NLP

Systems often do understanding and generation together

Statistical approach, on neural networks

Books: natural language processing in action, online in early access

NLP is difficult because human language is ambiguous, everything can have many meanings

Def: This process translates natural language into data that the computer can process, and this data can be used to generate other language.

NL UNDERSTANDING→transform human language into something that machines can process.**Extract meaning, context and intent from a text**.

Transform the text into a numeric representation (embedding) to do the same as before.

Applications: search engines, emails to find spam, social media to moderate posts, crm tools to analyze customers and recommendation systems to suggest articles based on what you read.

NL GENERATION→generate human-like text that is coherent, in context, and based on the numerical representation of understanding

Applications:machine translation from one language to another, summaries of long documents, development of chatbot dialogues, content creation.

Es ambiguity: I made her duck,There are many possible meanings . So many inputs so many ambiguous things→ lexical, syntactic, interpretative, contextual ambiguity.

Various aspects of linguistics are used to interpret: phonetics, morphology, syntax, semantics, context. Structure analysis.

Linguistics focuses more on the study of language, the structure.NLP on the other hand focuses on implementing algorithms and generating human language.

APPLICATIONS: translation, document analysis, search engines, text prediction, summaries, social media monitoring, chatbots, sentiment analysis, anti-spam.

ELIZA was the first conversational agent in1960 but had no complex conversations, lost context and was repetitive.

TURING TEST: talking behind a wall and not knowing if you are talking to a man or a woman machine.

Limitations of the Turing Test:not reproducible , is emulating a human so there is a need to collect intelligence, some researchers decided to change focus, not used today.

1970 Limitations: flexibility (difficult to adapt to new contexts) and scalability

1990:statistical approach , long short term memory, learn patterns from data, flexible

2000:improve neural networks , introduction of word embedding, before word embedding la numerical representation of words was very inefficient, words are represented as vectors of numbers, similar words have similar vectors.

2006: google translation, efficient.

2010: lstm and cnn adopted for NLP, year of big data,large texts easily available on the internet to pull

2013: computationally efficient representation of word embedding.

2014: deep learning era, encoder-decoder, encoder transforms text into a representation numeric and the decoder in another text.

Virtual assistants like Siri, Cortana

2017:**transformer** , is an architecture that allows the creation of complex texts

attention mechanism:pay attention to a particular word to understand the meaning of text, this is learned by the system and not encoded .

After the transformer, other LLM systems have appeared exponentially.

LLM APPLICATIONS

Text generation, machine translation, chatbots, code generation, question answering, test summary, writing assistance

MULTIMODAL LLM

Integrate text with other media such as audio to text or text to image or text to audio

Tokenization

Divide the text into meaningful small units calledtokens, special case of segmentation

Often a token is a word, emoji, punctuation, numbers, prefixes or suffixes (parts of a word), phrases (ice cream)

One hot vectors

Token→vocabulary→**one hot encoding**(embedding, list with 0 except 1 for the word's index in the dictionary)

Advantages : I don't lose the sentence

Disadvantages : I have big tables even for small sentences

It is representative of a single sentence

BagOfWords

Sum all the one hot vectors

Lines: dictionary

Columns: sentence number

I save space but I lose the information about what sentence it is

Represents multiple sentences

Overlap

I count the number of identical words (that are in both sentences)**to judge how similar two texts (sentences) are**

Normalization

I also remove the punctuation or the stop words (infrequent words without important meaning or very frequent as articles that only add noise).But I might lose the meaning of the text or the difference between common nouns and proper nouns.

NLT

tokens = nltk.tokenize.word_tokenize(text)

download different tokenizers

Case folding

using regular expressions can be complex because all cases should be covered

so NLP libraries are used.

case folding, disadvantage: I lose the difference between common nouns and proper nouns

Removing stop words

remove words like articles or very frequent words like prepositions that do not carry any information.

is like removing noise.

I can use this stop_words list, it exists for every language

I get the essence of the text to make the BOW

disadvantages: in some cases I can alter the meaning of a sentence, when they carry information

** referring to notebook 2*

Stemming

Identify a common root between various forms of a word

Ex: housing and housed by house

How it works:

By removing suffixes I can combine words with similar meanings

A STEM does not necessarily have to have a meaning in itself→Relational, relate, relating all from RELAT

** * token.isalpha()→only if a token is alphabetic*

Stemmers are of course language dependent.

For "Go" and "Went" for example, even though they are the same word, there is no stem, as it only considers the structure of the word.

Lemmatization

Consider the CONTEXT of the word, use the DICTIONARY, does a morphological analysis, NEEDS TO FIRST IDENTIFY THE VARIOUS PARTS OF THE TEXT

Went→go

Better→good

Best→good

Difference: The lemma is still a sensible word, but it is slower than stemming

PART OF SPEECH TAGGING

Second step of the processing pipeline

PoS tagging is the operation that tags tokens based on their lexical category (name, adjective, article, verb…)

It is done before lemmatization

Each category also allows subcategories→name has gender and number

POS TAGGING ALGORITHM

It is complex because it has to do with the ambiguity of language, as the same term can represent different parts of speech.

Light can be a noun or a verb. Only**the context**can help to discriminate

There are various algorithms, often machine learning models are used. Or conditional probability is used,you assign the probability to a token of belonging to that category conditioned by the previous token. I have to assign the tag that maximizes that probability.

P(t_i) = P(w_i | t_i) P(t_i | t_i-1)

How many times is light a verb? How many times is it a noun? That way I calculate the marginal probability, then subsequently the**I multiply with the prob that the current word is a noun given the tag of the previous token (how many times I found a noun preceded by a verb).**

Ex: the light is bright

You determine that the is an article so t1= DET I do the problem**P(t2 = NOUN) = P(light | NOUN) x P(NOUN| DET)**

How many times light is a noun x how many times noun is preceded by an article

Many machine learning algorithms analyze the entire sentence, not just what comes before or after it.

