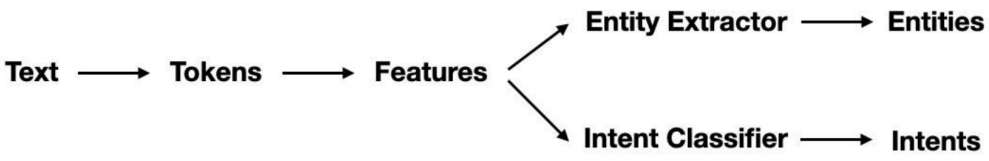
2. Named Entity Disambiguation: Once entities are recognized, NED resolves ambiguities about these entities. This involves linking them to unique identifiers in a knowledge base or database.
3. User Query Analysis: This step involves parsing and understanding the user's query in-depth. It's about comprehending the grammatical structure, extracting key phrases and understanding the overall sentiment or urgency of the request. This analysis helps in better understanding how to classify the intent accurately
4. Intent Classification: The final step is to determine the intent of the user query. This involves classifying the query into one of several predefined categories such as booking a ticket, asking for weather updates, making a reservation, etc.



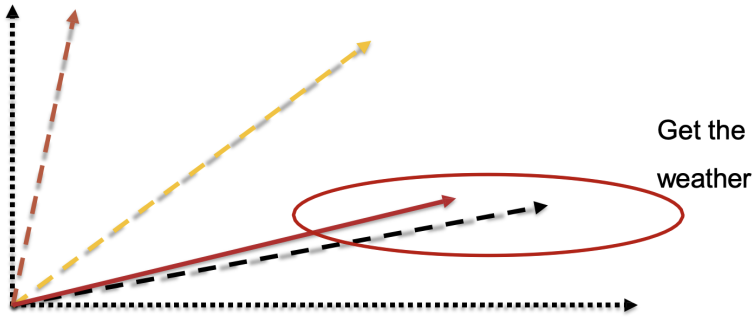
Regex can be useful for quick filtering simple and common request as "Start music", "Add item to shop list" and so on.

Machine learning approaches

As NER, text classification and other classification tasks, intent classification follows the same pipelines



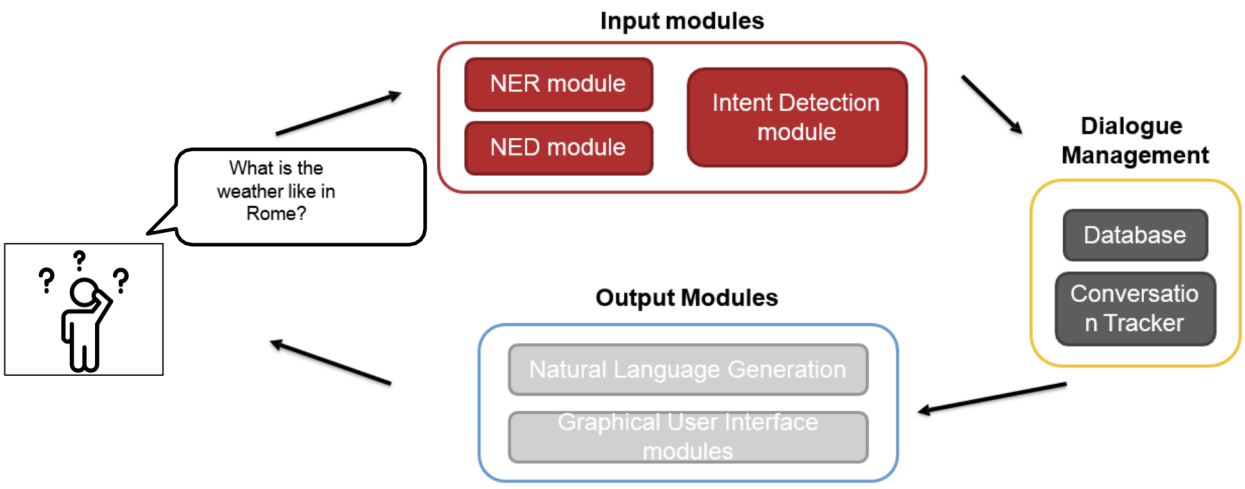
- **Intent classification**
  - Assign the intent whose latent representation is most similar to the user query



- Importance of an Unrecognized Class: Including an “unrecognized” intent class in your system helps handle ambiguous or novel user inputs that don’t match predefined categories. This prevents incorrect responses and improves the user experience by enabling the system to ask for clarification or direct the query to a human operator
- Monitoring Unrecognized Intents: Regularly analyzing inputs classifies as unrecognized can reveal gaps in the intent detection model. Using unsupervised learning techniques like clustering, these inputs can be grouped to identify potential new intents based on common themes or patterns that emerge from data
- Role of Semi-supervised Learning: This approach combines a small amount of labeled data with a larger set of unlabeled data to train or refine the model. It’s particularly useful for incorporating new intents into the system by allowing the model to adjust its decision boundaries based on both the newly labeled examples and the existing unlabeled data.
- Dynamic System Adaption: By continuously monitoring, analyzing and integrating feedback from unrecognized queries, the system can dynamically adapt to new trends, user behaviors and language use. This ongoing learning process ensures that the intent detection system remains effective and relevant over time.

Ai ChatBot

Pipelines



- Input modules: the input modules are the first point of contact in a chatbot's architecture, responsible for interpreting and processing the user's input into a dorm that can be handled by the system.
    - Named Entity Recognition: This component analyzes the user's input to identify and extract entities—specific pieces of data that can be categorized into predefined groups like names, locations, dates, etc.
    - Named Entity Disambiguation: This module resolves ambiguities related to entities extracted by NER module.
    - Intent Detection Module: After entities are recognized and disambiguated, this module determines the user's intent—what the user is trying to achieve with their query.
- Dialogue Management: Once the user's intent is understood and necessary entities are captured, the dialogue management system takes ove. This module manages the flow of the conversation based on the user's intent and other contextual information
    - Database: A backend database is often integrated to retrieve information needed to fulfill the user's request
    - Conversation Tracker: This component keeps track of the conversation context and history. This is crucial for maintaining the continuity of the

interaction, especially in complex dialogues that require multiple exchanges to complete a task or resolve a query

- Output Modules: After processing the input through the input modules and dialogue management, the chatbot needs to generate a response that is conveyed back to the user
  - Natural Language Generation: This module is responsible for converting the structured information retrieved from the database or computed by the application into human-readable text.
  - Graphical User Interface Modules: In addiction to text responses, modern chatbots often include a graphical interface that can display information more dynamically. For example, showing a weather forecast chart or proving interactive options for additional details

Frameworks

- DialogFlow: DialogFlow is a comprehensive development suite provided by google that allows developers to build conversational interfaces for websites, mobile applications and popular messaging platforms like Facebook Messenger, Slack and more. It is widely recognized for its powerful natural language understanding capabilities
- IBM Watson: IBM Watson Assistant is part of IBM0s Watson suite of AI services. It provides tools to create conversational agents that can engage with users via text or voice, across various channels
- Amazon ChatBot Development framework: Amazon ChatBot Development is a service for building conversational interfaces into any application using voice and text. It is the same technology that powers Amazon Alexa. This framework provides advanced deep learning Functionalities of automatic speech recognition and natural language understanding to enable developers to build engaging interfaces

RASA

Rasa is an open-source framework for building conversational AI applications, including chatbots. It’s known for its flexibility and control, allowing developers to

build sophisticated, context-aware conversational agents

RASA NLU

Rasa Natural language understanding breaks down user messages into structured data that can be understood by machine. Here's how it works in more detail:

- Intent Classification: This component classifies the user's message into an intent. An intent represents the purpose of the user's message
- Entity Extraction: Simultaneously, Rasa NLI also performs entity extraction, identifying and classifying key pieces of information from the user message.
- Pipeline Configuration: The processing pipeline for Rasa NLU can be configured with various components such as tokenizers, featurizers and entity extractors.

RASA Core

Rasa Core handles the decision-making part of the chatbot by predicting the next best action based on the current conversation state. It employs more complex mechanisms:

- Dialogue Management: It uses a machine learning model to predict the next action based on the dialogue history and the current state. This history is maintained in the conversation tracker.
- Forms: For collecting a series of information from the user, Rasa Core can use forms that prompt the user for specific details one piece at a time

Stories

Stories are essentially the training data for Rasa Core. they are structured examples of conversations that teach the model the sequence of actions to take in various scenarios.

Checkpoints

Checkpoints help to simplify and reuse stories. They are points in a story that can be referenced in other stories

Rasa Tracker

Rasa Tracker is essentially the memory of your chatbot. It keeps track of everything happening in a conversation

Rasa uses slots, which are like variables in programming, to store information gathered during the conversation. the tracker updates there slots as the conversation progresses, affecting the flow and decision of the dialogue.

Rasa Policies

Rasa Policies are algorithms that decide the next action. there are different types of policies for different needs:

- TED Policy: A Transformer Embedding Dialogue Policy that uses transformer models to decide the next action based on the dialogue context
- Rule Policy: For defining conversational rules that always apply, like starting a conversation with a greeting

Rasa Natural Language Generation

Rasa NLG is responsible for crafting the responses sent back to the user. It's closely tied to the chosen actions:

- Response Templates: Developers can define templates for responses in the domain file, which Rasa uses to generate messages.
- Custom NLG: More advanced setups can use a custom NLG server to generate responses dynamically based on the context, using more sophisticated language models.

Text Summarization

Text summarization in NLP is a task for automatically reducing a long text into a shorter version that captures the essential information and main points of the original content.

A good summary should be concise, coherence, cover all the relevant part of the text and neutral through the argument

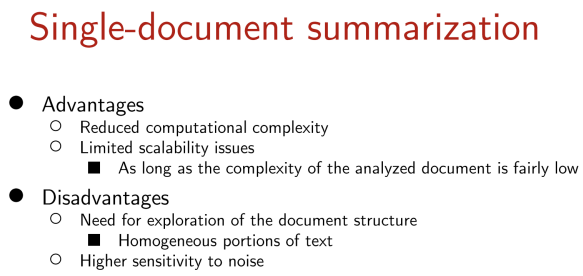
Formalization

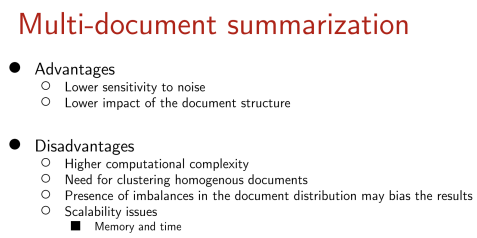
- Input - Original text: A document or set of documents which can range from a few sentences to multiple pages. The text may contain structured data or unstructured data
- Output - Summary: A condensed version of the original text that retains the essential information. The length of the summary can be predefined or variable, depending on the summarization task
- Objectives: To extract or generate key information from the text in a way that reduces the original length significantly while preserving meaning, coherence and context
- Type of summarization:
  - Extractive summarization: Selecting key phrases or sentences directly from the original text and concatenating them to form a summary
  - Abstractive summarization: Generating new phrases or sentences that capture the essence of the text, not necessarily using the exact wording from the original document

Issues/Challenges

- Collection Size: Larger text collections present scalability issues. Summarizing long documents or large sets of documents demands significant computational resources and effective algorithms to manage memory and processing efficiently
- Document Characteristics and Structure: Documents vary widely in structure and style, such as scientific papers with distinct sections versus narrative texts like novels. This variability affects the ability to identify and extract key points effectively
- Language: Different languages present unique challenges due to varying syntax, semantics, and complexity. Some languages may lack sufficient NLP tool support, especially low-resource languages
- Collection Updates: Document collections that receive frequent updates, such as news feeds or ongoing research topics, require summaries that can dynamically incorporate new information without losing relevance or coherence

- Summary Goal: Different applications require different types of summaries. For example, an executive summary for a business report focuses on bottom-line outcomes and data, whereas a summary for a literary analysis might focus on themes and character development





Multi-lingual vs Cross-lingual

Multi-lingual

Multi-lingual summarization involves summarizing documents that could potentially be written in different languages, but the output summary is produced in a single language. Here's what it entails:

- Problem: Summarizing content from multiple documents that may be written in different languages, but each collection of documents summarized is uniform in language.
- Challenges:
  - Need for solutions portable to different languages: The summarization solution needs to be effective across multiple languages, handling language-specific nuances without the need for extensive reconfiguration.
  - No need for combining models related to different languages: Unlike cross-lingual summarization, each summary is language-specific, meaning there is no immediate requirement to merge or translate outputs across languages.

