DOMANDE NLP

Explain PageRank Algorithm

First of all, we have to enumerate and explain the main steps of the algorithm:

1) Links from a graph node to itself are ignored.
2) Multiple outgoing edges are treated as a single edge from one node to another.
3) Set the same initial value for all nodes (e.g., P(x) = 1/N)

In the end, we apply the page-rank formula to all vertices, at each step of the algorithm. Specifically, the page rank formula is: P(x) = (1 − λ)/ N + λsum from y to x (P(y)/ out(y)).
One of the most important applications of such an algorithm is information retrieval. In fact, it's possible to obtain an estimation of the importance of a node using such an algorithm. In addition to that, it could be used to develop search engines.
It's essential to specify that the PageRank Algorithm can not be used in all situations. In fact, graphs must not have sinks(trapping nodes). To solve this problem it is possible to add an outgoing edge from the sink vertex to every other node in the graph

Explain HITS algorithm

It is a link analysis algorithm that rates Web pages. The main idea behind the algorithm is that certain pages, known as hubs, are used as compilations of a broad catalog of information that led users direct to other authoritative pages. The algorithm uses two scores:

- Authority: estimates the value of the content of the page
- HUB: estimates the value of its links to other pages

The algorithm operates in 2 steps:

- Authority update: each node's authority score is set to be equal to the sum of the hub scores of each node that points to it. A node is given a high authority score by being linked from pages that are recognized as Hubs for information.
- HUB update: update each node's hub score to be equal to the sum of the authority scores of each node that it points to. That is, a

node is given a high hub score by linking to nodes that are considered to be authorities on the subject.

Named Entity Recognition and Luke

Named Entity Recognition Task(NER) consists of the recognition of a portion of texts referring to a known entity which could be a person, a city, etc. The main purposes of this kind of NLP application are to detect word n-grams that explicitly refer to particular entities and to classify them as one of the predefined categories of named entities. Sometimes, it’s possible to implement rules by hand to identify entities such as people's names. An example of such rules could be the presence of a capital letter as the initial letter and the absence of special chars. A negative aspect of this kind of approach is the fact that an entity could be presented in different forms(Dott. Zingarelli, Dott. Valerio Zingarelli, etc.). Moreover, it’s important to specify that NER tasks can be completed also in an ontology-based way and using ML and the main NN architecture which is used is called LUKE. Luke is an approach based on Transformers and it derives from a pre-trained BERT and it shares the same architecture. Input embeddings are token embeddings, positional embeddings, and entity type embeddings. They are summed together. LUKE considers special tokens to model the entities and uses an attention mechanism to connect entities and words. In the end, during the training phase, entities with annotated texts are masked and the model predicts the original masked entities.
Luke_loss= MML_loss+ CrossEntropyLoss
N.b. Cross Entropy is computed on entity recognition

Explain LATENT DIRICHLET ALLOCATION(and topic models)

In the context of Topic Models, we can distinguish Latent Dirichlet Allocation. More specifically, it’s a probabilistic topic model and the main concept behind it is the fact that documents are mixtures of multiple topics. Each word in a document is assumed to be generated by

sampling a topic from a document's specific distribution over topics and sampling a word from the distribution over words that characterize that topic. After that, for each document and each word in the corpus, a topic is chosen according to the found document-topic distributions. Inputs of the model are a vocabulary V, the number of topics k, and the parameters of the distributions (alpha and beta). In the end, words are extracted from the input vocabulary V by taking into account the term probabilities for each given topic in the document mixture.
Concerning Author Topic Model(ATM), it is a sort of generalized LDA for documents because it includes authorship information. In fact, each author is associated with the topics by a multinomial distribution, and each topic is associated with the words by a multinomial distribution too. Usually, a document has more than one single author, in these cases, the document is modeled as a distribution over topics that is a mixture of the distributions associated with the authors.

In the end, if we want to list the real applications of these kinds of algorithms, we can think of finding the most important author on a certain topic, or, on the opposite, we could wonder which are the most important topic covered by a given author.

Recommendation systems and RecoBERT

Given a set of users, the task is to give to each user a suggestion on what s/he could like. More specifically, if we call U the set of users, I the set of items that can be recommended, and R the set of ratings, the task of recommendation is to find a function F from U x I to R. It's really easy to imagine where such algorithms could be used. In fact, nowadays, our world is based on data, and most business models are trying to use them for gaining money. For example, the Netflix home screen is customized based on the client and the same is for Tripadvisor and all similar platforms. In fact, if I like Sci-fi films, Netflix will suggest to me more and more sci-fi movies. In such a context, we can distinguish between content-based and collaborative filtering systems. More specifically:

- Collaborative- filtering systems use the similarity between users to suggest content. This kind of suggestion presents some criticism:

they can suffer from the cold-start problem(new users' preferences are unknown) and they can make mistakes due to the popularity effect. In fact, if all the people similar to me watch Squid Game just because it's popular and I am not interested in this kind of tv series, the algorithm will suggest Squid Game to me and it would be a mistake.

- Content-based recommendation systems suggest contents that are similar to the ones the user selected before. In this case, the most important problem is represented by the "bubble". In fact, the algorithm suggests to the user contents similar to the ones user selected before and this could lead the user to not explore a different kind of content because basically s/he doesn't see it.