MORPHOLOGICAL ANALYSIS

Useful when the algorithm does not know the token, the token does not appear in the corpus.**You look at the words that make up the token and you can tell which category it belongs to.**→infectious morphemes

Running→I know it's a verb with the suffix ing even if it doesn't appear

Post tagging allows LEMMATIZATION, I have to do it FIRST

spaCy

NLTK is inconsistent because for each task I have to import particular modules that are not even compatible with each other

- First tokenize with space
- Then it checks the individual tokens obtained to see if it can split into prefix, suffix or infix.

Dependency Parsing

Syntactic dependencies between tokens are identified (logical analysis)

Named entity recognition

Recognizes objects belonging to the real world

* all of these techniques can be used to decrease the size of a bow.

Term frequency

Instead of using binary numbers in the BOW we now use frequency: term frequency

I simply count how many times that word appears

It is better because the more the word occurs in the document the more it contributes to the meaning of the document., becomes representative of the document.

The limitation of this method is that it does not take into account the length of the document, so if in document b appears 100 times and in a 3 times,even if it appears more in b we are not taking into account the length of the document.

NORMALIZED TF

To address this problem we normalize the TF by dividing it by the number of words in the text 3/30 = 0.1, 100/580000=0.00017

NLTK Corpora

Corpus made of multiple documents, collection of news categorized into different topics

doc = nlp(sample) #Spacy does a lot of processing because it extracts a lot of stuff, but in this application we don't need everything, so let's change the spacy pipeline

Pipeline→tokenization, tagger, parser, ner

Nlp.pipe_names→I get the whole pipeline

EXAM: REMOVE OPERATIONS FROM THE PIPELINE TO SPEED UP

Vector Space Model

The matrix is the basis of all applications , I can make a search engine, calculate similarities, I have a numerical representation of the document corpus.

Each document is represented as a point in a multidimensional space The smaller it is the influence of w1 minus will be along the x-axis, so equal on the y-axis.

The axes are: TF of w1, TF of w2

How do I search for similarity between documents in vector space? **EUCLIDEAN DISTANCE** between documents, square root of the quadratic difference between the points of each document.

It is not used often **because it depends more on the size (length) of the document, a longer document will have more magnitude. The distance in this case also depends from the length.**

THINGS SIMILARITY

What is used is the COSINE SIMILARITY of the documents

Measure the angle between two DOCUMENTS→sim(A,B) = cos(theta)

The norm is the length of the document.

It is a value between -1 and 1 being a cosine

- 1→the vector points in the same direction in all dimensions, similar topics
- 0→vectors are orthogonal, they do not share words, they talk about different topics
- \- 1→opposite vectors on all dimensions, it's impossible with TF (just counting)

|A| = sqrt(sum i=1 an of a_i^2)→A*B is vector product

TF-IDF

Variation of normalization tf. This is the one actually used. Let's calculate the INVERSE DOCUMENT FREQUENCY.

Consider the uniqueness of the word in the corpus: Words like the, is, and appear frequently but do not help distinguish one document from another.

But subject-specific terms such as astronomy like stars or planets would not be discriminatory anyway.

IDF considers the importance of the word in the corpus

Increases the weight of rare words and decreases that of common words, the tf is the ratio between the word frequency in the corpus,divided by the sum of all dictionary words in the document (and not only of this word): it is different because considering the frequency on the whole vocabulary if the document has words that are not in the vocabulary and I consider documents on which the dictionary was not built it would not work (because it is still relating to the corpus in consideration). So I consider the occurrence of all the words of the dictionary in the document.→**IT SEEMS TO ME THE SAME AS BEFORE**

Idf is the logarithm of the ratio of the total number of documents in the corpus / number total of documents that include this word. If it appears in all I have N/N = 1, if it appears in 1 I have 10000/1. So for an uncommon word this number is really high.

The final calculation is tf * idf, so tf * a weight

Tf gives more importance to more frequent words in the document,**IDF gives more importance to LESS FREQUENT words IN THE CORPUS**

- HIGHT TF-IDF means that the term is frequent in the document but rare in the corpus,**THIS MAKES IT SIGNIFICANT/DISCRIMINATIVE FOR THE DOCUMENT (BEING RARE IN THE CORPUS)**
- LOW TF-IDFmeans not frequent in the document or in any case in the corpus, which makes it less discriminating for the document

** planet is wrong in the slide, it should be 0*

Zipf's Law

We use the logarithm to calculate IDF, **the frequency of a word is inversely proportional to the its rank in a frequency table**. It means that the most common word appears 2 times more than second most common, 3 times more than third, 4 times more than fourth, etc.

f(r) = K / r alpha

For example rank 2 appears ½ times, while rank 4 appears ¼ times and so on.

Using the logarithm in IDF tries to mitigate the influence of rare words (otherwise they would be worth too much)

So the lower the rank (1st position, 2nd position) the more frequent it is.

In practice, TF follows Zipf's law, while IDF corrects it to make the words in documents more informative.

Scikit Learn

Framework that does these things for us

There are also some calculation alternatives for TF-IDF

Building a search engine

- I have a corpus to search:
- I tokenize the corpus, create the tf-idf
- I ask for a query, I calculate the query as a document and I calculate the tf idf of the vector, **I only consider the words in the dictionary**
- I calculate the cosine similarity between the query vector and the rows in the tf-idf matrix.
- I identify the document with the greatest similarity to the query.