Cross-lingual

Cross-lingual summarization, on the other hand, focuses on creating summaries from multi-lingual sources and translating them into a different target language.

- Problem: Summarizing collections of documents where the individual documents within the collection may be in different languages, and the summary itself may also be in a different language from the sources.
- Challenges:
  - Need for solutions portable to different languages: Similar to multi-lingual summarization, it requires effective processing across various languages.
  - Need for combining models related to different languages: This involves integrating language translation capabilities into the summarization process to ensure that the final summary is cohesive and comprehensible in the target language.

Temporal Summarization

Temporal summarization is an advanced method of text summarization that specifically deals with time-sensitive documents, such as news articles or social media streams. The goal is to summarize a stream of textual content that is continuously updated over time. This form of summarization is particularly useful for users who need to stay informed about evolving topics without having to read every update

Process Overview

1. Preprocessing: This initial step involves cleaning the data, which may include removing irrelevant content, standardizing date formats, identifying key entities, and other NLP tasks such as tokenization and lemmatization. This step prepares the data for more efficient analysis.
2. Retrieving Relevant Sentences: The system filters the preprocessed stream to find sentences that match a specific query or set of keywords. This filtering is crucial to ensure that the summary only includes content that is relevant to the user's needs. The relevance may be determined by various factors including keyword frequency, semantic analysis, or user-specific preferences.

3. Retrieve Novel and Relevant Sentences: This stage focuses on extracting sentences that are not only relevant but also provide new information. This helps in minimizing redundancy. It's essential for a temporal summarization system to distinguish between merely repeating known facts and introducing new, relevant data.

Key Features and Challenges

- Stream of Content: Temporal summarization systems handle continuous input streams, such as live news feeds or real-time social media updates.
- Focus on Relevant Parts: The system must effectively identify and emphasize the parts of the stream that add the most value to the summary, disregarding less important updates.
- Latency: In the context of temporal summarization, latency refers to the delay between the occurrence of an event and its representation in the summary. Minimizing latency is crucial for applications where timely information is critical, such as financial markets or emergency response.
- Dynamic Updates: The summarization system needs to continuously update the summary as new information becomes available. This dynamic aspect means that the system must be capable of quickly integrating new data into the existing summary without having to redo the summarization from scratch

Evaluation Methods

Evaluating text summarization, particularly when determining the quality and effectiveness of summaries, is a critical aspect of Natural Language Processing (NLP). There are two primary types of evaluation methods used: intrinsic and extrinsic evaluations. Additionally, the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metric is extensively used for this purpose

Intrinsic Evaluation

Intrinsic evaluation methods focus directly on the quality of the summary itself, without considering its effect on any external task. This type of evaluation assesses characteristics like:

- Coherence and Clarity: How logical and readable the summary is.
- Informativeness: How well the summary captures the essential content of the original text.
- Consistency: The degree to which the summary accurately reflects the main ideas and facts of the original text without distorting them

Extrinsic Evaluation

Extrinsic evaluation methods assess the usefulness of a summary in the context of a specific task or application, such as information retrieval, user satisfaction in a practical scenario, or how well a summarization aids in the performance of another NLP task. This type of evaluation is more about the impact of the summary on subsequent actions or decisions.

The ROUGE Score

The ROUGE score is a set of metrics for evaluating automatic summarization of texts as well as machine translation. It works by comparing an automatically produced summary or translation against a set of reference summaries (typically human-produced). ROUGE is essentially a measure of recall: how much of the content in the human-produced reference summaries appears in the generated summary

- ROUGE-N: Measures the overlap of n-grams between the system-generated summary and the reference summaries. It is a recall-based measure, indicating the proportion of n-grams in the reference that are also found in the generated summary. Common values for N are 1 (unigrams) and 2 (bigrams).

$$\text{ROUGE-N} = \frac{\sum_{S \in \{\text{ReferenceSummaries}\}} \sum_{\text{gram}_n \in S} \text{Count}_{\text{match}}(\text{gram}_n)}{\sum_{S \in \{\text{ReferenceSummaries}\}} \sum_{\text{gram}_n \in S} \text{Count}(\text{gram}_n)}$$

- ROUGE-L: Uses the longest common subsequence (LCS) between the system and reference summaries. It considers sentence-level structure similarity naturally and identifies the longest co-occurring in-sequence n-grams of words. It is particularly good at evaluating fluency and sentence-level structure.
- ROUGE-W: An extension of ROUGE-L that incorporates a weighting scheme, giving more weight to longer sequences of matches. This is used to penalize

fragmented evaluations caused by shorter chunks of matching text spread across the summaries.

- ROUGE-S: Measures skip-bigram co-occurrence statistics, which count every pair of n-grams that are in order and allow for arbitrary gaps. This is designed to capture more distant phrase relationships that are less strict than the adjacency requirements of n-grams.
- ROUGE-SU: A variant of ROUGE-S that also considers unigram co-occurrence, providing a slightly more comprehensive picture of content overlap.

JRouge

JRouge is a Java implementation of the ROUGE evaluation toolkit, typically used for summarization evaluation. It is utilized by developers who prefer Java or are working in environments where Java is the primary programming language.

Multimodal summarization

aspect-based

incremental summarization

BERT score

Mean Reciprocal rank (idk)

METHODS

Clustering based

Clustering-based summarization methods involve grouping similar pieces of text together into clusters and then selecting representative text from each cluster to form the summary.

The idea is to cover the main themes or topics of the entire document set efficiently by identifying and condensing these groups

How it Works

- Text Representation: Each sentence or paragraph in the document is represented as a vector of features, which could include term frequencies, TF-IDF scores, or embeddings.
- Clustering algorithm: These vectors are then fed into a clustering algorithm. The algorithm groups texts onto clusters based on similarity metrics such as Euclidean distance or cosine similarity
- Selection Strategy: From each cluster, a representative sentence or paragraph is selected based on centrality, frequency or the ability to best represent the cluster

Pro vs Cons

- Pro
  - Language-agnostic: Clustering algorithms generally work on numerical representations of text (like vectors), so they don't inherently require knowledge of the language's syntax or grammar. This makes them versatile across different languages
  - Incremental: Hierarchical clustering methods can handle data incrementally, allowing for summarization models to be updated dynamically as new documents are added without needing to reprocess the entire dataset
  - Robust to Noise: Density-based clustering methods, such as DBSCAN, can effectively handle noise and outliers by focusing on dense regions of data space, thus ignoring anomalous data points that could skew the results
- Cons
  - Limited Effectiveness on Complex Document Collections: Clustering methods might struggle with very diverse or complex document sets where the distinction between different topics in not clear-cut. They might also fail to capture nuanced relationships between different pieces of text.

- Dependence on Feature Extraction: The effectiveness of clustering largely depends on the initial representation of the text. poor choices in features extraction and vectorization can lead to suboptimal clustering, affecting the quality of the summer

Graph based

Graph-based summarization methods utilize the concept of representing the text as a graph to determine the importance of each sentence or unit of text, helping to produce summaries that capture the main points of the text. These methods are effective because they can leverage relationships between sentences to assess significant and relevance

In graph-based approaches, text units are treated as nodes in a graph. The edges between these nodes represent relationships based on similarity or other linguistic connections. The importance of each node in the graph is determined through various ranking algorithms, which help identify the most significant sentences to include in the summary

Graph-Based methods are particularly effective because they can leverage the entire structure of the document to determine the significance of each

TextRank

- Idea: TextRank uses a weighted graph where each node is a sentence, and edges represent the similarity between sentences. Sentences are ranked based on their connection, with the rationale being that a sentence connected to many highly ranked sentences is also important
- Pro: Simple and unsupervised; does not require training data. Captures contextual relevance through mutual reinforcement
- Cons: May omit important sentences that are less well connected but still valuable. Relatively sensitive to the way sentences are connected and weighted

LexRank

- Idea: Similar to TextRank, but typically uses cosine similarity of TF-IDF vectors to establish connections between sentences. It's an approach that considers

both the degree of connectivity and the centrality of sentences in the graph

- Pro: Effective at identifying broadly relevant sentences due to its focus on connectivity
- Cons: Like TextRank, it can miss important but less connected sentences. May also produce summaries that are less diverse if many sentences are closely related to each other

GraphSum

- Idea: This methods extends basic graph-based techniques by incorporating additional linguistic features or structured data into the graph, such as named entities, thematic roles and more. It often involves more complex algorithms for ranking nodes in the graph.
- Pro: Can produce more nuanced and contextually rich summaries by integrating deeper linguistic or semantic analysis
- Cons: More computationally intensive and can require more sophisticated setup and tuning

Pro vs Cons

- Pro
  - High Effectiveness: Graph-based methods, such as TextRank or LexRank, often produce high-quality summaries because they consider the global interconnectivity of sentences or text units within a document. This helps in capturing the context and relevance of each unit more accurately.
  - Easy-to-Use: Many graph-based methods are relatively straightforward to implement and do not require labeled data, making them accessible for quick deployment in various summarization tasks.
- Cons
  - High Computational Complexity: Constructing and analyzing graphs, especially for large texts, can be computationally expensive. The process involves calculating pairwise similarities for potentially every pair of sentences in a document, leading to high computational overhead.

    - Not Incremental: Graph-based methods typically require a complete view of the document to construct the graph and perform calculations. They are not inherently designed for incremental updates, meaning any addition to the document set may require a complete reprocessing.

Optimization based

Optimization based approaches use mathematical optimization techniques to select the best set of sentences that satisfy certain criteria, aiming to maximize an objective function. These methods provide a structured way to tackle summarization by framing it as an optimization problem

Submodular Optimization

Submodular functions are a class of mathematical functions that exhibit the property of diminishing returns, meaning adding an element to a smaller set provides a greater increment in value than adding it to a larger set. In text summarization, submodular functions can model various aspects of a summary, such ad coverage, diversity and information.

- A submodular function is defined over sets of sentences, measuring the “value” of any given set
- The goal is to select a subset of sentences that maximizes this function while often subject to constraints
- Example functions might measure the coverage of important terms, diversity between sentences, or overall relevance to the document’s main themes

- Pro
    - Natural Handling of Redundancy: Due to diminishing returns, these functions inherently avoid selecting redundant information.
    - Efficient Approximations: While finding the absolute optimal subset can be computationally hard, greedy algorithms provide good approximations quickly.
- Cons

    - Complexity in Design: Designing the right submodular function that captures all facets of a good summary can be complex.
    - Scalability Issues: Although efficient algorithms exist, the computational load can still be significant for very large text corpora.

COSUM

- Idea: COSUM frameworks incorporate constraints into the summarization process, allowing for the enforcement of additional requirements such as summary length, style or inclusion of key phrases
    - Define an objective function that measures the quality of any potential summary
    - Add constraints that solution must satisfy
    - Use optimization techniques that maximizes the objective function while adhering to the constraints
- Pro
    - Flexibility in Specifications: Allows specific requirements to be programmed into the summarization process.
    - Highly Customizable: Can tailor the summary to meet diverse needs by adjusting constraints and objectives.
- Cons
    - Computational Demand: Solving constrained optimization problems can be resource-intensive, particularly as the number of constraints grows.
    - Complexity in Setup: Requires careful setup to ensure that constraints do not conflict or render the problem unsolvable.