In the end, a possible approach to recsys uses ML and, in details, a modified version of BERT called RecoBERT. Its main feature is the possibility of computing similarity between objects without similarity labels. The inputs of the models are title-description pairs corresponding to positive ("real") and negative ("fake") samples, extracted from a given catalog. Then, the pairs are passed to the BERT backbone and transformed into feature vectors and, in the end, generated vectors are inserted into TDM and a cosine loss between them is optimized.

Aspect-based Summarization

Given a set of documents D covering different aspects A, produce a summary S of D covering each of the aspects in A. Each aspect-specific summary should contain all the key-information from the documents in D related to the corresponding aspect. For example: if I have financial documents concerning football and basketball, I want to generate one summary for football and another one for basketball. Moreover, if I have all the articles of a newspaper, I want to generate a summary for the ones concerning a topic I choose(e.g. COVID). The main difference with general-purpose summarization is that each summary is specific to an aspect. This allows for a more targeted and concise summary. Such an additional constraint is commonly not enforced by general-purpose summarizers. The summarization process is also more fine-grained, as it needs to account for the different aspects that might be present in the documents.

(inventata) Explain Latent Semantic Indexing (LSI)

Topic modeling can be studied by exploiting Latent Semantic Indexing. It relies on the LSA, Latent Semantic Analysis: the key idea behind this approach is to assume that words with closer meaning will probably occur in similar pieces of texts. Another mathematical point that's exploited by LSA is SVD, Singular Value Decomposition: we want to reduce the entries' matrix while preserving similarities (computed with the cosine similarity), hence reducing computational complexity and redundant information. By doing so, we want to first investigate the algebraic procedure behind SVD.
Let's call A our studied matrix. There exists a mathematical relationship in which:

$$A = USV'$$

where:

- V column vectors = right singular vectors
- U column vectors = left singular vectors
- S diagonal values = singular values (ranked from high to low)

We can interpret this definition also as a linear combination of several min(m,n)-rank-1 matrices. Therefore, one may investigate just the top k values instead of the overall min(m,n): this is what's called Low Rank approximation, i.e. how to approximate A by a rank-k matrix. Choosing k is an heuristic process, but in general it follows the "elbow rule" similar to the PCA top-k components selection.
Eventually, the topic detection will indeed result in choosing those first k instances from the new A, written as the following:

$$A_k = U_k S_k V'_k$$

And in particular, by choosing that k for which we obtain:

- U first k columns = term relevance to first k concepts
- V first k columns (V' first k rows) = doc relevance to k concepts

To a more real-world usage, one can think to use this kind of topic modelling technique to discern the most relevant topics regarding a scientific paper, or maybe a Twitter post, etc.

Explain how Word2Vec works (+ training example + drawbacks + difference with FastText)

Word2Vec is one of the well established Embedding models for NLP. The basic idea is to have as inputs large doc corpora, and generate word vector space as output by means of a FeedForward neural network language model (FFNN LM). In particular, we aim to compute for each word its pairwise vector similarities, then adjusting the word vector to maximize the computed conditional probability $p(w|w') \; \forall w'$.
There are two training approaches:

- CBOW, to investigate syntactical relationships. Here we train the NN to predict a word from the context.
- Skip-gram, to investigate semantical relationships. Here instead we train the NN to predict context starting from a w.

One may be interested in explaining the former process. CBOW has the fastest training procedure among the two approaches: it ignores repetitions, it doesn't use the non-linear hidden layer, and the projection exploited by the FFNN backbone is shared among the vectors, averaging them. Another important property about CBOW is that words' sequence order has no influence on the final result, that's why this model also exploits "future" words. Eventually, its final goal is to maximize the average log probability, that is conditioned to the context window's words surrounding the imputed word *w,* no matter their order:

$$max \; log(P(w_c|w_{c-m}, \dots , w_{c+m}))$$

However, Word2Vec suffers from several drawbacks, for instance it's unable to model sequences (countered by the usage of a LSTM module), or it's unable to consider OOV (out-of-vocabulary) words.
The latter issue is well resolved by FastText, a Facebook-based Embedding model. Here the key idea is to work by investigating the sub-words that compose the words of interest, i.e. their n-grams. In fact, a word w can be seen as a compound of several n-grams, with n fixed, and including the word itself in the sum/count:

$$s(w, c) = \sum z'_g v_c$$

where *w*=word, *c*=context, and z_g=vector of the n-gram "*g*".

This solution can infer a faster and efficient training for the model, thanks also to its OOV words inclusive method, that investigate semantic or syntactical relationships that may be avoided with Word2Vec (e.g. "kingdom" has inside the word "king", that could be an OOV word that may not be included in the studied sentence).

How do Transformers work? What's the "Attention Mechanism"?

Transformers are a more efficient and faster alternative to Seq2Seq models for contextualization in NLP. Their goal is to maximize parallelization and minimize recurrence: as an example, the complexity for Transformers is O($n^2 d$) thanks to self-attention, against the O($n^2 d^2$) given by recurrence. The key idea is avoiding RNNs, while letting the architecture relying on a combination of encoders and decoders, stacked together, and the attention mechanism. Sticking to the original paper, each encoder/decoder module consists in 6 layer, moreover:

- for each encoding layer we have 2 sublayers, 1 multihead attention, 1 FFNN, adding and normalization operations among the skeleton, surrounded by residual connection for the gradient flow;
- for each decoding layer we have 2 multiheads attention (the former is masked and it's used for the output embeddings, the latter is non-masked and aims to communicate with the encoder, for hidden vectors), 1 FFNN, and residual connections.

Eventually, positional encoding to take into account the relative word position is used.