Real search engines compare the query**only with documents that contain at least one word of the query through an inverted index structure.**

TF-IDF MATRIX: lines→n.document, columns→dictionary

Text classification

It is the process of assigning 1 or more classes to the document for various purposes:

- topic labeling
- intent detection→if the text is to answer, to ask
- sentiment analysis
- spam detection

Classify documents based on text content, based on text analysis.

It is different from the document classification because there is no metadata. IT IS BASED ONLY ON TEXT CONTENT

Also classes are PREDEFINED, I have to define them. Otherwise I would have document clustering: that is, how documents can be divided and joined into similar sets.

Given a set of documents D and a predefined set of classes**, text classification finds a one function fi: DXC→ {true, false}.**

If true I can associate it in that class

If false no

Furthermore, each document can have multiple true, to go into multiple classes.

- We have single label where each document belongs in one class.
- Binary, same as the first but there are only 2 classes
- Multi label, each document can belong to multiple classes.

We have several algorithms, the best ones are based on machine learning, where I train a model with documents with associated class.

- Should I use a TF-IDF to pass the documents to the model or a vector representation?
- I also have to make a matrix of labels in hot encoding, with labels on the columns and documents on the rows.
- Then I use a machine learning model
- A sample is composed of the tf-idf vector with its labels.

CODE:

- Read dataset
- Convert texts and labels into numeric values, with TF-IDF MATRIX
- A sample is composed of (text, label)
- Split into test and train sets
- Make the network and tow it by giving input (text, label)
- See the results on the test

Word Embeds

Another way to turn text into vectors

New method, also deals with synonyms

Words with similar meaning have similar vectors

TF-IDF Limitations

Text with similar meaning has a completely different TF-IDF representation

If I calculate a TF-IDF of two similar sentencesbut who don't share words will have different vectors

Term Normalization

This technique also allows, through transformations, to collect words WITH SIMILAR SPELLING under a SINGLE TOKEN.

But with synonyms it still doesn't work

Or maybe they have the same SPELL BUT DIFFERENT MEANINGS

TF-IDF is still useful for some applications such as text classification.But for other more difficult problems like TEXT GENERATION, Automatic Translation, Question Answering or paraphrasing it is not very useful.

Applications that require semantic knowledge of the text.

TF-IDF is based on the concept of bag of words (THE DICTIONARY IS BUILT WITH BAG OF WORDS, THEN TRANSFORMED INTO NORMALIZED TF, THEY ARE THE COLUMNS OF THE TF-IDF MATRIX)

- The problem with one hot encoding is that the distance between words is the same (1000) is far away from (0100) as it is (0010) etc
- **This thing doesn't make you understand the semantics of the words, it's not efficient also because it uses sparse vectors (half-empty vectors 00100)**

With word embeddings the vectors are:

- DENSE
- With smaller dimensionsof the size of the vocabulary
- They work in a continuous vector space
- Vectors that generate words with similar meanings are similar to each other
- **POSITION IN SPACE REPRESENTS THE SEMANTICS OF WORDS**

- Implement semantic text reasoning , subtracting royal from king we arrive at man: **COMPOSITION OF VECTORS**(even if not exactly but it gets there)

If we subtract man from king and add woman we get queen

Interpret queries semantically **, adds the words and gets the word vector of the results→query "famous european", wv['famous'] + wv['european] = wv['Marie_Curie'].**

It also allows analogy searching by subtracting one word vector and adding another.

** An interesting property is that word embeddings of nearby cities are neighbors in the vector space*

**** Now we are looking for analogies between words, no longer between documents.***

Word2Vec (2013)

Methodology for generating workd embedding**based on neural networks**using **unsupervised learning on a large corpus without labels**

Words with similar meanings are often found in similar contexts, where the context is a sequence of words in a sentence.

2 methods to do it: Continuous bag-of-word, Skip-gram

Continuous bag of words

A neural network is trained to predict the center token of a context of m tokens**.. The input is the sum of the one hot vectors of the surrounding tokens.**

- The 2 words surrounding it will be the input: claude monet, the grand
- Output:probability distribution on the dictionary that has the missing word as its maximum
- N hidden neuron,where n is the size of the word embedding we want
- The size of the dictionary is the number of inputs and outputs.

Once towed, the output layer is discarded.

I only keep the input weights in hiddenwhich represent**the semantic meaning**, words similar have similar connections , similar contexts correspond to similar words.**The behavior of the network is similar, it arrives at the same predictions for the same contexts.**

*** it is an unsupervised neural network, so like lvq, competition of neurons where the closest one wins.**

**** I can also give other words, not necessarily the surrounding ones, just add the one hot vectors.***

**** the weights of a word, ARE THE SEMANTIC REPRESENTATION OF THE WORD, for similar words they are updated in the same way.***

Finally, as an embedding of a word, I take the weights associated with that word (that input).

Skip Gram

I train the neural network to predict the context, the words surrounding the token.

- Input: one hot vector of the central token
- Weight output:one hot vector of surrounding words (1 iteration for each word)
- Network output: probability distribution

** * In the slide, instead of Monet there should be Painted*

I discard the output of the trained network and keep the weights that connect the input layer to the hidden layer.

The more hidden neurons I have the more meaning I collect, but still carefully, not too many if I have little input.

CBOW→faster to train, very accurate for frequent words, useful for large datasets

Skip gram→works well for small corpora and rare terms

Emebddings size→not too wide otherwise expensive computation,**but broad enough to capture the semantic meaning of the tokens**. For complex texts I can use large embeddings to capture more semantics.