Maximal Marginal Relevance

- Idea: MMR aims to balance relevance and diversity within a summary by selecting sentences that are both informative and novel compared to those already chosen
    - At each step, select the sentence that offers the greatest “marginal relevance”, defined as highly relevant to the document but not redundant

with the summary's existing content

    - This is typically operationalized as a combination of maximizing relevance to the document and minimizing similarity to sentences already in the summary

- Pro
    - Balance of Relevance and Novelty: Effectively manages the trade-off between these two crucial aspects.
    - Adaptable to Different Content: Can be used across various types of text documents.
- Cons
    - Iterative Complexity: Each selection requires recalculating marginal relevance for all remaining sentences, which can be computationally expensive.
    - Dependency on Relevance and Similarity Measures: The effectiveness of MMR heavily depends on the methods used to measure relevance and similarity

Pro vs Cons

- Pro
    - Good Performance: These methods often produce summaries that are both informative and concise.
    - Integration with ML: Can incorporate machine learning techniques to enhance objective functions and constraints.
- Cons
    - Good Performance: These methods often produce summaries that are both informative and concise.
    - Integration with ML: Can incorporate machine learning techniques to enhance objective functions and constraints.

Item-set based

Item-set based summarization is a technique that identifies frequent itemsets, or commonly co-occurring items(words or phrases), in documents to generate summaries. This approach relies on the notion that the importance of content can be estimate by the frequency and co-occurrence of its components

How it works

- Identification of Frequent Itemsets: The method involves analyzing documents to identify sets of words or phrases that frequently appear together across the text. This is similar to market basket analysis in data mining, where frequent itemsets are used to find commonly bought items together.
- Construction of Summaries: Once these itemsets are identified, sentences that contain a high number of these frequent itemsets are chosen for the summary. The rationale is that sentences including common itemsets likely cover key themes or topics of the document

Pro vs Cons

- Pro
    - High Performance: Itemset-based methods can produce high-quality summaries by focusing on the most frequently mentioned or thematically significant terms and phrases within a document. This ensures that the resulting summary is rich in content that best represents the overall themes of the text.
    - Model Explainability: This approach offers a clear explanation of why certain sentences are chosen for the summary — they contain high-frequency itemsets that are deemed important according to the data-driven analysis. This transparency in the selection process helps in understanding and justifying the contents of the summary.
    - Language Independence: As these methods primarily rely on statistical measures (like frequency of terms and their co-occurrence), they are not tied to any specific language's syntactic or grammatical rules. This makes them adaptable and effective across different languages, provided the necessary text preprocessing is handled accordingly.
- Cons

- High Computational Complexity: Detecting frequent itemsets can be computationally intensive, especially as the size of the text corpus increases. The need to scan the text multiple times to count itemsets and verify their frequency can lead to significant computational overhead.
- Limited Scalability: While itemset-based methods are inherently scalable to some degree, their performance and efficiency can degrade as the volume and complexity of the dataset increase. The method's reliance on comprehensive frequency analysis can become a bottleneck in processing very large datasets or in environments where computational resources are constrained.

LSA-based

Latent Semantic Analysis is a technique that uses mathematical modeling to identify patterns in the relationships among the terms and concepts contained in an unstructured collection of text. LSA aims to reduce the dimensionality of a textual data by constructing a semantic space where texts with similar meanings are placed close to each other

How it works

- Matrix Construction: LSA starts by constructing a term-document matrix, where rows represent unique terms in the corpus, and columns represent the documents. Entries in this matrix typically contain the frequency of each term in each document
- Singular Value Decomposition: This matrix is then decomposed using SVD, which helps to identify patterns in the relationships between the terms and documents, reducing the number of dimensions while preserving the similarity structure
- Summary Extraction: Sentences or terms that are central in the semantic space are chosen to construct the summary, as they are considered to capture the main topics or concepts of the text

Pro vs Cons

- Pro

    - Simplicity: LSA involves straightforward linear algebraic operations, which are well-understood and can be efficiently implemented with existing libraries.
    - Scalability: Once the initial term-document matrix is set up, the decomposition and summarization processes can handle large datasets effectively.
    - Language Independence: LSA does not depend on any specific language features and works on the statistical properties of the term-document matrix, making it applicable to any language.
- Cons
- It considers only word-level relations: LSA tends to focus on the presence and frequency of words, overlooking the deeper syntactic or semantic relationships between them. This might result in summaries that are semantically coherent but may miss nuanced meanings or contextual subtleties.

Neural summarization - Extractive Scenario

Extractive Text summarization using NNs transforms the task into a binary classification problem, where each sentences in the document is classifies as either being part of the summary or not. This approach leverages the capabilities of deep learning models to understand and evaluate the importance of each sentence within its contextual relevance to the document

BERTSum

BERTSum is an adaptation of the BERT model, specifically tuned fro the summarization task. It extends the standard BERT model by applying a classification layer on top of the output embeddings that BERT produces for each sentence. This allows BERTSum to determine the probability of each sentence being important enough to include in the summary

- Sentence Representation: Each sentence is fed into BERT to generate contextual embeddings. Special tokens like `[CLS]` are often added to sentences to help in distinguish them.

- Importance scoring: A classifier layer is applied to the embeddings of the `[CLS]` tokens to predict the importance of each sentence
- Sentence Selection: Sentences with the highest importance scores are selected to form the summary, adhering to constraints such as maximum summary length
- Pro
    - High Accuracy: BERT's deep contextual understanding allows BERTSum to accurately assess the relevance and importance of sentences.
    - Context Awareness: By leveraging BERT, this model takes into account the broader document context, which enhances the quality of the summary by maintaining topic consistency.
- Cons
    - Computational Intensity: BERT models are resource-intensive, requiring significant computational power and memory, which can be a limitation in resource-constrained environments.
    - Length Limitation: BERT has a token limit (typically 512 tokens), which can constrain its ability to process longer documents directly.

MatchSum

MatchSum focuses on selecting sentences that best match the reference summaries, assuming availability of such references during training. It works by comparing candidate sentences or groups of sentences to the reference summaries and selecting those that have the highest semantic similarity.

- Candidate Generation: Generates multiple candidate summaries by selecting different combinations of sentences from the document.
- Similarity Scoring: Each candidate summary is compared to a reference summary (or summaries) using semantic similarity metrics (like cosine similarity between sentence embeddings).
- Summary Selection: The candidate summary with the highest similarity score to the reference summary is selected as the final output.
- Pro

    - Semantic Matching: By focusing on matching to reference summaries, MatchSum can generate summaries that are stylistically and thematically similar to human-generated summaries.
    - Flexibility: Can adapt to different summary styles based on the reference summaries used during training.
- Cons
    - Dependence on Reference Summaries: The performance heavily depends on the quality and representativeness of the reference summaries used during training.
    - Scalability Issues: Generating and evaluating multiple candidate summaries can be computationally expensive, especially for longer documents.

Neural summarization - Abstractive Scenario

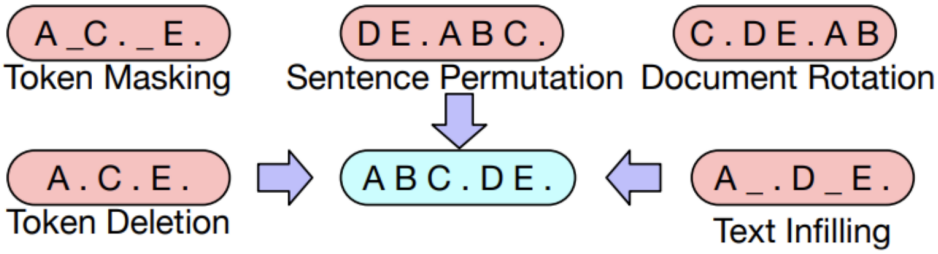
Abstractive text summarization using neural approaches involves generating a concise and coherent summary that may use new phrases and sentences not found in the original text. This task is commonly treated as a sequence-to-sequence (Seq2Seq) problem, where the input sequence (the original text) is transformed into an output sequence (the summary). The focus in abstractive methods is on understanding the overall content and then expressing the main ideas in a new form, which can require sophisticated language models like BART and PEGASUS.

BART

BART is a transformer model that combines both bidirectional and autoregressive transformers. It is designed for sequence-to-sequence tasks and has been particularly influential in tasks that require text generation or transformation, such as summarization.

- Pre-training: BART is pre-trained in a denoising autoencoder setting. The training process involves corrupting text with a noise function (such as token masking, text infilling, sentence permutation, and document rotation) and then learning to reconstruct the original text. This pre-training helps the model

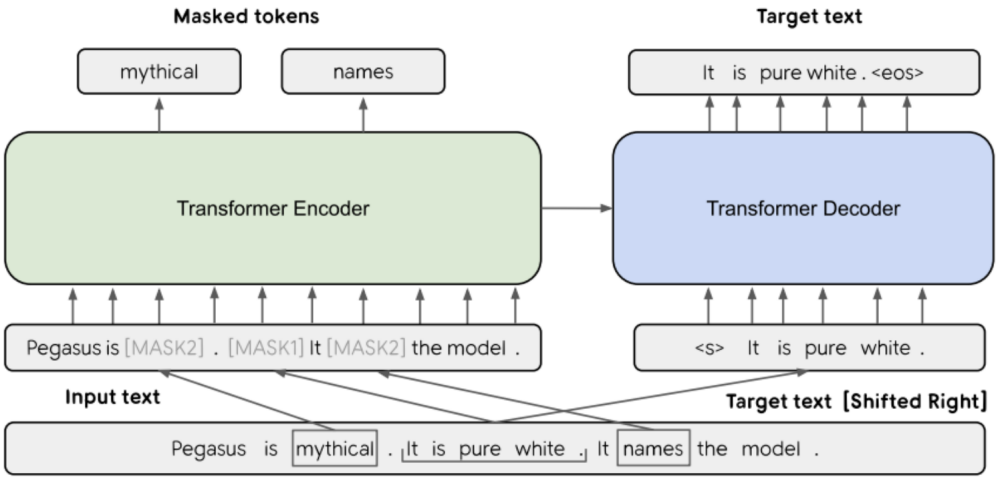
understand the structure and semantics of language, preparing it for generation tasks.



- Fine-tuning: For text summarization, BART is fine-tuned on a dataset of document-summary pairs. During this phase, the model learns to condense articles into summaries by understanding the mapping from the "noisy" input document (as seen during pre-training) to the concise summary output.
- Advantages
  - Deep Understanding: Thanks to its bidirectional encoder, BART is good at understanding context and dependencies in the text.
  - Flexible Generation: Its autoregressive decoder excels at generating coherent and contextually appropriate text.
- Disadvantages
  - Resource Intensive: Like most large transformer models, BART requires significant computational resources for both training and inference.
  - Risk of Hallucination: Sometimes, the model may generate plausible but incorrect or unfaithful information in the summary

PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive SUmmarization Sequence-to-sequence)

PEGASUS is specifically designed for abstractive text summarization and introduces a novel pre-training objective called "gap sentences generation"



- Pre-training: In gap sentences generation, important sentences are removed/masked from an input document and are then treated as a summary target. This simulates a summarization task during pre-training, where the model learns to generate these “gap” sentences from the remaining content of the document. The sentences for removal are selected based on their importance, determined by their centrality to the document.
- Fine-tuning: Similar to BART, PEGASUS is fine-tuned on a summarization-specific dataset. The model adjusts from generating gap sentences to generating actual summaries, refining its ability to distill and express the main points of full documents concisely.
- Advantages
  - Highly Specialized Pre-training: The unique pre-training approach makes PEGASUS particularly effective for summarization.
  - Efficiency in Learning: By focusing on summary-like sentence generation during pre-training, it requires potentially less adaptation during fine-tuning.
- Disadvantages
  - Pre-training Complexity: Selecting the right sentences to mask and reconstruct during pre-training requires careful design and can impact model performance.

- Dependency on Quality of Training Data: The effectiveness of the model is highly contingent on the representativeness and quality of the document-summary pairs used in training.