The idea behind Transformers was born to investigate the potential power behind Attention Mechanism itself. This approach mimics the retrieval of a value *v* from query *q* with a key *k*: we study the similarity between q and k_i , we normalize it with softmax (p_i), and then we compute the weighted sum between v_i and p_i. Different similarities can be investigated for different purposes. The Attention gives us more information about the context that surrounds words, that's why it's becoming more and more popular in NLP.

Sentence encoding and InferSent

Sentence encoding is the sentence version of word encoding. What we want to obtain is an embedding(i.e. a vector) representing a phrase. InferSent is a supervised model (it uses SNLI dataset) whose purpouse is to learn universal representations of sentences. It's based on bilateral LSTMs. In the context of InferSent, transfer learning of a supervised model used fort textual Entailment Recognition is used. In such a scenario, we can distinguish Positive Textual Entailment("Giving money to a poor man is a good thing"), Negative Textual Entailment("Giving money to a poor man is not a good thing") or NON-TE("Giving money to a poor man will make you better"). Turning back to architecture, h(t) is computed basing on the outputs of the forward and backward LSTMs models and, in the end, hidden attention mechanism is added.

Sent2Vec

Sent2Vec is an unsupervised model for sentence embedding. It's based on compositional n-grams features and extends FastText and CBOW Word2Vec. In fact it composes sentence embeddings using word vectors and n-grams vectors. As a significant difference with respect to Word2Vec, we can notice that the dynamics context window of word2vec model is not used anymore. In fact, the entire sentence is considered as the context window, instead of sampling the context window size for each subsampled word uniformly between 1 and the length of the current sentence. Concerning the model, well, it is inspired by simple matrix factor models where we want to optimize

$\sum f_s(UVd_s)$, where U is the target word vector and V is the learnt

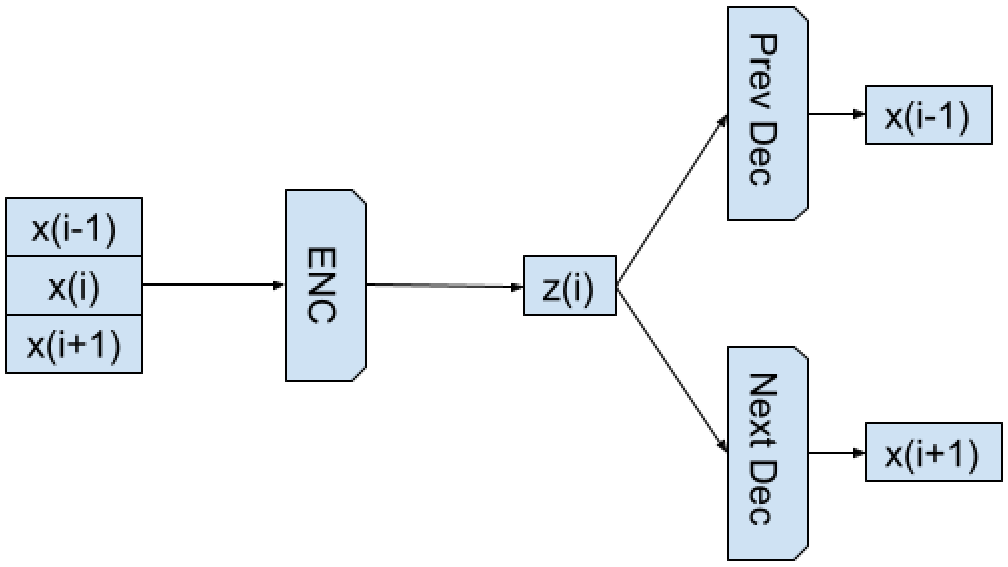
vocabulary word vector.

Skip-thought Vectors

It's an unsupervised model for sentence embedding. It is based on an encoder-decoder architechture and it recontructs surrounding sentences of an encoded passage. Concerning architecture:

- ENCODER: it takes sentence x(i) and generates a representation z(i). It is composed by RNN with GRU activation

- PREV DECODER: takes z(i) and tries to generate a prevision of x(i-1). It is composed by RNN with conditional GRU activation
- NEXT DECODER: takes z(i) and generates a prevision of x(i+1)



Describe ELMo and its training procedure

Embedding from Language Model(ELMo) is an embedding model which main feature is the replacement of static embeddings with context embeddings depending from the context. In fact, the main idea behind the model is the fact that there is not a single embedding for a word but it depends on the context(i.e. sentence) the word is in. In fact, in this kind of approach, each token embedding is a function of the entire input sentence and it is computed by a deep multi-layer bidirectional language model. In the end, the model returns, for each token, a linear combination of the tokens across the different layers of the NN. Let's now have a look at the training procedure of such a model.

- Step 1: train forward and backward LTMs
- Step 1.2: ELMo concatenates the results of forward and backward models in order to obtain a single embedding(this result is the output of epoch t and it will be the input of the next epoch t+1)
- Step 2: embeddings are now adapted to the task we have to fulfill.

- Step 3: weight concatenation is multiplied with a weight based on the task which needs to be solved
- Step 4: Elmo represents the tokens as linear combinations of corresponding hidden layers.

IMPORTANT: Have a look at the slides with formulas!!
Several tasks can be fulfilled using ELMO. In fact, it can be used for question answering, sentiment analysis, NER, etc.