Word2Vec Improvements – Frequent Bigrams

Many words occur in combination , elvis is always followed by presley so it doesn't have much value if I predict it.

To make the network make meaningful predictions I consider these types of terms together as a single word in the dictionary.

Score(w1,w2) = count(wi,wj) – delta / count(wi) x count(wj), if it is bigger than a thrashold it I include it in the vocabulary as a single word

Sub-sampling frequent tokens

Words like stop word that do not carry information→ during training the sampling problem is calculated as the inverse of the frequency in the text.

To have a similar effect to tf-idf

Negative Sampling

Each training example causes the network to update all the weights, but with so many words in the vocabulary this process is expensive. **Instead of updating all weights I select 5 to 20 words (negative words) that are not in context and I only update the weights of those words (the links that start from that word) and the target.**

Word2Vec Alternatives

Glove: similar accuracy to word2vec, much faster and more effective on small corpora.It is based on singular value decomposition rather than neural networks .

FastText:

- based on sub-words,which predicts the n characters rather than the surrounding words.
- Effective for rare words, works on missed words and multiple languages.
- Generate word vectors even for unknown words

All these are**STATIC EMBEDDINGS** that is, each word is represented by a single static vector that captures the average meaning of the word based on the corpus.

But a word could have different meanings depending on the context : it's A PROBLEM

Another PROBLEM is that the semantics of a word changes according to time

Furthermore, word embedding could perpetuate and amplify social bias since the vector is static and depends on a fixed context (man is to doctor as woman is to nurse)

The datasets are full of content with BIAS.

WEs usually**They cannot process unknown words not present in the training data. FatText can.**

Lack of transparency:It can be difficult to interpret the vector space, difficult to analyze and improve the model or explain its behavior to stakeholders.

Contextual Embeds

Context is used in both training**that during use** , based on TRANSFORMERS.

Contextual embeddings can be updated based on the context of surrounding words.

Neural Networks for NLP

Neural networks are widely used in text processing , a limitation of feedforward networks is the lack of memory, they process without maintaining state.

In the models so far**we presented an entire text AS A SINGLE DATA** →bow or tf-idf or word vector media

When I read: I read word by word, I keep a memory of what I have already read, an internal model is constantly updated. This is the principle that a recurrent neural network adopts.

Process word sequences, iterating over elements and maintaining a state. We give the word as input embedding a word.

Each new piece of information is processed by keepingtaking into account what the network has already seen.

The output of the hidden layer both goes to output and returns to the same hidden layer for the next step.

At time t+1 the hidden layer will have as input the new input and the old hidden layer.

The output returned to the hidden is like a summary of what it saw before.

Input vect: size of the word embedding

THE WEIGHTS THAT CAN BE TRAINED ARE ALSO THE HIDDEN TO HIDDEN ONES

Useful output: it is the summary of the document WHICH IS CALCULATED BY THE NETWORK (not as an average)

The output is a word embedding of DIMENSION OF THE HIDDEN LAYER WHICH IS A SUMMARY OF THE DOCUMENT(I guess it's the weights)

The intermediate output, sent to the output, is usually discarded.

True Output = Depends on usage

Error = y_true_label – y_output -BACKPROPAGATION

How do I use the template?

- By classification, many to one
- Many to many (translation, tag)
- One to many: input 1 tensor, I get as output more tensors: question and answer (more words), I can also give an image

Text Generation

The output of each STEP, IN THIS CASE, is USED → and what the network saw in the previous steps

It is something generated by the network with respect to the input

RNN VARIANTS

BIDIRECTIONAL RNN →left to right as human, it is also read right to left to extract further information from the text

This is because information may eventually be lost due to the vanishing gradient.

It has 2 recurring hidden layers, one for each direction, the outputs of both are concatenated together

LSTM – Long short term memory

It is an evolution of RNNs that uses a MEMORY STATE with the various gates. Designed to solve the vanishing gradient problem.

Rules are introduced to tell you what to forget and what to remember

I have a cost that is additional processing to do

GRU – Gated recurrent unit

- Simpler architecture
- Does not use memory state
- Final performance similar to LSTM, resolves vanishing gradient

Stacked LSTM

I use multiple hidden layers, to capture complex relationships

INSTALL TENSORFLOW AND KERAS

Spam Detector

Classifier with word embedding and recurrent neural networks

Text Generation

We can generate text word by word or letter by letter.

The generated text is based on the patterns and structures extracted from the corpora

The best way to generate texts are transformers, but I can also do it with RNN

Applications:

- automatic translation
- answer questions
- automatic summaries
- complete the text
- systems of dialogues
- automatic creative writing

Language Model

Mathematical model that determines the probability of the next token compared to the past

The model captures the STATISTICAL STRUCTURE of the language

You give it a token, the model generates a new token, adds it to the input, and repeats this several times.

I can decide when to stop

TRAINING

At each step, the RNN receives a token extracted from a sentence as input and produces an output.

Compare the output just obtained with the expected one that I will give as input in the next step. (it's just the RNN recursion that makes it do this appears in my opinion)

Unlike traditional RNNs,**I don't do backpropagation just at the end, but at every step.**

The weights are influenced by the difference between the predicted word and the correct one.

Generative Models vs. Discriminative Models

Using generative models is different from discriminative models which are based on probability, when we use generative models we make a random sample from this distribution and put the result.

The difference is that in the discriminative ones we use the most probable, in the generative ones we do sampling from that probability distribution

** we use generative*

Temperature: T parameter used to adjust the randomness of the sampling

The lower the temperature, the more deterministic T<1 is, the higher the temperature, the more probabilistic T>1 is.