Timeline Summarization

Timeline summarization is a specialized form of text summarization aimed at creating a chronological timeline of events from a series of documents. This approach is particularly useful in scenarios involving news articles, historical documents, or any data set where events evolve over time. The objective is to provide a concise, temporal narrative that highlights the progression and key events within a given topic or story

Pipeline for Timeline Summarization:

The process of timeline summarization typically involves several stages:

1. Document Collection: Gathering all relevant documents that pertain to the topic of interest. These documents are often filtered by date to ensure chronological processing.
2. Preprocessing: Standard NLP tasks such as tokenization, removing stop words, and possibly part-of-speech tagging. This stage may also involve named entity recognition (NER) to identify key entities like people, locations, and dates.
3. Event Detection: Identifying distinct events within the corpus. This can be achieved through clustering techniques where similar documents or sentences are grouped together, or by identifying key sentences that represent unique events.
4. Event Linking and Ordering: Linking related events and placing them in a chronological order. This might involve analyzing the timestamps of documents or the temporal expressions within the text.
5. Summarization: Generating concise descriptions for each identified event. This may be extractive, where key sentences are pulled directly from the text, or abstractive, where new sentences are generated to summarize the events.

6. Timeline Creation: Organizing the summarized events into a coherent timeline. This step often involves visual or textual presentation that shows the progression of events over time.

Graph with Time Relation

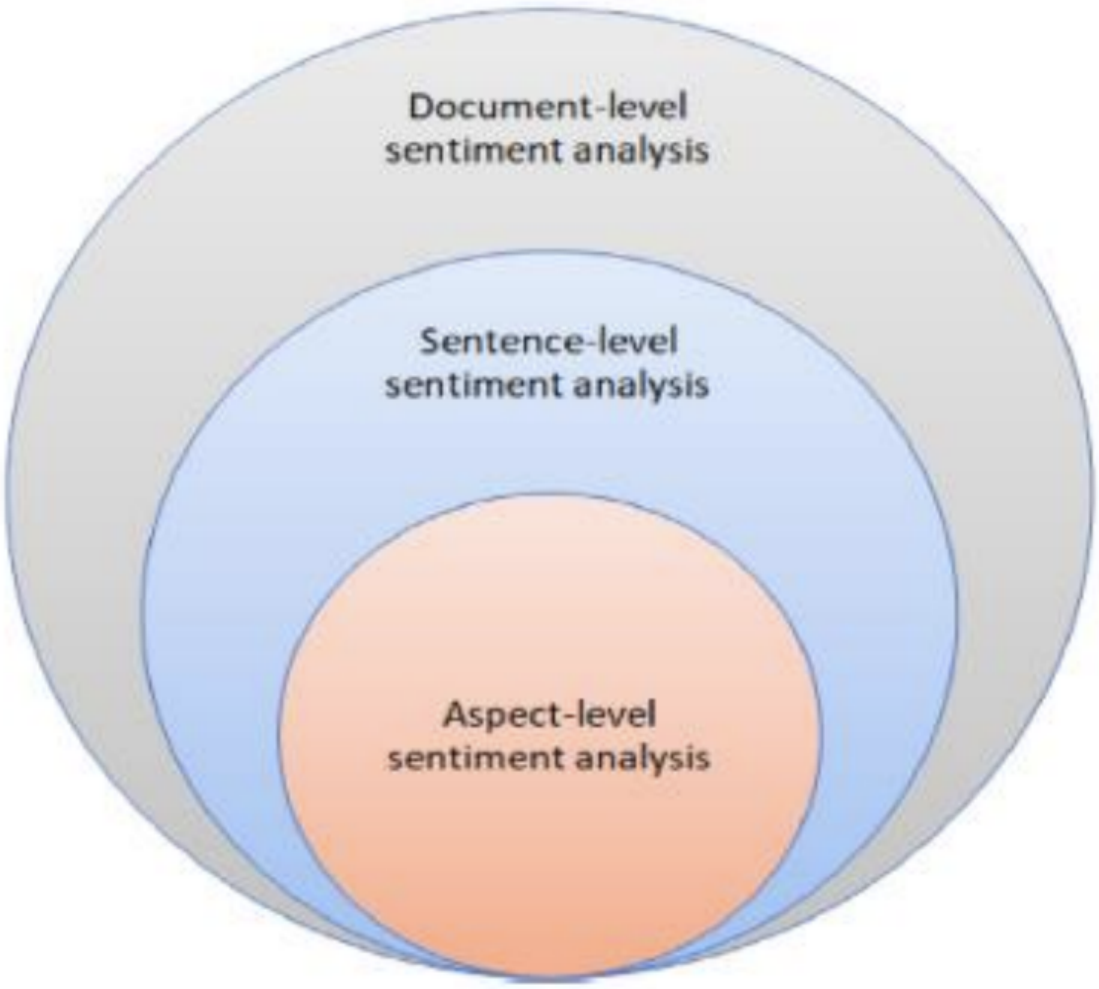
Incorporating a graph-based approach in timeline summarization can significantly enhance the analysis and representation of events:

- Graph Construction: Building a graph where nodes represent events or key concepts, and edges represent the relationships between these events (such as causal relationships, succession, or correlation). Time-stamped edges can be used to show the progression from one event to another.
- Temporal Analysis: Using the graph to analyze how events are interlinked over time. Techniques like path analysis or flow analysis can help in understanding the development of topics or stories.
- Dynamic Summarization: Updating the graph dynamically as new information becomes available. This is crucial for topics that evolve over time, such as ongoing news stories.
- **Pros**
  - Provides a structured summary that illustrates the development of events over time.
  - Helps in understanding complex stories or topics by highlighting major events and their connections.
- **Cons**:
  - Requires sophisticated algorithms to accurately detect, link, and order events.
  - Managing the graph's complexity, especially with large volumes of data or long time spans, can be challenging.

Sentiment Analysis

Sentiment analysis is a crucial task in Natural Language Processing (NLP) where the goal is to determine the sentiment expressed in a piece of text. Typically categorized as a form of text classification, this task aims to label the text according to the sentiment it conveys, such as positive, negative, or neutral.

Levels and applications of Sentiment Analysis



1. Document-level Sentiment Analysis

   This level of analysis considers the overall sentiment of an entire document, which could be anything from a tweet or a review to an entire article. The task here is to determine the predominant sentiment expressed throughout the

document. For instance, analyzing a product review to decide if the overall sentiment is positive, negative, or neutral. This approach assumes uniform sentiment across the document.

2. Sentence-level Sentiment Analysis:
   Sentence-level analysis breaks down the document into its constituent sentences and analyzes each one independently to determine its sentiment. This is useful for texts where different sentences express different sentiments, such as in a detailed movie review discussing various aspects like plot, acting, and direction. Each sentence is treated as a separate entity, and the sentiment is classified accordingly.
3. Aspect-level Sentiment Analysis:
   Also known as aspect-based sentiment analysis, this approach is more nuanced and sophisticated. It not only determines the sentiment but also links it to specific aspects or features within the text. For example, in a restaurant review, the sentiment might be positive when talking about the ambiance but negative regarding the service. Aspect-level analysis identifies these nuances by parsing the text for sentiment expressions specifically tied to defined aspects or features.

Use cases

- Hotel review analysis
- News trading
- Hate speech detection
- Advertisement placing

- ...

Applications

- Business intelligence
  - Make product/service improvement
  - study the customers' feedback
  - Plan new marketing movements
- Recommender systems
  - Suggest relevant items to users
- Government Intelligence
  - Identify opinions on government policies
  - Monitori publics' mood in real time
- Healthcare
  - Shape Healthcare services
  - Allow healthcare actors to obtain information about diseases, epidemics, treatments, patients' mood, etc

Data sources

- Social media
  - Analysis of user-generated content
  - Individual and collective life and behavior
  - Web- and mobile-based internet applications
- Review websites
  - Where people can post opinions or reviews about a particular entity
  - Often integrated into e-commerce platforms
- Forums
  - Posting text messages to share thoughts and ideas

- Interviews
  - Transcripted record of completed oral interviews
  - Either in real time or from audio or video recordings

Preprocessing

- Tokenization: Necessary for sentence- and aspect-level sentiment analysis
- Stopword removal: unnecessary when embedding models are applied. It also can be hurtful when remove negation
- Text Cleaning: Abbreviations/repetitions/punctuation can be relevant to sentiment detection

- N-gram Presence and Frequency
  - In sentiment analysis, n-grams help capture context within the text that may indicate sentiment. For instance, bi-grams (two-word sequences) like “not good” or “very good” can signify different sentiments.
  - Weighting Schemes: Techniques like TF-IDF (term frequency-inverse document frequency) and BM25 are used to assign weights to n-grams based on their importance in distinguishing sentiment across documents.
- POS Tagging: Part-of-Speech tagging involves assigning parts of speech to each word in a text, like verbs, nouns, adjectives, etc., based on both its definition and its context.
  - Importance in Sentiment Analysis: Adjectives and adverbs are often crucial indicators of sentiment. Words like “excellent” or “terrible” directly contribute to understanding the sentiment expressed.
- Opinion Words/Phrases: These are words or phrases that typically convey sentiment. Examples include “amazing” for positive sentiment and “horrible” for negative sentiment.
  - Domain Dependence: Some expressions may have different sentiment implications in different contexts. For example, “unpredictable” in a movie review might be positive, but in a financial report, it could be negative.

- Negations: Words like “not”, “never”, and “no” can invert the sentiment of the words that follow them. For example, “not good” has a negative sentiment, whereas “good” is positive.
    - Challenges: Negations can be tricky for automatic sentiment analysis systems, especially if negation words are removed during preprocessing like stopword removal.
- Feature Relevance
    - Distinguishing Relevant vs. Irrelevant/Redundant Features: It’s crucial to identify and use features that are actually informative for sentiment analysis while ignoring or removing those that do not contribute meaningful information.
- Selection Methods
    - Lexicon-based: This involves using predefined lists of words known to convey sentiment. Tools like WordNet can be used to expand these lists through synonyms. This method is precise but can be expensive and inflexible.
    - Statistical Methods: These involve algorithms that automatically select features based on statistical criteria. This can be more scalable and efficient but might also introduce errors if important contextual clues are missed.

Methods

- Supervised Learning: Utilizes labeled datasets to train models that can classify sentiment in new texts. Common algorithms include linear classifiers, decision trees, and rule-based systems. In sentiment analysis, these models are trained with samples of text labeled as expressing positive, negative, or neutral sentiments.
- Unsupervised Learning: Applied when labeled data isn’t available. Techniques like clustering are used to group texts with similar sentiments based on their intrinsic properties. For example, hierarchical clustering might reveal naturally occurring sentiment groups in product reviews.

- Semi-supervised Learning: Combines a small amount of labeled data with a large amount of unlabeled data. The process often involves self-training, where a model initially trained on a small labeled dataset is used to label the unlabeled data, which is then used for further training.
- Generative Models: Assume that different sentiment classes are generated from different underlying distributions. These models aim to learn the parameters of these distributions to classify sentiments. For instance, a generative model might learn the typical word distributions in positive vs. negative movie reviews.
- Co-training: This approach assumes that the data can be split into two views (e.g., textual content and accompanying metadata), each providing different, complementary information about the sentiment. Separate classifiers are trained on each view and help improve each other iteratively.
- Self-training: A form of semi-supervised learning where a classifier is initially trained on a small labeled dataset and then used to classify and retrain using unlabeled data, assuming that different sentiment classes follow distinct distributions.
- **Graph-based Models**: Represent the data as a graph, where each node is a data point (e.g., a document or sentence), and edges represent similarities. This can help in sentiment classification by leveraging the similarity between documents to propagate sentiment labels across the graph.
- Multi-view Learning: Considers multiple perspectives or features of the data to train separate models, whose outputs are combined to improve the overall sentiment classification. This can be especially useful when different features provide unique insights into the sentiment expressed.
- Reinforcement Learning: Applies a trial-and-error learning method where an agent learns sentiment classification by receiving rewards for correct predictions. This approach can dynamically adapt to new forms of sentiment expression over time.
- Transfer Learning: Useful for applying knowledge gained from one domain (e.g., movie reviews) to another (e.g., book reviews), especially when direct training data for the target domain is sparse.