Describe BERT

Bidirectional Encoder Representation for Transformers(BERT) is an encoding model which uses transformers. BERT is a stack made of encoders and each encoder is made of Multi-Head Attention+ FFNN.
Key ideas behind the model are the following

- Use Transformers for sentence encoding
- Bidirectional models

Its main features managed to make it the state of the art. In addition to that, we can state:

- It is very fast to train
- It is very easy to tune for specific tasks

Let's now have a look at the pre-training strategies of such a model. We can distinguish basically two strategies:

- Masked LM: 15% of the words are masked
- Next Sentence Prediction: predict the semantic relationships between sentences. The input is a pool of phrases and the model should predict if the second sentence follows the first in the original document

Notice that BERT cannot process documents with more than 512 words.
In the end, a combination of the strategies explained before is used to form a single unique loss and the fine-tuning of the model for specific tasks is performed.

Describe GPT(1,2 and 3)

GPT is an embedding model and uses a stack of decoders(on the contrary, BERT is a stack of encoders). Each decoder block is formed by a masked-self attention mechanism and a FF Neural Network. The main

idea is: “ What happens if we want to model only the following word?”. Well, the answer is that we don’t care about the encoders anymore. The key point of GPT 2 is the usage of transformers for decoding sentences and the training is performed using an unsupervised environment. In addition to that, GPT uses a masked self-attention mechanism and byte pair encoding which consists of the decoding of the most frequent sub-words.
According to the dimension of the model, we can distinguish GPT Small, Medium,etc.
Its main applications are Summarization and Machine Translation.

Explain differences between self and cross attention and multihead and masked multihead attention.

Transformers rely on the so-called “attention mechanism”, i.e. a smart way to avoid recurrence and deal with word processing. In particular, there exist two types of attention:

- Self-attention, when the attention mechanism is performed on queries, keys and values generated from the same embedding
- Cross-attention, When attention is performed on queries generated from one embedding and keys and values generated from another embeddings

Let’s consider now the whole structure of Transformers. Starting from the Encoder, we can notice the presence of the “Multi-head” module for self-attention. Calling X the input word, the key steps are:

1. Embedding each X, adding positional encoding
2. Split into N heads, computing the multiplication between X and the weight matrices
3. We end up with q,k,v matrices, then we use them to calculate attention (Z)
4. We concatenate Z to weight matrix W to generate the output

“Masked-multi-head” instead is positioned as a first module in the Decoder: let’s suppose we want to compute $P(x_i|x_{j<i};\ \theta)$, but we still feed for efficiency concerns the whole sequence $x_{i=1:n}$. The way to correctly train the model without looking at $j \geq i$ and computing the corresponding probability is called “Masking”, and that’s why it is positioned at the beginning of the Decoder.

Rouge, different kinds of rouges, and summarization metrics

Recall-Oriented Understudy for Gisting Evaluation(Rouge) is an intrinsic metric based on syntax and is useful because it allows model creators to compare an automatically generated summary with gold summaries(typically humanly generated). Basically, this kind of metrics measures how much units overlaps in gold summaries and generated ones. We can distinguish:

- Rouge- N: it computes the n-grams co-occurrence in the gold summary and generated one
- Rouge-L: it uses the longest common sequence. To be clearer, if we consider 2 sequences X and Y, the longest common sequence is the common sub-sequence with maximum length. The intuition behind this rouge is the fact that the longer the common sequence the more similar the texts will be.
- BERT score
- BLEU score
- MRR

What do we mean by “Shallow Parsing”?

Shallow parsing is a textual analysis that is not deep. Traditional algorithms and methods are really time-demanding and that’s why shallow parsing has been introduced. It sacrifices the depth of analysis in favor of speed and robustness. In fact, instead of providing a complete analysis (a parse) of a whole sentence, shallow parsers produce only parts that are easy and unambiguous. Typically, small and simple noun and verb phrases are generated, whereas more complex clauses are not formed. Similarly, most prominent dependencies might be formed, but unclear and ambiguous ones are left unresolved. For the purposes of information extraction, shallow parsing is usually sufficient and therefore preferable to full analysis because of its far greater speed and robustness.

ElasticSearch and Information Retrieval

Information Retrieval consists in finding material, usually texts, that is able to answer a question or an information need. Of course, this kind of research is meant to be executed in a big pool of data stored on computers. But what does "search" mean?
Well, we have to take a string as input and confront it with the whole set of documents we have, in order to check if the word of the query is present in each document. Then, we need to sort the documents according to the relevance of the results and, in the end, we should display the sorted list of documents to the user.
In such a context, we need search engines that do this kind of research. One of these engines is ElasticSearch.
ElasticSearch is a real-time distributed search and analytics engine that is really scalable and efficient. Wikipedia, StackOverflow, and Git use ElasticSearch, for example. In ElasticSearch data are stored in named entries belonging to different data types. If we want to make a comparison between SQL and ES, we can state that the attributes of SQL correspond to the "fields" of ES. The rows of the tables in SQL are the "documents" in ES. An SQL table is an "Index" in ES and the entire SQL database corresponds to a "cluster" in ES. The interpretation of the data in each field is called "Mapping".
In addition to that it is very relevant the difference between "FILTER" and "QUERY" :

- Filter is used for fields containing exact values (Gender, age, name,..)
- Queries are used for searching full text in the documents. In fact, there is not a unique correct solution for a query.

It's important to specify that each document has a field named "version" which is used for tracking the current version of the document. Of course, ElasticSearch can perform updating and deleting.

How is the scoring evaluation for ElasticSearch?