With low temperature I have the network distribution

With high temperature I have uniform distribution, all with the same probability

➔ Expression q' to calculate the new distribution and see how it is affected by temperature

Q'i is the new probability of the sample with probability pi (that's why it is divided by all samples), if T decreases the ratio is large and the exponential is even larger. So the probability of the sample increases.

Dialog Engines

We have several conversational AIs

Chit Chat→ It does not have a specific goal, it focuses on natural responses, the more turns pass by conversation is better

TOD→helps the user achieve a particular goal, the focus is not to entertain the user but understand his intentions and what he needs and generate the next action. In this case less conversation turns the better (get to the goal first)

Task Oriented Dialogue TOD

Uses: questions, tasks or advice

Shave: particular framework for this kind of systems (train models), you ask something to the chatput, an input module analyzes the input trying to get what the user wants, it analyzes itself and then pulls out the response from systems it is connected to as a database

What you need to do is to understand what the user's PROMPT is, you need to do 2 tasks:

- **intent classification** →Multi-label classification task
- **entity recognition** →I must recognize particular entities that SPECIFY the intent, peculiar entities (intent: weather, date: tomorrow). It is done with NER (rule base or ml based)

Conversation design – assistant design

I can plan a type of conversation the assistant should have:

The assistant should ask who the users are, what the purpose of the assistant is, and I have to document What is the typical conversation a user might have with the assistant?

It is impossible to anticipate everything the user may ask, so at the beginning I can count on hypothetical conversations and then real ones as soon as possible.

SHAVE

Framework

The basic units of rasa are:

- **intent**the→what the user wants to achieve
- **entity**→relevant terms or objects that specify the particular intent
- **Actions**→what the bot should do in response to intents , the most used are 2 namely **RESPONSES**(predefined expressions)or interact with other systems by calling python code

- **Slots**→variable used to keep information extracted from the conversation for later use (NOT IN THE NEXT ACTION)
- **Forms**→set of slots to capture multiple pieces of information
  ** the network is almost deterministic, it uses a probability distribution from which it takes the next most likely action*
- **Stories**→sequences of intents and actions used to program predefined dialogue scenarios

Eg: in this case the chat perfectly matches the story,but in normal cases he has to find the most reliable story to give after

Domain.yml

I have the stated intentions and the answers to the intentions

There is also the duration of the session, if it ends the data of this session must be transported to the new one

Nlu.yml

In "date"

For each intent there are various samples used to train it and which in an expression will lead back to that intent

To recognize an INTENT, RASA NEEDS AT LEAST 7-10 EXPRESSIONS THAT HE WILL USE IN TRAINING

Stories.yml

Describes possible interactions between the user and the model

Intent – action – intent – action

There is no need to match this flow perfectly, the closest one is used

Rules.yml

If the rules are recognized then the action is implemented.

With**shave visualize**→I can visualize the flow of the stories as a graph

Commands:

I have to go into the "myBot" folder if I have already done rasa init

When I change anything I have to re-trick the model → **shave train** and stored in . /models

Shell shave→I can interact with the model from the command line

Rasa run→start a server with my model towed

Shave -h→help

- To make rasa communicate with the outside I have to uncomment something
- I can do POST JSON requests, the recipient_id is the same as the request
- From github I can download a tool for Frontends interfaces

Building a Chatbot with RASA

The domain file contains actions, intents, entities etc.

If I have multiple responses under a single one utter_great will automatically choose one

I can also add buttons to the RESPONSES, when clicked the INTENT changes, it's like I force them to send the payload I wantto facilitate the recognition of the INTENT

I can add intents and define entities,**BETTER TO START WITH AS LITTLE INTENT AS POSSIBLE**

NLU File(natural language understanding)

It is used to recognize intents and the entities involved.

Contains additional information to recognize entities.

In square brackets I specify a sample of entity + entity type after

Lookup_table→where do I look for entities, give examples of entities, can I also put synonyms or regex

I can also use a standard recognizer

Without tables and regex custom entities are recognized with machine learning, but external things like spacy are needed

Entity roles→ I add more information for entities after naming one in an example for INTENT

Good Practices

Start with a few common intentions, the others will come from real use

To store information it is better to use entities rather than intents

Stories File

What the assistant should do next compared to the past , after recognizing the intent the model creates a probability distribution and chooses the most likely action (based on the stories written) . If he doesn't understand, he uses a standard action saying he doesn't understand.

Often a happy path is used to start and then adapted to real conversations.

- **OR** →same action for different purposes
- **Checkpoints** →I'm connecting to other stories

It is used to tow the model

We have 3 models:model that recognizes intent, that recognizes entities and that recognizes past interactions (stories)

Rules File

If something happens then respond this way, always.

I specify in the configuration files the priority order between rules and stories

Machine learning was not applied

Slots

It's like the memory of assistants, variables that store information during the conversation, **Sydney you have to recognize it first as an entity and then store it in the slot** (I can call it whatever I want)

They can be used to influence the flow of communication

Declare slots:**They are declared in the DOMAIN FILE**→I have to specify the type, which can be text, boolean..., I have to say whether or not it influences the conversation and the mapping: every time it is recognized a target entity is saved in that slot. If a new one is recognized replaces

I can also say that they must be respected **both the entity and the intent and not have it replace any time.** I can also specify that this type of entity must not be changed with that intent

I can also specify role

{name}→can i include slots in answers

** destination is the name of the slot, it can be anything*

** if name is not set, its value will be None*

Config.yml→NLU pipeline

Defines the bot's language, pipeline specifies the time to process messages, policies says how it should predict next actions. I can also leave it unconfigured (null).