- Multimodal Learning: Integrates multiple types of data (e.g., text, audio, video) to perform sentiment analysis. This is particularly relevant in scenarios where sentiments are expressed across multiple communication channels, requiring the fusion of diverse data types to understand the overall sentiment

Attention based

- Key Idea: In aspect-based sentiment analysis, the relationship between aspects (like specific features of a product) and opinion words (like "great" or "poor") is crucial. Attention mechanisms can implicitly learn to focus on these relationships without explicit aspect labeling in the training data.
- Implementation: Neural network architectures, typically those based on transformers or LSTM (Long Short-Term Memory) units, use attention layers to weigh the importance of each word in the context of an aspect. For instance, in a review like "The battery life is long, but the screen is too dim," the attention mechanism can learn to associate "long" with "battery life" and "dim" with "screen" effectively.
- Explainable Aspect-Based Analysis
    - Advantages: One of the major strengths of attention-based models is their explainability. By examining the attention weights, researchers and practitioners can identify which words the model focuses on when making sentiment decisions related to specific aspects.
    - Tools: Techniques such as heat maps over the text can visualize how each word contributed to the sentiment score of an aspect, offering insights into model decisions and allowing for fine-tuning and error analysis.
- Combination with NLP Techniques: Combining attention with dependency parsing can enhance the model's ability to understand the syntactic relationships between words, which is essential for accurately linking opinion words to their respective aspects.
    - Example Application: In sentences with complex structures, the model uses dependency trees to better grasp the grammatical relationships, ensuring that the sentiment "dried out" is accurately associated with "falafel" and not incorrectly linked to a positive aspect like "the chicken was fine."

Machine Translation

Machine translation is a significant task in the field of NLP, focusing on the automated translation of text or speech from one language to another. It's a complex task as it involves understanding and generating text while maintaining the meaning, style, and cultural nuances.

Rule- or Dictionary-Based Machine Translation

The earliest approach to machine translation is rule-based or dictionary-based, also known as knowledge-based machine translation. This method relies heavily on linguistic data and rules:

- Linguistic Rules: It uses a comprehensive set of rules about grammar and syntax of both source and target languages. For instance, rules about verb conjugations, noun-adjective agreements, and sentence structure are manually coded into the system
- Dictionaries: This approach employs large dictionaries with translations for each word or phrase. Specialized dictionaries may also be used for different fields or context
- Translation process: During translation, the text is first parsed and analyzed according to the source language's rules. The system then uses the rules and dictionaries to convert the analyzed text into the target language structure. This often includes reordering phrases to match the syntactic norms of the target language

The strength of rule-based MT lies in its ability to reliably apply grammatical rules, which can be particularly effective for languages with less resources. However, its rigidity is also its downfall, as it struggles with ambiguities and nuances that don't fit neatly into predetermined rules

Statistical Machine Translation

This approach uses statistical models to generate translations based on the analysis of large amounts of bilingual text data

- Data-Driven: Statistical approach does not rely on linguistic rules but on statistical methods. It require large corpora of aligned texts (texts that are

translations of each other) from which it learns how to translate

- Bayes' Theorem: The actual translation process in SMT can be framed as a statistical decision process where Bayes' theorem is used. The goal is to maximize the probability of a translation given the original sentence, combining the translation model and the language model
- Translation Model: The translation model first determines probable translations of words and phrase based on bilingual text data. This involves creating word alignments which are associations between words in the source text and words in the translated text. These alignments can be:
    - Many-to-One: Multiple words from the source language map to a single word in the target language. For example, The english phrase "a lot of" might translate into a single word in another language
    - One-to-Many: A single word in the source language could translate into a multiple words in the target language. For example, "the" in english might need to be translated into a language that uses grammatical gender as "lo" or "la" in Italian, depending on the context.
    - Many-to-Many: Complex phrases or idioms that don't translate directly on a word-for-word basis but rather as whole units. For example, "out of the blue" in English might translate into a completely different idiom in another language

  The translation model uses these alignments to create a statistical model that can predict the likelihood of a particular translation being accurate, given a specific source text.
- Language Model: The language model is designed to ensure that the output text is fluent and grammatically correct in the target language. Its primary responsibilities include:
    - Fluency: The language model evaluated how likely a sequence of words is to appear in the target language. This is generally calculated using the probability of a word or a sequence of words appearing after a given sequence of words
    - Grammatical Correctness: By analyzing large amounts of monolingual text in the target language, the language model learns the typical patterns and

structures of the language. This includes common phrases, typical sentence structures, and other linguistic characteristic

Neural Machine Translation

Neural Machine Translation represents a significant shift in machine translation, employing deep learning models to improve translation accuracy and efficiency.

- Sequence-to-Sequence: seq2seq is designed to handle sequences of input data and generate sequences of output data. This model typically consists of two parts:
  - Encoder: The encoder read the input sentence, one token at a time, and transforms it into a context-rich representation. This representation encapsulates the entire input sentence, capturing its semantic and syntactic properties
  - Decoder: starting from the encoded representation, the decoder generates the output sentence, one token at a time. It uses the context from the encoder along with what it has already generated to predict the next word
- Stacked RNNs: Within the seq2seq framework, Recurrent Neural Networks are often employed both in the encoder and the decoder components. Stacked RNNs, where multiple layers of RNNs are places on top of each other, are used to enhance the learning capabilities of the model. Each layer captures different levels of abstraction, which can lead to a more nuanced understanding and generation of language

Encoding Strategies

- Greedy Encoding: Here, the model chooses the most likely next word at each step in the decoding process. While fast, this method can lead to suboptimal overall translations because early mistakes cannot be corrected later.
- Exhaustive Encoding: This approach attempts to evaluate all possible translations exhaustively to find the best one by maximizing a given metric. This is generally impractical due to the immense computational cost.
- Beam Search Encoding: Beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set. In the

context of machine translation, it's used during the decoding phase of the translation process.

- Selective Exploration: Unlike greedy decoding, which only considers the single best option at each step (leading to potentially suboptimal overall results), beam search keeps track of a set number of the best options at each step, known as the "beam width."
- Beam Width: This is a crucial parameter in beam search. A wider beam allows the algorithm to explore a broader range of possible translations at each step, increasing the likelihood of finding a higher-quality overall translation. However, a wider beam also requires more computational resources and time.
- Pruning: As the beam search progresses, it prunes less promising paths, focusing computational resources on extending only the most promising candidates. This pruning is essential for managing the exponential growth of possible sequences and making the problem tractable.
- End Conditions: The search can end under several conditions, such as reaching a maximum sentence length, all sequences reaching end-of-sequence tokens, or after finding a predetermined number of high-scoring sequences.

The advantage of beam search over exhaustive search is that it balances between breadth and depth, exploring a manageable number of options at each step but more than just the top one. This method often leads to better translations than greedy decoding, as it can bypass local maxima

BLEU Score

BLEU, or Bilingual Evaluation Understudy, is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another.

- N-gram Comparison: BLEU evaluates the quality of translation by comparing the n-grams of the candidate translation with the n-grams of the reference translation(s) and counting the number of matches. These comparisons are made using different n-gram lengths (typically up to 4), capturing the fluency and accuracy of the translation at various levels.

- Precision: BLEU calculates the precision for each n-gram (the proportion of n-grams in the translated text that appear in the reference text), which helps in assessing the translation's adequacy and fluency.
- Brevity Penalty: To counteract the bias toward shorter translations (since shorter texts can achieve higher precision more easily), BLEU includes a brevity penalty. If the candidate translation is shorter than the reference, this penalty reduces the BLEU score, encouraging translations to not only be precise but also complete.
- Composite Score: The final BLEU score is a weighted average of the precision scores for different n-gram lengths, adjusted by the brevity penalty. This composite nature allows BLEU to be a balanced measure, considering both the richness of the translation and its appropriateness in length and structure.

BLEU is widely used because it is language-independent and correlates moderately well with human judgment. However, it's not without criticisms—it doesn't account for the semantic appropriateness of a translation and can undervalue linguistic and stylistic nuances. Despite these limitations, BLEU remains a standard metric for evaluating machine translation due to its simplicity and effectiveness.

Pro vs Cons

- Pro
  - Text fluency
  - Better contextualization
  - No sub-components to be optimized
  - Limited human engineering effort
  - Same method for all language pairs
- Cons
  - Hard to debug
  - Hard to control
  - Out-of-vocabulary words

- Domain mismatch from train to test data
- Low-resource language pairs
- Pronoun resolution errors
- Morphological agreement errors

BART in Machine Translation

BART is originally designed as a pre-trained model that combines bidirectional encoding (like BERT) with auto-regressive decoding (like GPT). For machine translation, modifications are made to its architecture and training process to better suit the task of translating text from one language to another.

1. Integration of a New Encoder
    - Replacing Encoder Embeddings: BART's original encoder is modified by replacing its embedding layer with a new, randomly initialized encoder. This allows the model to better adapt to the specifics of the source language in the translation task.
    - End-to-End Training: The modified BART model, with its new encoder, is trained from scratch (end-to-end). This training involves adjusting both the new encoder and the pre-existing decoder to work harmoniously, optimizing the model for translating specific language pairs.
    - Separate Vocabulary: Often, the new encoder will use a separate vocabulary from the original BART model. This is particularly useful when the source and target languages have distinct linguistic properties or alphabets.
2. Utilization of the Entire BART Model
    - Single Pretrained Decoder: In another approach, the entire BART model (both encoder and decoder) is used as a unified framework. A new set of encoder parameters is introduced, which are learned specifically from bilingual corpora. This approach retains the powerful decoding capabilities of BART while enhancing its encoding strength for translation tasks.
    - Learning from Bilingual Corpus: The additional encoder parameters are tuned based on bilingual text, enabling the model to learn effective

translation patterns and nuances.

3. Learning a Small Additional Encoder
    - Replacement of Word Embeddings: Instead of completely overhauling the encoder, another approach involves learning a small additional encoder that replaces only the word embeddings in BART. This smaller, more focused change allows for adjustments in how input words from the source language are represented, catering specifically to translation needs.
    - Disjoint Vocabulary: This additional encoder might use a disjoint vocabulary, which means it could have a completely different set of tokens from what is used in the original BART model. This is useful for handling languages with very different linguistic structures or scripts.

Large Language Model

Autoregressive Language Model

Autoregressive language models predict subsequent tokens based on the tokens that precede them. Each token generated influences the prediction of the next, creating a chain-like effect. This approach is fundamental to models like GPT, where the sequence of text generation follows a casual, one-directional pattern. By training on extensive datasets, these models learn complex patterns in language and can generate coherent and contextually appropriate text

Temperature

The concept of "temperature" in language models refers to a parameter that controls the randomness of predictions by scaling the logits before applying softmax during text generation. A lower temperature makes the model's output more deterministic and confident, often sticking closer to the most likely outcomes. Conversely, a higher temperature increases diversity and creativity in the output, though it may sometimes sacrifice coherence. Adjusting temperature allows users to balance between creativity and accuracy based on the application's need

- Temperature = 0 deterministically choose the most likely token at each step
- Temperature = 1 sample from the pure language model using a normal distribution
- Temperature = ∞ sample from the uniform distribution over the entire vocabulary

Conditional Text Generation (Prompting)

Conditional text generation, or prompting, is a technique where the language model generates text based on a given input prompt. This is integral to how models like ChatGPT operate, where the prompt sets the initial conditions and context for the model's output. Effective prompting can guide the model to produce specific styles, formats or content. This adaptability makes LLMs incredibly versatile for a range of applications, from composing emails to generating creative content. Moreover them down into simpler, intermediate steps before arriving at the final output.