ElasticSearch provides a smart way to match queries in its search process: if we are investigating a full-text input, we have to use the right analyzer, then perform searching and, as a final step, compute the

relevance scoring (between 0 and 1), while for exact-value-based search the relevance score will trivially be equal to 1.
As we can see, Relevance scoring has an important role in ES. In particular, it will:

1. select docs matching the searched query
2. evaluate the importance through ad-hoc weight metrics
3. evaluate the similarity between doc and query

In particular, TF-IDF can be a suitable choice for single-term weight, while the query/doc should be represented through a Vector space model. The latter representation allows us to properly investigate the similarity between the result and the query thanks to the adopted cosine similarity.

What is an “Intent”? What are “Intent detection” and “Intent classification”? Which are the most used ChatBots?

The intent is the intention a user manifests. According to Cambridge dictionary’s definition, it is: «the fact that you want and plan to do something». In NLP intent detection is a very important part of any AI Chatbots. In fact, when the user asks for something, ChatBot should be able to understand the request and fulfill it properly. Usually, it’s possible to generate a list of intents and use them as labels to train an AI classification model; in such a way the chatbot will just classify the user’s queries between the ones it has been trained on. Of course, the ending part of the model is just a classifier, but the problem is the previous part: feature creation. How do we create features for the classifier? Embeddings generated via Deep Learning Models(such as BERT) could be a suitable choice.
Concerning Chatbots, the most important architectures are Amazon’s Azure, IBM’s Watson, Google’s Dialog Flow, and RASA. The last one is a framework for Natural Language Understanding based on LSTM Neural Networks and it’s divided into Core, NLU, and NLG. The first one is the decision-maker of the chatbot and the second one is a model for extracting structured data from raw text. In the end, the NLG unit takes the outputs of the Core and transforms them into something readable to

the user. Recently more and more companies are using chatbots for customer assistance and recruiting.

Sentiment analysis: task, main applications, and main methods

Sentiment Analysis, from now on SA, is a sub-group of text categorization/classification. More specifically, in this task, our purpose is to detect if a text is positive or negative about something. In this kind of scenario, we can rely on both ML techniques and classical NLP methods. It's important to underline that, in general, sentiment analysis works better if the document we are analyzing is written by one person. Concerning applications, it can be easily used to "read" reviews or social media posts in order to understand if users like or not something.
In order to obtain a sentiment analyzer we should follow the correct pipeline:

1) Preprocessing: tokenization, stopwords removal(IMPORTANT do NOT remove "not", "never" etc.), text-cleaning (regex)
2) Feature extraction/selection: pos-tagging, entity recognition, dependency tree generation in order to understand if adjectives refer to an entity or not
3) Apply a Machine Learning classifier to classify the sentences/documents.

In addition to, points 2 and 3 could be replaced by a Deep Learning model which generates the features by itself.

QA: task, main applications, and main methods

Question and Answering is a task based on trying to answer a question that is given in input. More specifically, the input of the model consists of the question and the text where it should find the answer.
Some application examples are search engines, chatbots, and virtual assistants(Alexa, Siri, Cortana,...).
The main datasets used for this kind of task are

- SQuAD: (question, answer) on a lot of documents that are divided into paragraphs.
- NewsQA: questions and answers generated on CNN articles.

- HotpotQA: the questions require finding reasoning on multiple documents. It introduces “Supporting facts”
- TriviaQA: documents are not guaranteed to contain all the facts needed to answer the question.

A simple pipeline for this kind of task could be:

1) Question analysis: understanding the kind of question and which is the entity.
2) Documents retrieval: take the most important documents and their most important passages and, according to the task, documents and paragraphs could be ranked
3) Answer generation: using paragraphs selected during the previous step, an answer is generated.

In addition to that, Deep Learning models can be used to fulfill such a task. In fact, models based on LSTMs (Bidaf) and BERT have been used. BERT is trained using normally and fine-tuned with (question, separator,paragraph_with_answer) as an input. The purpose is to find the start and the end of the answer.
Concerning BiDAF, it’s composed by:

- Contextual Embedding Layer: combinations via a bidirectional LSTM of Char Embedding Layer and Word Embedding Layer
- Attention Flow Layer :
    - Query to context: indicates which query words are most relevant to each context word
    - Context to query: indicates which context words have the closest similarity to one of the query words
- Modeling layer: two bidirectional LSTM
- Output Layer provides an answer to the query:
    - FNN to predict the start index of the answer
    - Bi-directional LSTM to predict the end of the answer

The task can be open-domain or closed-domain. In the former case, questions and answers are referred to general knowledge while, in the latter one, they are referred to specific domains and thus exploit domain-specific knowledge. If the task is closed-domain, the model should be able to state that it is not able to answer the question.

What’s Tfidf? How does it work?

TF-IDF is one of the main approach for what concerns occurrence-based text representations. It's a suitable choice for homogeneous documents, yet it provides a solution to data sparsity.
There are two mathematical ingredients:

- Term Frequency tf_{ij}. It is the observed frequency of the i-th word in the j-th doc, it also gives positive relative importance to the words, focusing on the local pattern. Mathematically speaking we have:

  $$tf_{ij} = \frac{n_{ij}}{d_j}$$

  where n_{ij} is the number of times the word i appears in the doc j

- Inverse Document Frequency IDF. We start from Document Frequency: we are in the "doc perspective", it is the number of docs in which the word occurs. It negatively affects the relative word importance: word occurrences spread all over the docs. To get the IDF we have to make some algebraic manipulations:

  $$idf_i = \log_{10} \frac{|D|}{\{d: i \in d\}}$$

  where |D| is the absolute number of total doc collections. The denominator trivially explains the domain support, in which we find all the docs in which the word i appears at least once.