PIPELINE→The pipeline is composed of sequences of components to process the message (the ones we've done so far):**tokenizers** ,**Featured r**(BOW, TF-IDF, WE),**classifier** (the most used is**DIETClassifier using transformers for multi-task problem** which classification is intended which extracts entities), epochs,**entity extractor**

You can also use spacy tokenizer

I can also specify training policies→ techniques that the assistant uses to decide the next action, I can specify THE PRIORITY WITH WHICH THE RESPONSE IS CHOSEN

Rule Policy:prioritize the rules (in this case it is the last in order of priority)

Momoization policy:try to answer FOLLOWING THE STORIES, try to match the flow of the conversation with the written stories . (**It is a rule based, NOT MACHINE LEARNING**)

TEDPolicy:most used,**Transformer neural network trained on STORIES FILE**, I can specify the maximum number of previous messages to consider (max_history) and the number of epochs to train the model

In the config I have to allow the use of LOOKUP TABLE

The policy comes into play when subsequent decisions have been taken by different policies but with equal probability.

Custom Actions

I don't just want the assistant to answer me, I ALSO WANT IT TO DO SOMETHING, TO INTERACT WITH OTHER SYSTEMS.

- Send an email
- Make an appointment
- Get information from a database
- Confirm information
- Calculate something specific

- **NLU CORE of rasa** →understand what the user is saying and respond appropriately
- **RASA SDK** →used for these custom actions

To specify an action I go to**actions.py in the Actions folder**

I can import things

ACTIONS ARE DECLARED IN DOMAIN.YML

1. **It is**: I specify a dictionary which in this case is my city database, with entities equal to those in the lookup table.
2. **I need to define a class that extends the Actions class.**
3. 2 methods: name method and run
4. **The name method returns the name of the action (which is the name specified in domain.yml)**
5. The run method tells what happens as soon as the action is called**, takes the dispatcher that allows you to send messages to the user** ,The tracker provides the status of the conversation .
6. I take the value of the last place entity recognized by the model (with the tracker)
7. Then I calculate the UTC time
8. If current place is not set, I give the current time + message and send it with dispatcher
9. Otherwise I take the right format from the database made before, if null I send an error otherwise I invoke the utc.to method
10. Send the message

To enable custom actions, I need to enable the action server in endpoints.yml

Then I do: RASA RUN ACTIONS

AND IN ANOTHER TERMINAL I MAKE SHELL RASA

ENABLE REST: # in endpoints.yml to start the server→then I do rasa run --cors "*"→then I open the html

Use (pizza) if it is only 1 information of the entity

It's called lookup→no lookup_table in nlu

Type: from_entity→means that the slot will be filled with that entity

PIZZERIA BUSINESS

Transformers

For a chatbot, behind the scenes, you use**RECURRENT NEURAL NETWORK**

Before transformers it was the only way to create chatbots

RNN PROBLEMS: long-term memory loss (enc-dec models) → if we have a long text with 2 sentences, we have a long sentence and a small one, in the second one we have "it" which refers to the first sentence, the name is far away and the rnn could forget this important information

They are slow to tow for long series : the longer the series the longer the training time,**TO MAKE THE NEXT STEP, I NEED THE PREVIOUS ONE, SO I CAN'T PARALLELIZE.** I can't do computations on sentences in parallel

They suffer from the vanishing gradient :backpropagation of loss, if I have many levels, it decreases the gradient which we add to the weights. So it's like we're increasing less.

- **Solution to lack of memory**→Instead of analyzing a single word each time, I can consider that I already know the previous sentence, I can use it to find the relationships between each part of the sentence.
  Let's take into account the relationships between words.**And we give it a threshold of importance.**
- Slowness solution→in transformers every computation for every word**can be parallelized**
- Vanishing Gradient→I have a lot of multiplications, the problem also arises because through many times the same level (**having to unfold**), transformers solve it

** paper on transformers, they used ATTENTION for the first problem.*

It has an Encoder-Decoder architecture (the original one)→ for each word I have an embedding and I give it to you give as input

The encoder gets an intermediate representation of the input sequences, **I have an encoding of the meaning of the PHRASE as output** . (new word encoding)

The dimensionality of the input is the same as that of the encoder output

The decoder uses the meaning of the entire sentence,**the first word of the translation is generated, and uses all the previous words to generate the next one.** (it can be parallelized because you can give all the words, he rephrases the embedding based on the context)

All the output is the translation, but at each step it gives the encoding of one word.

SO IN THE END YOU GET BACK A CLASS, 1 WORD IN A DICTIONARY IS NOTHING MORE THAN A CLASS.

INPUT

Tokenization:we can choose which token to have.With characters it is difficult to get the meaning total, with words I have a dictionary too big (output vector too many probabilities) . I use sub words because I get a good trade off. (there is a relation that transforms word into sub words)

** word embedding, I must also encode end of the sentence*

After token I have embedding for each token. The length is d_model

The encoding problem used so far with RNNs**is that we would like to do all the parallel computations**, so we lose the concept of sequence, we don't have a way of take into account the position in the method used so far.

In parallel the order could completely change the sentences

ES: The encoding for apple in the sentences "the students s eating an apple" and "an applies eating the student" is the same, it does not depend on position.

Positional encoding

I want to add a small perturbation for each word in the dependent sequence. from the position within the sequence.

This way the same element in different positions will have slightly different vectors.

I don't lose the semantic meaning.

It is made for both encoder and decoder.

The most used positional encoding is the one with PERIODIC functions.

With different representations for even and odd, the value we add to different token sequences depends on the size and position.

- dmodel is the size of the embedding
- i is the position of the element in the embedding vector
- position of the word in the sentence.
- PE is a vector**of the same size**of embedding from**ADD to normal embedding**

PE(pos,2i) = sin(pos/10000^2i/d_model)

This will be the input of the transformers

We solved the slowness problem for parallelization by inserting the order.