Large Language Models

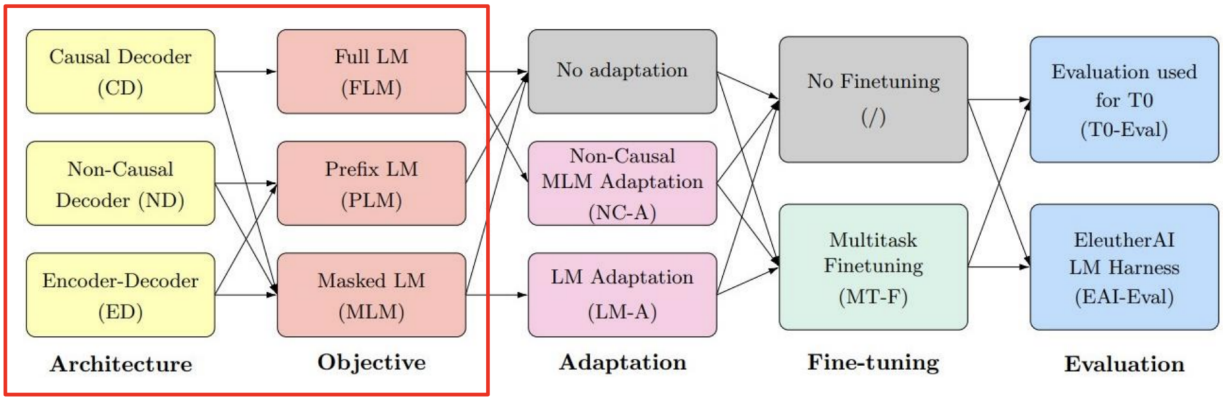
Large Language Models like GPT and BERT are built upon the transformer architecture. These models are characterized by their vast scale, both in terms of the model's architecture and the data they are trained on. Typically, LLMs consist of millions or even billions of parameters, allowing them to capture complex language nuances. Training such models requires a massive amount of data, often sourced from diverse internet texts, which enables them to generalize across various contexts and styles.

LLMs are pretrained using self-supervised and/or semi-supervised learning

- Proprietary LLMs are developed and controlled by specific organizations. They often come with restrictions on usage and are not openly available for customization.
- Open Source LLMs, in contrast, are accessible for use and modification by the general public. This openness fosters a collaborative environment where improvements and innovations can be shared across the community

- Single Language Models are tailored for tasks involving one language. Their focused training on a specific language often result in superior performance for tasks in that language due to the tailored linguistic nuances and idioms
- Multi-language Models are trained on data from multiple languages. This approach enhances the model's versatility but often faces challenges such as data imbalance, where some languages are overrepresented compared to others, potentially skewing the model's performance

Architecture



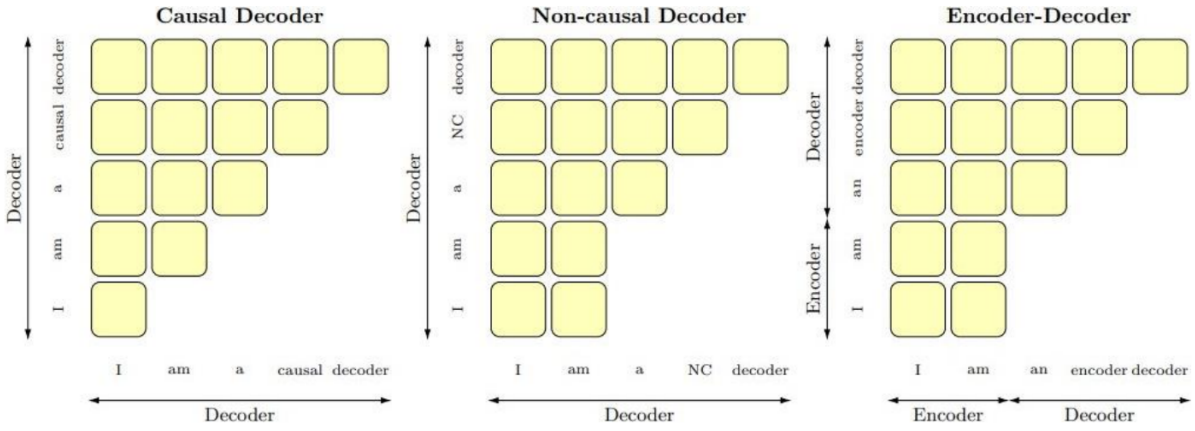
Encoder-Decoder vs Decoder-Only models

- Encoder-Decoder: These models consist of two main parts: an encoder that processes the input text and a decoder that generates the output based on the encoded information.
  The encoder transforms input text into a rich contextual representation and the decoder uses this representation to produce output step-by-step
- Decoder-only: These models are optimized for generative tasks. They operate by processing the input as a continuous stream of tokens using self attention mechanism, which means each token can only attend to previously generated tokens during training or inference
  This architecture simplifies the model and often results in faster training and inference times compared to encoder-decoder

Notably, state-of-the-art LLMs with billions of parameters typically employ a decoder-only structure, achieving impressive performance in zero-shot learning tasks by leveraging the vast amount of pretraining data

Casual vs Non-casual Decoder-only Models

- Casual Decoder: In casual LLMs, each output token is predicted based solely on the previously generated tokens. This mimics the forward flow of human language, making it natural for generating coherent long-form content
  These models are the backbone of many generative LLMs, including the entire GPT line, which relies on this method to produce text that is contextually relevant to the input provided up to that point
- Non-Casual Decoder: These models differ in that the self-attention mechanism is not strictly limited to previous tokens; they can consider a broader context or event the entire input sequence at once. This approach can be advantageous for tasks requiring a holistic understanding of the input data before generating any part of the output
  However, the non-casual approach is less common because it can potentially lead to issues like information leakage where future tokens influence the prediction of current tokens, which is not desirable in many generative tasks



Pre-Training objective

- Full Language Modeling
  - Objective: The model is trained to predict the next token in a sequence based on the previous tokens. This is the simplest form of language

modeling and is foundational to understanding how sequential data can be processed by neural networks

    - Usage: Commonly used in models like GPT (Generative Pre-trained Transformer), where the entire training process revolves around predicting the next word in a sentence given all previous words, thus learning the context and the structure of language.

- Prefix Language Modeling
    - Objective: In Prefix LM, a non-causal attention mask is used where the model can look at an initial set of tokens (the prefix) and is trained to predict the following tokens considering the entire prefix. This allows the model to have a broader context during the prediction phase, which can be beneficial in tasks where understanding the entire input is crucial before making a prediction.
    - Usage: This method is particularly useful for tasks like sentence completion where the initial part of the sentence sets the context for the remaining part, or in scenarios where the model needs to generate text based on a given topic or keyword.
- Masked Language Modeling
    - Objective: In Masked LM, random tokens in a sentence are replaced with a mask token, and the model is trained to predict these masked tokens based only on their context. This training methodology allows the model to focus more on understanding the context and relationships between words rather than just predicting the next word.
    - Usage: Widely used in models like BERT (Bidirectional Encoder Representations from Transformers), which are particularly powerful in understanding the context of a token in a bidirectional way, thus improving performance on various NLP tasks like question answering and named entity recognition.

LLM Adaption

Adapting LLMs to specific tasks is a crucial step in leveraging their general capabilities for targeted applications. This process involves modifying a pre-

trained model so that it can perform well on a particular task, which can vary significantly from the tasks the model was originally trained on.

The pre-training of LLMs involves large, diverse datasets that help the model learn a wide range of linguistic features and general knowledge. However, these models are often trained in a task-agnostic manner

Implementation Strategies

- Formatting: Adapting the input or output format of the model to match the specific requirements of a task, such as adjusting the model to output structured data from unstructured input.
- Domain and Temporal Adaptation: Shifting the model's focus to specific types of content or updating its knowledge to reflect recent information not present in the initial training set.

Training techniques

1. Probing: Probing involves adding a small, new task-specific model or layer on top of a frozen pre-trained model. The underlying LLM's parameters are not updated during training; instead, the probe learns to make prediction based on the features extracted by the LLM
2. Fine-Tuning: Adjust all or most of pre-trained model's weights on a specific task using a task-specific dataset. This technique is more resource-intensive but can lead to significant improvements in task performance because it tailors the model's parameters to the nuances of the specific task
3. Lightweight Fine-Tuning: Involves updating only small fraction of the model's parameters. This might include training a few layers or just the parameter of specific components of the model, such as the attention heads

Prompting

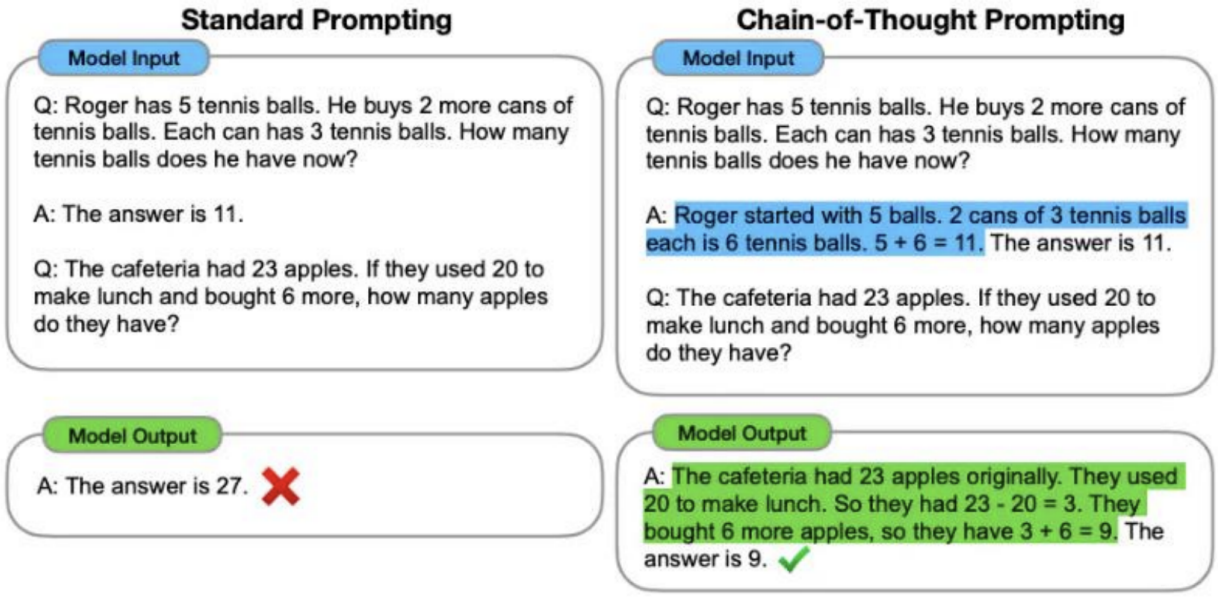
Prompting (or in-context learning) is a powerful method for adapting LLMs to specific tasks without the need for extensive retraining. By crafting a prompt or a series of prompts that describe the task at hand or provide examples of desired behavior, the LLM can generate appropriate responses or completions based on

the context provided in the prompt. this method leverages the model's pre-trained knowledge, applying it to new scenarios effectively and efficiently.

- Zero-Shot Learning: the model generated a response based on a prompt without any prior specific examples of the task. It relies entirely on its pre-trained knowledge and understanding of the task instructions.
- One-Shot Learning: the model is provided with a single example at inference time to guide its response. this example acts as a reference point, helping the model understand the task context or desired output format
- Few-Shot Learning: Similar to one-shot but with a few examples provided. This method typically yields better performance that zero-shot or one-shot as it gives the model more context about the variability and scope of the task

- Single Task: The model is prompted or fine-tuned to perform one specific task. This focus can lead to higher performance in that task due to the specialized nature of the training or prompting.
- Multi-Task: The model is prompted or fine-tuned to handle multiple tasks. This approach improves the model's generalizability and flexibility, as it learns to apply its knowledge across different types of tasks and datasets

Chain-of-thoughts prompting is an advanced technique where the prompt includes not only the query but also a simulated reasoning process that leads to the answer. This method helps the model to break down complex problems into intermediate steps, significantly improving its ability to handle tasks that require deep reasoning, such as math word problems, complex decision-making, or reasoning over multiple pieces of information. This approach is shown to improve performance across various tasks and datasets, as it aligns the model's processing more closely with how humans tackle complex problems



Retrieval augmented generation

Retrieval Augmented Generation (RAG) is an innovative approach that enhances the capabilities of Large Language Models (LLMs) by integrating them with content retrieval systems. This hybrid method not only utilizes the inherent knowledge of the LLM but also leverages external, relevant information from a corpus to generate responses

How RAG Works

1. Document Retrieval: When a prompt is received, the RAG system first identifies and retrieves relevant documents from a connected database or corpus. This retrieval is typically powered by vector embeddings, where documents are converted into vector forms that can be efficiently searched to find matches to the query.
2. Context Integration: The retrieved documents are then concatenated with the original prompt to provide a rich, augmented input context to the LLM.