What's BM25?

BM25 is one of the available occurence-based text representations. It's main idea is to limit the bias due to the usage of TF-IDF by infering statistics on the whole corpus, and to overcome the TF-IDF issues related to unbounded values.
To do so, its main formula needs 3 mathematical ingredients:

- *avgdl* = the avg document length
- *k* = a saturation coefficient referred to TF
- *b* = the penalty for the document's exceeding-threshold length.

The formula is the following:

$$BM25_{ij} = IDF_i \cdot \frac{TF_{ij}(k+1)}{TF_{ij}+k(1-b+\frac{bd_j}{avgdl})}$$

From this idea, we can exploit and leverage different representations by changing the free parameters in the general formula. Here there are some examples of this alternative scoring functions:

- DOUBLE NORM, i.e. setting an unique free parameter $k \in [0, 1]$
- FREQUENCY MAX, i.e. bounding the variation range of the frequency values
- LOG NORM, i.e. scaling the skewed weight distributions

What’s the distributional hypothesis?

Distributional hypothesis is based on the idea that words that occur in similar contexts tend to have similar meanings.
It is possible to capture the underlying context of a word by looking at the surrounding words. The goal is to find semantically related words that answer properly to a target. In doing so, the key idea is to work with a “Context window”, i.e. a fixed-size sliding window that allows the model to catch the surrounding target context properly.
The distributional hypothesis is the base for a lot of well-established embedding models, such as Word2Vec.

Explain the RASA framework, describing each module and analysing its structure.

RASA framework is one of the main architectures on which the chatbot environment relies on. It handles the input from user in a structured way. We can find several “blocks” that cooperate in this structure:

- RASA NLU = it manipulates the input raw text to get a more structured data to be elaborated by the environment. It works independently from the Core.
- RASA CORE = it decides the next action for the chatbot exploiting an ML-based dialogue. In particular, it uses an LSTM approach to guess the next action.
- RASA NLG = it provides neural generative models for the answers, and it can be seen as the “mouth” of the model. It could be either template-driven, consisting in static answers and defined context, or DL-based, more dynamic and “human”-like

Zooming in, we can also find single elements that guarantee the right functioning process inside the RASA framework:

- Stories = user-bot interactions, e.g. intents or actions
- Domains = they are the context in which the framework opertes. It consists in intents, entities, slots, responses, actions..
- Tracker = the bot's memory storage. It both links the various interactions and provide info for the next possible action
- Policies = they decide which action to choose, and it relies on a mix between ML-based and Rule-based approach.
- Actions = they are predicted after each message, describing the bot interaction with the environment. They are assistant-based.

(NOTE: maybe a little draw of the architecture)

Explain the BEAM encoding, where we use it, why we use it and how it works

We use it in the decoding stages of a seq2seq model. In the decoding stage of a seq2seq model, we want to maximize the probability that output is the right one given all the outputs decoded in the previous stage and the encoder outputs.
BEAM encoding comes to help us because it makes a trade-off between the computational complexity (exhaustive encoding is computationally unfeasible) and the quality of the result (Greedy encoding can't allow us to undo previous decisions on what is the token maximizing the probability). ATT! It does not guarantee finding the optimal solution.
BEAM tracks what are the top k tokens, the ones having the highest log probability, (typically 5<=k<=10) and for each of them computes its k top follower tokens. Now we have k^2 combinations (hypothesis). Between those k^2 hypothesis, we keep only the top k and for each of them we compute its k follower tokens and so on, until the model produces an <EOS> in one of the hypotheses, or until we have n hypothesis (each one with its <EOS>), or until we reach a predefined time step T.
We make a comparison between different length hypotheses normalizing the score with the hypothesis length because longer sentences will have

lower scores due to the intrinsic nature of the score.

$$\text{score}(y_1,\dots,y_t) = \log P_{\text{LM}}(y_1,\dots,y_t|x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\dots,y_{i-1},x)$$

$$\frac{1}{t}\sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\dots,y_{i-1},x)$$

What are the steps to compute BLEU score?

- Compute the modified n-gram precisions p_n using n-grams of length n, for n=1,...,N

$$p_n = \frac{\sum_{C\in\{Candidates\}} \sum_{n\text{-}gram\in C} Count_{clip}(n\text{-}gram)}{\sum_{C'\in\{Candidates\}} \sum_{n\text{-}gram'\in C'} Count(n\text{-}gram')}$$

- Compute the geometric average of p_n, using weights w_n summing to one

$$\exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

- Compute the Brevity Penalty factor BP, to penalize candidate translations having length c smaller than the effective reference corpus length r

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \le r \end{cases}$$

- Compute BLEU score as the product

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

- Rank the BLEU scores using the log domain

$$\log \text{BLEU} = \min(1-\frac{r}{c},0) + \sum_{n=1}^{N} w_n \log p_n.$$

Describe BART architecture, provide some examples of applications and explain the attention mechanism behind it.

The BART architecture is a type of Transformer, and it can be seen as a denoising autoencoder for pretraining Seq2Seq. So, in some sense, it can be seen as a seq2seq models since it is composed by encoders and decoders. It consists of a mixture from BERT and GPT:

- BERT encoder with random masks and a bidirectional approach
- GPT decoder with tokens that are predicted auto-regressively, forward.