Now I'm trying to solve the lack of memory problem.

ENCODER

Transforms an input sequence of vectors into an intermediate representation of the same length.

The vectors z1,...,zt can be generated in parallel

Each vector zi does not depend only on xi but on the entire input sequence (precisely thanks to the concept of position)

Context is added by computing for each word the relationship with itself and with the others.

How much this word is related to word 1-2-3...

How much should I take into consideration word 1-2-3... while considering word 1?

The new encoding is generated by considering: original word, position and correlation with the others - context representation

The dimensionality of the output is the same as that of the input.

IF I USE 12 BLOCK ENCODERS THE BLOCK SIZE IS ALWAYS THE SAME, by adding blocks you only increase the quality of understanding the context.

The encoder is made of a sequence of encoder blocks with the same structure, possible precisely because of the the fact that input and output have the same size.

Each Encoder block uses:

- **self-attentive module**, where the same vectors are used as **Q, K, V**
- A **feed forward layer** applied to each element of the sequence
- A **skip connections** for vanishing gradients (I don't do some multiplications)
- And normalization to obtain vectors of the same size (because the embedding is multiplied by weights that may have a different size than the embedding)

Self-attention

What does "it" refer to?

I can tell it apart because I can understand how animal, it, wide, street are related to each other.

Self attention means understanding which words to consider most when encoding it

Of course it depends on the topic.

An attention function is used when I need to compute the value of a word that depends on other words (in the sentence) and we want to give different weights to these words based on how much they are related to ours.

"How much animal should I consider for the encoding of the"

I have to introduce **the attention function.**

Attention function

It is used when the value that has to be computed (value of the token in a certain position considering the context) depends on a set of other values (sentence) and we want to give different values each time i.e. how important each token is the token I am considering.

Query: How much each value is related to this one, query from my world with key from another world allows me to understand the relationship. Query, key(another word), value(value to multiply by weights)

"Let us assume that for each v_i we have an additional piece of information k_i that can be used to decide how much attention to give to v_i."

We have an input value qe**we want to define a target function**

Q is the query, A WAY TO ENCODE THE WORD

Both q and ft(q) are vectors

We want to express ft(q) as a function of a set of elements – words of the sentence

We want the attention given to each person to be different based on the q

For each vi I have ki that can be used to decide how much attention to give

For each input embedding we need to get 3 vectors: query, key, value

- **Query**It is used to find the relationship between this world and another
- **What**y I'm from another world to meet and find similarities
- **Value**It is used to give a weighted contribution to the embedding of the word I'm querying for

F = a(q,k1)*f(v1)→this is the weight * fv(v1)→attention between word q and word k multiplied by a certain weight v

Alpha is the attention function

Ft and fv are different functions

We want alpha and fv to be learned by our system

The total sum must be 1 because it is as if it were a probability , we want a score between 0 and 1 for each word, we can interpret the score before the end as a probability.

The transformer uses a particular definition of attention function based on linear vectors and softmax function

AND':

- differentiable, computationally efficient, parallelizable
- 3 input matrices: Q, K, V
- Q has query vectors (mx dq)
- K i key vectors (nx dk)

- V i value vectors (nx dv)

They are all matrices because dq,dk,dv are all vectors that represent the size of the embedding

Cosine similarity is done with the scalar product.

If the query and the key are represented by vectors of the same size, a matching score can be obtained with cosine similarity.

We want to have a value between 0 and 1 so SOFTMAX

Procedure:

query computation, key, value – I multiply them by weight matrices WHICH ARE LEARNED FROM THE NETWORK

From the product I get:**Q' = Q * W_q, K' = K * W_k, V' = V * W_v**

** same columns for addition*

Query, key and value calculation for each word

Self attention matrix using query of my word and key and value of other words, aij is a value of attention function

We get the various scores, the sum must be 1 for the softmax

(selft attention block)

Finally we multiply by the weights, it is the weighted sum

The final result is the encoding of that word which takes into account the meaning of that word. word in the context of the sentence. (it's actually the target matrix for each Q)

Finally we put a normalizer to have the same input size

** don't think about mask*

I have a sentence, I put the embeddings of all the words into a matrix.

I use this matrix and multiply it by wq to get Q

I multiply it by wk to get K

I multiply it by wv to get V

Explanation

If I do softmax(QK^T / sqrt(d_k)) I get the matrix representing the attention function, where for each pair qk I have the correlation (represented by element aij).

I then multiply this by V to weight the results. The new matrix will contain the new embeddings for each word.

From the point of view of the single word→I have the Q of the single word, I want to find the correlation with respect to kj (i.e. the element aij of the attention matrix), I multiply Q by Ki and divide by the sum of this multiplication for all the keys (so as to fall within [0,1]). I multiply this result by the V of that K.

To obtain the final embedding of Q I add all the final results obtained.

** The continuous bow (bag of words) assigns the embedding vector statically, it does not vary based on context.It's just based on the words surrounding that word. , is unable to define that word based on the words in the entire sentence and therefore on the entire context.*

So finding that word again, later on, maybe with a different meaning, it will have the same embedding and I lose a new semantic meaning that it could have assumed.

Transformers 2

Self attention is not enough to store the meaning of the whole sentence as one word
It could have more meanings.

“apple” has multiple meanings

So we use multiple-head

Multiple Q,K,V matrices. One Q,K,V for each head learned during training, this way I can learn multiple meanings of a word.

It is not a problem to parallelize because it is as if we have more words in a sentence.

The results (the output of each level) are concatenated row by row and then multiplied by a final weight matrix to normalize and get the same size of the embedding again.