3. Response Generation: This augmented prompt is fed into a sequence-to-sequence (seq2seq) model, which generates the final output. The seq2seq model, therefore, operates with a deeper, broader context than it would if relying solely on its pre-trained knowledge.

Advantages of RAG

- Enhanced Accuracy and Relevance: By pulling in specific information related to the query, RAG systems can provide more accurate and contextually relevant answers than those generated by standalone LLMs.
- Dynamic Knowledge: Unlike traditional LLMs that only rely on the knowledge they were trained on, RAG can access up-to-date information from external sources, keeping the model's outputs current.
- Efficiency in Specialized Domains: For tasks requiring specialized knowledge, such as technical support or medical inquiries, RAG can significantly boost the model's performance by retrieving expert documents on the subject.

ChatGPT

ChatGPT, developed by OpenAI, is a variant of the GPT (Generative Pre-trained Transformer) architecture that has been specifically fine-tuned for conversational applications. It leverages both supervised learning and reinforcement learning from human feedback (RLHF) to improve its performance

Training

1. Supervised Fine-Tuning
    - Data Collection: Initially, ChatGPT is trained on a mixture of licensed data, data created by human trainers, and publicly available data. This diverse dataset helps the model learn a variety of conversational patterns and responses.
    - Training Method: Using these conversations, the model is fine-tuned in a supervised manner where the correct responses are provided. This step is crucial as it sets the foundational understanding of conversation flow, making the responses more contextually appropriate

2. Reinforcement Learning from Human Feedback (RLHF)
    - Collection of Comparison Data: Once the model has a baseline conversational ability, it undergoes a reinforcement learning phase where new outputs are generated based on sampled prompts. These outputs are then presented to human trainers who compare different responses and select the most appropriate or engaging ones.
    - Training the Reward Model: The preferences from human trainers are used to train a reward model. This model learns to predict the "value" or quality of a response based on human feedback, essentially learning what humans consider a good conversation.
    - Policy Optimization: With the reward model established, the ChatGPT policy (or the decision-making part of the model) is optimized using Proximal Policy Optimization (PPO), a reinforcement learning algorithm. This algorithm adjusts the model's parameters to maximize the expected reward from the reward model, essentially teaching the model to generate responses that are more likely to be rated highly by humans.

Other LLMs

- LaMDA

LaMDA

- **La**nguage **M**odels for **D**ialog **A**pplications
- Developed by: Google
- Release date: May 2021 (v1), May 2022 (v2)
- Number of parameters: 137 billion
- Training corpus: 1.56 trillion words
- Paper: https://arxiv.org/pdf/2201.08239.pdf

LaMDA

- Relative attention (variant of classic attention mechanism)
- Model hyper-parameters:

Parameters	Layers	Units	Heads
2B	10	2560	40
8B	16	4096	64
137B	64	8192	128

- Fine-tuning with manually annotated responses for sensibleness, interestingness, safety and groundedness

- PaLM

PaLM

- **Pa**thways **L**anguage **M**odel
- Developed by: Google
- Release date: April 2022 (v1), May 2023 (v2)
- Number of parameters: 540 billion (v1), 340 billion (v2)
- Training corpus: 780 billion (v1), 3.6 trillion tokens (v2)
- Paper: https://arxiv.org/pdf/2204.02311.pdf (v1)
  https://arxiv.org/pdf/2305.10403.pdf (v2)

PaLM

- Transformer modifications (e.g., SwiGLU activation function, parallel layers, multi-query attention)
- Model hyper-parameters:

Model	Layers	# of Heads	d_{model}	# of Parameters (in billions)	Batch Size
PaLM 8B	32	16	4096	8.63	$256 \rightarrow 512$
PaLM 62B	64	32	8192	62.50	$512 \rightarrow 1024$
PaLM 540B	118	48	18432	540.35	$512 \rightarrow 1024 \rightarrow 2048$

- Bard (Google chatbot) was initially based on LaMDA, then upgraded to PaLM

- BLOOM

BLOOM

- **B**igScience **L**arge **O**pen-science **O**pen-access **M**ultilingual Language Model
- Developed by: Hugging Face collaboration
- Release date: July 2022
- Number of parameters: 176 billion
- Training corpus: 366 billion tokens
- Paper: https://arxiv.org/pdf/2211.05100.pdf

BLOOM

- Based on GPT-3, trained on a corpus in 46 natural and 13 programming languages
- Leverages alternative position embedding schemes and novel activation functions
- All models and code are publicly released

- Galactica

Galactica

- Galactica: A Large Language Model for Science
- Developed by: Meta
- Release date: November 2022
- Number of parameters: 120 billion
- Training corpus: 106 billion tokens
- Paper: https://arxiv.org/pdf/2211.09085.pdf

Galactica

- Stores, combines and reasons about scientific knowledge
- Working memory & citation tokens, prompt pre-training
- Model hyper-parameters:

Model	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{heads}	Batch Size	Max LR	Warmup
GAL 125M	125M	12	768	12	64	0.5M	6×10^{-4}	375M
GAL 1.3B	1.3B	24	2,048	32	64	1.0M	2×10^{-4}	375M
GAL 6.7B	6.7B	32	4,096	32	128	2.0M	1.2×10^{-4}	375M
GAL 30B	30.0B	48	7,168	56	128	2.0M	1×10^{-4}	375M
GAL 120B	120.0B	96	10,240	80	128	2.0M	0.7×10^{-5}	1.125B

- Trained using papers, encyclopedias, scientific sources and special tokens for citations, formulas and numbers

- LLaMA

LLaMA

- **L**arge **La**nguage Model **M**eta **A**I
- Developed by: Meta
- Release date: February 2023
- Number of parameters: 65 billion (around 1/3 wrt ChatGPT)
- Training corpus: 1.4 trillion tokens
- Paper: https://arxiv.org/pdf/2302.13971.pdf

LLaMA

- Smaller Transformer-based models trained on more tokens
- Transformer modifications: (e.g., pre-normalization, SwiGLU activation function, rotary embeddings)
- Trained using only publicly available data
- Model hyper-parameters:

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Applications

- Content Generation
- Summarization
- Machine Translation
- Personalized Recommendations
- Sentiment Analysis
- Coding Assistance
- Healthcare Support

Limitations & Failure cases

- Lack of common sense and contextual understanding
  - Nonsensical responses when reasoning is needed (hallucination)
- Rely on patterns learned from training data
  - Propagation of biases in training data
  - If not monitored, can reinforce social inequalities
- Limited knowledge and outdated information
  - Lack of knowledge beyond training data
  - Lack of awareness of recent events

- Ethical, privacy and security issues
  - Need for comprehensive rules for trustworthy AI (EU law)

Environmental cost of LLMs

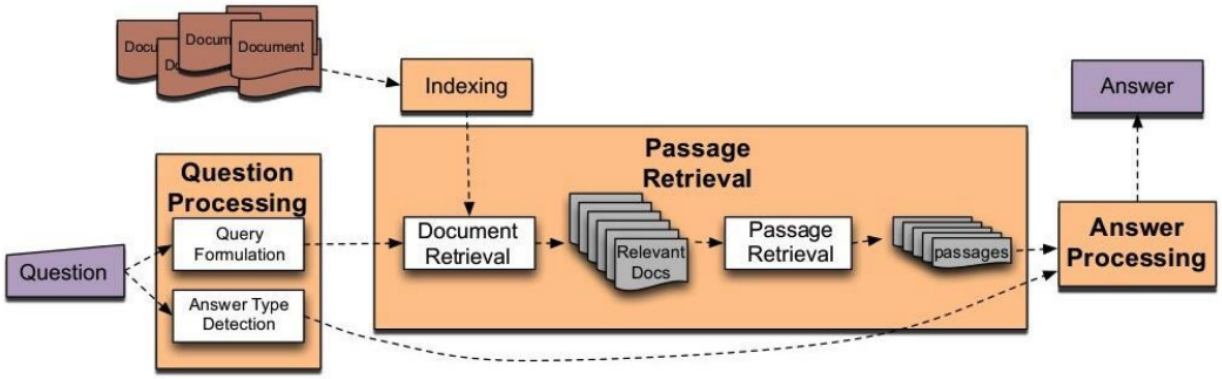
- Training and running LLMs require significant hardware infrastructures, including data center and cooling systems
- This leads to high energy consumption and carbon footprint, contributing to climate change
- Research efforts are underway to develop more energy-efficient training strategies
- Example: BLOOM required 3.5 months and consumed 1,082,990 compute hours. Training was conducted on 48 nodes, each having 8 NVIDIA A100 80GB GPUs

Question Answering

Question Answering (QA) is focused on building systems that automatically answer questions posed by humans in a natural language. It is a key domain in AI that combines understanding of natural language content with the retrieval of information and inferencing based on that information. QA systems can be broadly categorized into open-domain and closed-domain QA, each presenting unique challenges and requiring different approaches. The capability of QA systems spans from answering simple fact-based questions to complex reasoning and analytical tasks

Classical QA Pipeline

1. Question Processing: This initial step involves understanding and parsing the user's question. Techniques like classification and extraction of keywords, named entity recognition, and determining the expected answer type (e.g., person, date, location) are employed to better understand the user's intent.
2. Document Retrieval: Once the question is processed, the system searches through a collection of documents (often pre-indexed for efficiency) to find relevant documents. This step is crucial, especially in open-domain QA, where the breadth of potential knowledge sources is vast.
3. Passage Retrieval: After identifying relevant documents, the system extracts passages that are likely to contain the answer. This step narrows the search from entire documents to specific passages, making the answer extraction process more manageable.
4. Answer Processing: In the final step, the system works to pinpoint the exact answer within the selected passages. This might involve further natural language understanding techniques to interpret the context around potential answer candidates.

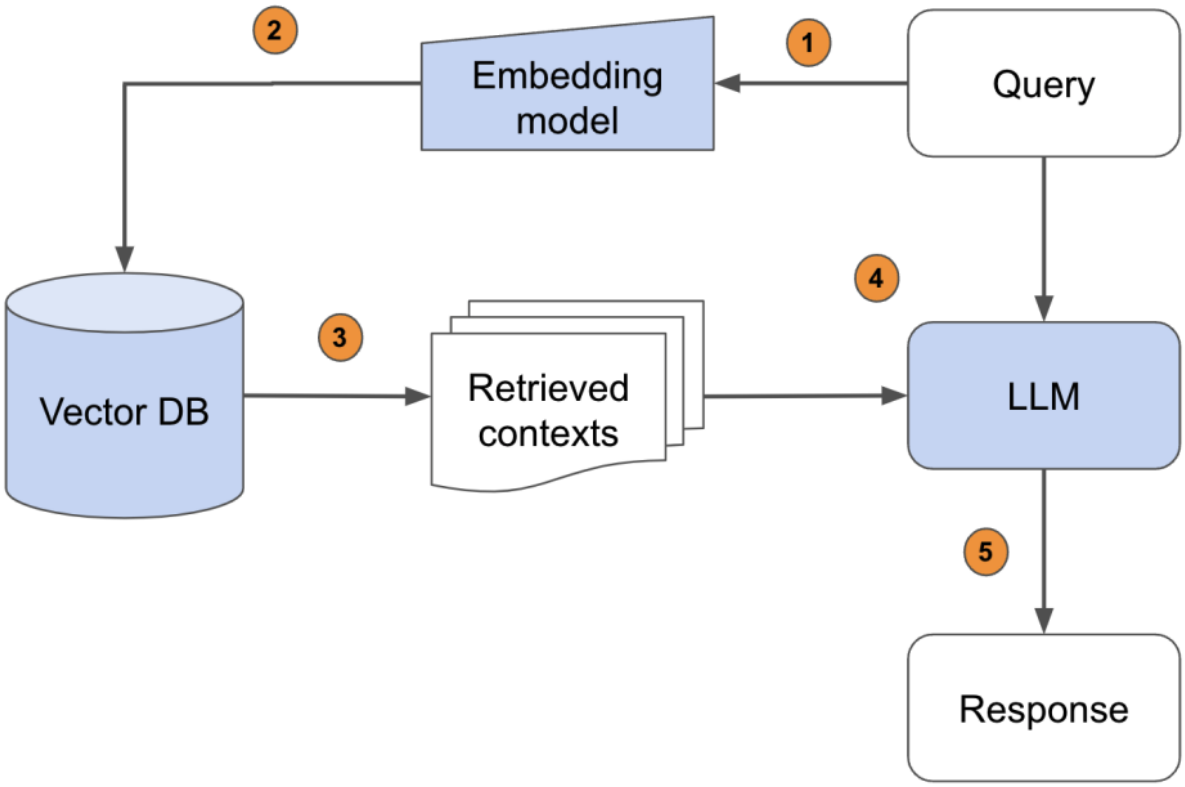


The classical QA pipeline relies heavily on keyword matching and retrieval techniques, often limiting its ability to understand context or perform deep semantic processing. However, it provides a structured approach to tackle QA tasks, especially in closed-domain settings where the scope of possible questions is limited and well-defined