It's used for many NLP applications, and the structure is quite the same for each of them (summarization, sequence classification..), except for the Machine Translation task: here we find a Randomly Initialized Encoder that can use a disjoint vocabulary, and "sees" the enc-dec structure as a full decoder. It also replace BART's encoder embeddings.
Since BART is a Transformer, it also relies on the Attention mechanism: for each decoding step (t), attention uses direct connection to the encoder to retrieve its distribution, by chaining a multiplication between the encoder hidden states ($h_i,\ i = 1{:}N$) and the decoder one (s_t), getting the probability distribution of the aforementioned vector with softmax (α^t), and retrieving the output to obtain the distribution for each token.
As an example, think about the sentence "*Je m'appelle X": in order* to translate it, at the first step t=1 the attention distribution will be higher on the word "*Je"* even though we have also scanned the other ones.
(NOTE: maybe attach a little draw about the structure/step with distributions)

What are Seq2Seq models? Is BERT a Seq2Seq?

Seq2Seq models are models based on encoders and decoders. Both encoders and decoders are made of RNNs and each RNN takes into account the current sample and a representation of the previous inputs. Then, sequential information are stored into an hidden state and used at next istance. These kinds of models have a heavy drawback: they are really computationally demanding because of recurrences and no parallel training is allowed. In order to solve this problem, transformers and attention mechanism have been introduced. Seq2Seq models are used in different NLP tasks: Summarization, Machine Translation, Parsing, Code Generation.

Concerning BERT, we can state it is not an example of Seq2Seq because it uses only encoders and not both encoders and decoders.

Explain Textrank and provide the main steps

(Same general explanation also for Lexrank)Textrank is an unsupervised extractive graph approach for Summarization tasks. It's based on graph centrality measure, and being a graph model is highly effective, easy to use, but with high complexity and it's not incremental.
The main steps of this approach are:

1) identify text units that best define the task at hand, and add them as vertices
2) identify relations that connect such text units and let them be the edges (undirected, weighted, or unweighted)
3) iterate the graph-based ranking algorithm until convergence
4) sort the vertices based on their final score (use their values for ranking/selection)

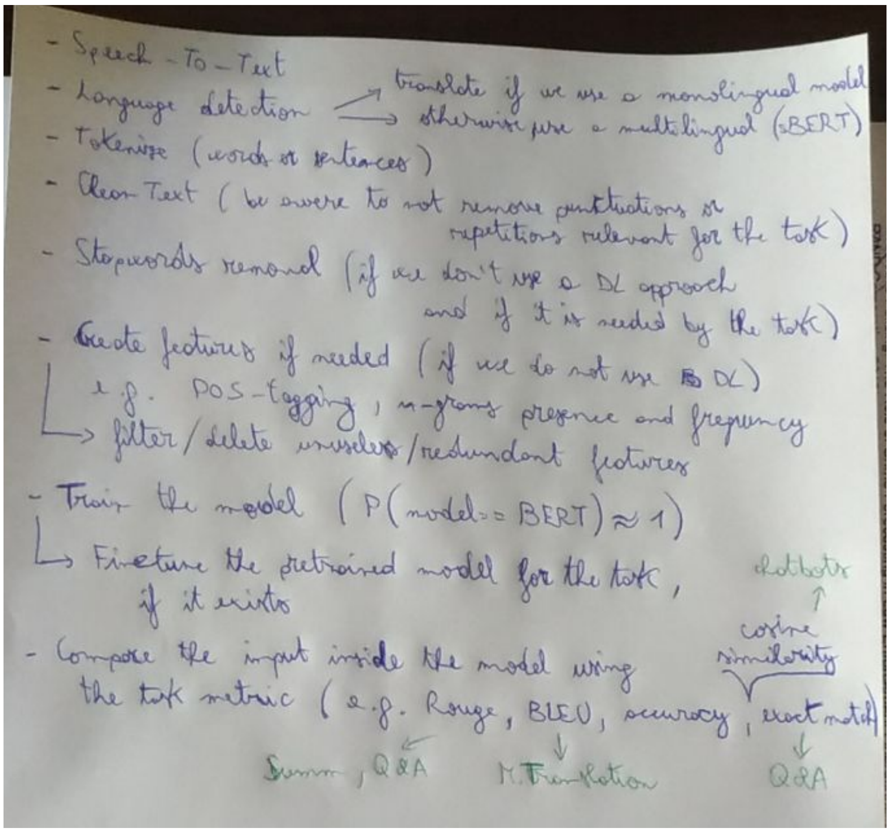
Explain Lexrank and provide the main steps

(Here we put the same general explanation for Textrank, previous quest.)
The main steps of this approach are:

1) compute tf-idf scores for each word
2) compute the cosine similarity in the tf-idf vectors
3) prune low-value edges, below the given threshold
4) compute the degree centrality score for each sentence
5) apply PageRank to sort the sentences.
6) give the user back the top k sorted sentences

THIS SHEET IS ONLY FOR PIPELINES(https://huggingface.co/docs/transformers/main_classes/pipelines could be useful)

TRY TO BUILD A GENERAL TASK PIPELINE



- Speech-To-Text
- Language detection → translate if we use a monolingual model
  → otherwise use a multilingual (sBERT)
- Tokenize (words or sentences)
- Clean Text (be aware to not remove punctuations or repetitions relevant for the task)
- Stopwords removal (if we don't use a DL approach and if it is needed by the task)
- Create features if needed (if we do not use DL)
  - e.g. POS-tagging, n-grams presence and frequency
  - → filter/delete useless/redundant features
- Train the model ($P(\text{model} == \text{BERT}) \approx 1$)
  - → Finetune the pretrained model for the task, if it exists
- Compare the input inside the model using the task metric (e.g. Rouge, BLEU, accuracy, exact match)