RAG pipeline

The Retrieval Augmented Generation (RAG) pipeline represents a significant evolution in handling complex question answering (QA) tasks in NLP. It combines the generative power of Large Language Models (LLMs) with the information retrieval capabilities, making it particularly useful in scenarios where the LLM might not have all the necessary information in its parameters. Here's a detailed breakdown of the RAG pipeline:

1. Query Processing: The process begins when a query or question is posed. This query is first passed to an embedding model.
2. Embedding Model: The embedding model converts the query into a dense vector, also known as a query embedding. This embedding aims to capture the semantic essence of the query in a high-dimensional space.
3. Vector Database: The query embedding is then used to search a vector database (Vector DB). This database contains pre-computed embeddings of potential context passages or documents that are semantically indexed.
4. Retrieval of Contexts: The system retrieves the top-k most relevant documents or contexts based on the similarity between their embeddings and the query embedding. The relevance is typically measured using cosine similarity or other distance metrics in the embedding space.
5. Response Generation: The retrieved documents, along with the original query, are fed into an LLM. The LLM considers both the content of the query and the information contained in the retrieved documents to generate a coherent and contextually relevant response.



LlamaIndex: A RAG-based Approach

LlamaIndex is a specific implementation of the RAG framework designed to enhance the efficiency and effectiveness of information retrieval and answer generation:

- Efficient Information Retrieval: Instead of generating an answer directly from the LLM, LlamaIndex first retrieves relevant information from data sources. This step ensures that the context added to the question is up-to-date and directly relevant.
- Enriched Prompt Generation: The retrieved information is then appended to the original question to form an enriched prompt. This approach helps in grounding the LLM's response in specific data, improving both the accuracy and trustworthiness of the answer.

Advantages:

- No Need for Continuous Training: Unlike traditional fine-tuning approaches, LlamaIndex does not require the model to be continually trained on new data, reducing operational costs and complexity.

- Up-to-Date Information: Since data is retrieved in real-time or near-real-time, the responses generated are based on the most current information available, which is critical for many applications such as news-related queries or dynamic content areas.
- Transparency: LlamaIndex can provide insights into the documents or data it used to generate a response, allowing for better interpretability and trust in its outputs.

Close-Domain QA vs Open-Domain QA

- Closed-Domain QA: Closed-domain question answering refers to QA systems that focus on a specific domain, such as medicine, law, or finance. These systems are limited to a predefined scope and exploit domain-specific knowledge.
- Open-Domain QA:Open-domain question answering deals with questions on virtually any topic, similar to how search engines operate. It does not restrict the topic area and thus requires access to a broad range of general knowledge.

Single-Hop Reasoning vs Multi-Hop Reasoning

- Single-Hop Reasoning
  - Definition: In single-hop reasoning, the answer to a query can be found in a single document or a specific passage. The system needs to perform only one step of reasoning to arrive at an answer.
  - Example: If asked “Who is the President of the United States?” a single document listing current world leaders would suffice.
- Multi-Hop Reasoning
  - Definition: Multi-hop reasoning involves making multiple inferential steps to arrive at an answer. This type of QA system needs to gather and synthesize information from various sources or parts of a text.
  - Process:

        - Complex Queries: Questions that require synthesis of information or drawing connections across different documents.
        - Example: Answering a question like “What policies did the predecessors of the current U.S. President implement that affect current economic reforms?” would require connecting information about multiple individuals and policies over time.
    - Capabilities: These systems are adept at handling complex queries that require understanding context, drawing inferences, or resolving ambiguity by connecting disparate pieces of information.
- Comparison: Single-hop reasoning is straightforward and faster as it involves direct retrieval of information. In contrast, multi-hop reasoning is suited for complex questions that require deep understanding and connections across multiple data points, but it is computationally more intensive and challenging to implement effectively.

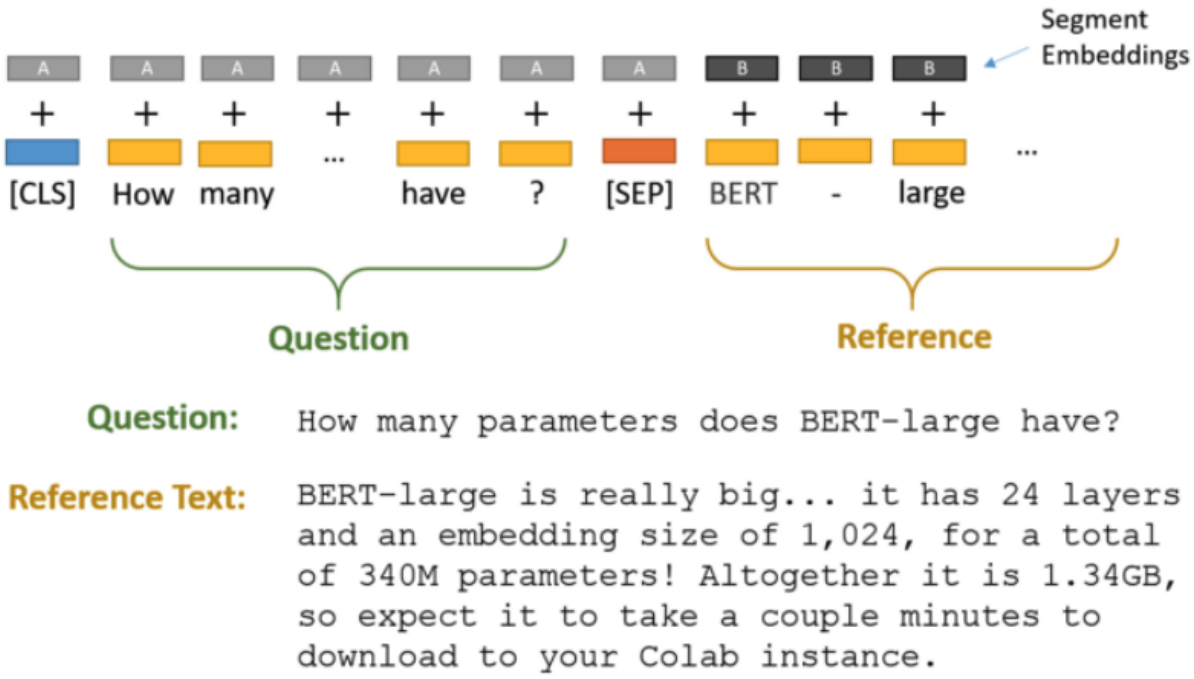
These distinctions are crucial in the design and application of QA systems, influencing everything from the architecture of the model to the type of data it needs to access and the algorithms used to parse and understand that data.

Evaluation Metrics

- Exact Match (EM)
    - Definition: The Exact Match metric measures the percentage of predictions that match the ground truth answers exactly.
    - Characteristics:
        - Binary Evaluation: It is a binary metric where a score of 1 is given if the predicted answer matches the ground truth exactly, and 0 otherwise. The comparison is typically case-insensitive and ignores punctuation and articles.
        - Strictness: This metric is strict as it requires the predicted answer to match the ground truth perfectly, without any deviations.
- F1-Score

- Definition: The F1-Score is calculated based on the precision and recall of the predicted answer compared to the ground truth.
- Components:
  - Precision (P): Defined as the ratio of the number of correct words in the predicted answer to the total number of words in the predicted answer. Precision measures the accuracy of the prediction, indicating how many of the words in the predicted answer were actually relevant.
  - Recall (R): Defined as the ratio of the number of correct words in the predicted answer to the total number of words in the ground truth answer. Recall assesses the completeness of the prediction, indicating how many of the relevant words were captured in the predicted answer.

Bert for QA



Inputs for BERT in QA

- Token Embeddings: Each word in the input sequence is converted into a token and passed through an embedding layer to get dense vector representations. These embeddings capture the semantic properties of the words.
- Special Tokens:
  - `[CLS]`: A special token added at the beginning of the question. It serves as a unique identifier for classification tasks.
  - `[SEP]`: A separator token used to demarcate different parts of the input, such as the end of a question and the beginning of a passage or between two different sentences.

Process of Using BERT for QA

1. Concatenation: The question and the passage are concatenated together into a single sequence with the `[CLS]` token at the start, the question, the `[SEP]` token, the passage text, and a final `[SEP]` token.
2. Segment Embeddings: These are used to differentiate between the question and the passage. Tokens from the question and the first `[SEP]` token might be marked as Segment A, and the passage as Segment B.
3. Positional Encodings: Since BERT uses a transformer architecture devoid of recurrence, positional encodings are added to give the model information about the position of tokens within the sequence

Fine-Tuning BERT for QA

- Start and End Token Classification: BERT is fine-tuned for QA tasks by training it to predict the start and end positions of the answer in the passage.
  - The output of the `[CLS]` token can be used for answerable/unanswerable question classification.
  - Each token's output embedding is fed into a start vector and an end vector, each with a softmax layer to predict the probability of that token being the start or end of the answer.

- Training: During training, BERT learns to adjust its weights so that the softmax layer outputs the highest probability at the correct start and end positions of the answer.
- Inference: At inference time, the model processes a question and a passage, predicts the start and end positions, and extracts the answer from the passage based on these positions

T5 for Generative QA

T5, or Text-to-Text Transfer Transformer, approaches every text processing challenge as a "text-to-text" problem, whether it's translation, summarization, or question answering (QA). This model, built on the Transformer architecture, redefines the task of question answering by framing it as generating text from text.

Key Features of T5 for QA

1. Input and Output:
    - The input is typically structured as a combination of the question and its context (a passage that potentially contains the answer).
    - The output is the answer generated by the model.
2. Pretraining:
    - Similar to BERT, T5 uses a denoising objective where it learns to reconstruct the original text from corrupted versions. However, it extends this by treating the reconstruction as a generation task, filling in missing text spans or reordering shuffled sentences.
3. Encoder-Decoder Structure:
    - T5 utilizes a full encoder-decoder setup. The encoder processes the input text, and the decoder generates the output text sequentially.
4. Training:
    - It employs a supervised learning approach using "teacher forcing," a technique often used in training sequence-to-sequence models. This

involves training the model to predict the next word in the sequence, given the previous correct words, as opposed to its own predictions.

5. Fine-tuning for QA:
    - During fine-tuning for QA, T5 adjusts to focus more on the task-specific demands of question answering. The model learns to attend to relevant parts of the input passage and generate accurate answers.
6. Versatility:
    - One of the strengths of T5 is its versatility. It can be fine-tuned on a variety of datasets and for different kinds of QA formats, whether they require direct answers, multiple choice answers, or even more complex reasoning.