NER pipeline

1) Tokenization
2) Human annotation (B-prefix, I-prefix, O-prefix)
3) Feature extraction (POS tags, symbols, word shapes, word substrings)
4) Training dataset (word-feat1-feat2-...-class)
5) Model training. NER is a classification task
6) Sequence labeling using sequential models (classify each token independently + sliding window for neighbors info)

SVD for topic modelling pipeline + find relevance (doc-level, word-level)

1) Reduce the SVD (U*S*V’)
2) Apply heuristic approach: select k (e.g. = 2) + filter (e.g. smallest singular value must be above the half of the highest one)
3)
    a) **(doc level)** The first k=2 rows in V’ = relevance of the doc to each of the selected topics ($i = [0, 1]$)
        i) d1 = $[S_{00} * V'_{00}, S_{11} * V'_{10}]$
        ii) d2 = $[S_{00} * V'_{01}, S_{11} * V'_{11}]$
        iii) d3 = $[S_{00} * V'_{02}, S_{11} * V'_{12}]$
        iv) …
    b) **(word level)** The first k=2 columns in U = relevance of the word to each of the selected topics ($j = [0, 1]$)
        i) w1 = $[U_{00} * S_{00}, U_{01} * S_{11}]$
        ii) w2 = $[U_{10} * S_{00}, U_{11} * S_{11}]$
        iii) …

QA bot with the most FAQ

1) input: QA answers, user query
2) BERT to encode QA into semantically-awared fixed sized vectors (pre-trained BERT)
3) Compute embeddings for the user query
4) Compute cosine similarity between query and each question
5) Answer to the user with the most similar question

ML Classifier for topic detection

1) Parse the file and perform sectioning (e.g. if we use Latex files, sectioning will exploit Latex tags)
2) (if required) Detect language + translate if monolingual or use proper models for multilingual
3) Filter out redundant tags/special chars
4) Tokenize raw title text + splitting it into words
5) Scan the language stopword

6) lemmatization/stemming
7) compute BM25 matrix / look for each word in the pretrained W2V / ...
8) build a co-occurrence dataframe (rows = text / cols = BM25 for k-th word or W2V value… + target)
9) Train/test split
10) Fit SVM + test

Recommend item-to-item (e.g. books)

1) input = book catalog (dict) C, previously selected book b_u
2) get embeddings from BERT pretrained model
3) infer similarity function $F: C \times C \rightarrow R$ (e.g. cosine)
4) find and provide as the output the top-K most similar books to b_u

Questions asked - previous exams

1) Open questions:
    a) Pagerank algo (main steps + examples + constraints to be applied to a given graph)
    b) Named Entity Recognition task (main goals + examples of rules + high-level description of LUKE)
    c) LDA (main steps + atm and comparison with LDA + examples of atm)
    d) Recommendation task (main goals + business examples + content-based vs collaborative filtering)
    e) Aspect-based-summarization (task + comparison with general-purpose + examples)
    f) Attention mechanism (intuition + main formula + complexity)
    g) Seq2Seq(idea, mechanism, is BERT a Seq2Seq, examples of usage)
    h) Multilingual and cross-lingual summarization
    i) **(Simulation)** Distributional hypothesis (initial hyp + examples + models)
    j) **(Simulation)** Statistical Machine Translation (general problem + pipeline + other solutions comparison)
    k) **(Simulation)** LSARank paper (summarization problem + reasons why it is useful for summarization + procedure to extract most relevant sentence)
    l) **(Simulation)** Stopword (concept + examples + approaches)
    m) **(Simulation)** Word2Vec (main goals + one training procedure + main drawbacks + comparison with FastText)

2) Closed questions:

a) n-gram definition? *A contiguous sequence of n characters that are part of a word*

b) normalization technique in NLP? *Lemmatization*

c) metrics for summarization? *BLEU*

d) cold-start problem? *recommending item to new users*

e) true for BERT pre-training phase? *Self-supervised training with masked language modeling and next sentence prediction is used in the training phase*

f) negative sampling? *to select negative samples during W2V training*

g) OHE? *textual unit represented by a sparse vector of boolean elements*

h) HITS algorithm, during HUB update step? *for each node, hub score is the sum of the authority scores of each node that it points to*

i) supervised domain adaptation? *exploiting in-domain data to specialize models trained on open-domain data*

j) false for word embedding models? *w2v relies on a Deep Learning model (it's false because it uses FFNN)*

k) intent of AI chatbot cannot? *be derived from associated action*

l) shallow parsing? *doesn't require Named Entity Disambiguation*

m) **(Simulation)** True for transformer architecture? *Each encoder has: input fed into positional encoding, then multi-head self-attention, then ffnn*

n) **(Simulation)** fastText? *consider n-grams beyond entire words*

o) **(Simulation)** RASA story? *sequence of entities, intents, or actions*

p) **(Simulation)** MultiLingual BERT? *doesn't allow end-users to encode input text snippets longer than 512 tokens*

q) **(Simulation)** true for w2v? a *model with a larger context window takes longer time to train*

r) **(Simulation)** deep-learning sentence embedding model? *can be used to tackle IR tasks*

s) **(Simulation)** given query Q, recall of IR system? *measures the fraction of the relevant documents in the collection that are returned by the system*
t) **(Simulation)** NER system? *can be used to identify location references*