We have weights for each head

Add = skip-connection and norm

I normalize the values so that mean is 0 and std dev is 1 (similar to batch normalization in CNNs), and then regularization that makes the next level independent of the average of the previous one.

Skip connection to eliminate vanishing gradient and tow deep nn.

Feed Forward

We add nonlinearity to get more complex combinations . There is a feed forward layer FOR EVERY WORD in the representation.

Even after this we have a skip connection

After THIS OPERATION WE GET THE OUTPUT: that is, a representation of each word of the sentence that takes into account THE MEANING OF THE ENTIRE SENTENCE.

Output and input have the same size, so I can stack multiple encoders to get deeper representations.

** the encoder is just the square*

So the encoder:

- **use positional encoding**needed to do multi head self-attention (for parallel)
- use residual connections for vanishing gradient (**less multiplications is better, it goes faster**)
- **normalizes to stabilize the network**
- **adds nonlinearity (capture more meanings)**
- **Output size equals input which allows stacking encoders**
- the output of the first is given to the second encoder block

DECODER

Uses the information from the encoder's intermediate representation to generate the sequence of output y1,...,ym.**I have m because the output sequence can also be of different length.**

The decoder works sequentially.**At each step the decoder uses all the encoding and words already generated to generate yi.**

I have to recalculate y1,...,yi-1 every time even though I already have it→**because in self attention I have the softmax, if I add a new value it changes (because I still have to generate them such that the total probability is 1) (even if I can reuse some of them)**

The decoder is a sequence of decoder blocks with the same structure

Keys and value are taken from the intermediate representation of the encoder z1,...,zt

Because at the beginning of the decoder we have AN ENCODING OF THE NEW SEQUENCE

So let's also input AN ENCODING OF THE OLD SEQUENCE

Key and value from encoder, and query from decoder.

We use world queries that we want to encode but we also need to add the meaning of the key from the other world.

How much of the original sequence do I have to consider to generate the new word.

MASKED SELF-ATTENTION:The query at position i uses only the words (value) up to i (USED ONLY DURING TRAINING)

After that we do a normalization

I add a linear layer and softmax to compute the probability of the next element yi

So the last level has a number of neurons equal to the cardinality of the target dictionary.

For the first word, we don't have a query to give, we use a dummy word, SOS start of sentence→I already have the embedding code, it's already defined, so I already have the positioning, it's something that is used.

In the first word the contribution of Q is limited, it is as if only encoding was taken into account→ he considers it as if it were the subject

Masked multi-head attention

We know what we just generated but we don't know what we will generate. I have to mask the other elements of the sequence.

This is done by setting the other values to -infinity so that the softmax of the other values is 0. (in the softmax I have e^-inf)

In the first step I get the encoding of <sos>.

It is important to do this, otherwise the network uses future knowledge that it does not have.

Encoder decoder attention

Use kev from encoder output and q from decoder input

** the decoder block are all 3 so, if I have more,they each receive the same encoder output*

Output

I use the information to generate the next word in the sentence, with a classifier. The important thing is that I have the output as probability (and the layer) with the same SIZE AS THE DICTIONARY.

Softmax or temperatures (**low is more rigid, it chooses the one with the highest probability, if I repeat several times I always get the same result – high is the opposite**)

Transformer's pipeline

I have 1 transformer architecture so far. Even just 1 part of the architecture can be used for some tasks, The encoder alone could be used for sentiment analysis, document comparison etc..

Only decoder can I use it to generate one word from another

The decoder pipeline continues until <eof> tokens are generated.

** in the tool there is only the decoder part, it generates new words from the sentence*

** the dots are the correlation value that the final word has with that word for each step (which are different for each head)*

** in the last slide I have the giant matrix because to calculate the probability each token is projected into the space of the entire dictionary*

* * LEARN THE LEVELS OF ARCHITECTURE WELL

Transformers to LLMs

Transformer is just the architecture to create the language model

Large stands for the amount of data we use to drive the network, but the most important thing IS HOW WHERE THESE NETWORKS ARE TRAINED

The data we use for pre-training does not take into account THE TASK OF INTEREST (without annotation)→ they use SELF SUPERVISED LEARNING

After the pre-training there is the FINE TUNING for the SPECIFIC TASK

Transformers for text representation and generation

- Encoder does representation and decoder generation
- The encoder alone could be used for sentiment analysis, intent/entity recognition (classification tasks that involve comparing word embeddings)
- Transformers: sequence-to-sequence translation, summary
- Decoder: Generate Text

Representation (encoder):

- input→input tokens, output→hidden states.
- We can see all the time steps (the complete sentence).
- We don't have auto-regression because it doesn't see the output tokens
- It can be adapted to generate tokens by adding a module that maps hidden states to the vocabulary

Generation (decoder):

- input→output tokens and hidden states, output→output tokens
- I can only see previous timesteps (masked)
- It's autoregression
- It can be adapted to generate hidden states by looking at the output first

First examples of representation and generation: bert and gpt

Transformers: t5, bart, mt5

Paradigm Shift in NLP

Before:

- **Feature Engineering**: How do we design or select the best features for a task?
- **Model Selection**Which model is suitable for a specific task?
- **Transfer Learning**: How do I transfer knowledge to other domains if the text is not annotated?
- **Overfitting vs. Generalization**

Now:

- I focus on**pre-training and fine-tuning**to handle large amounts of unannotated data
- **Zero-shot(no sample) and few-shot learning (few samples for training), I want to make the models perform on tasks they are not trained on.**
- **Prompting , make the model understand its task by describing it in natural language, what is the best input I can give to the LLM to better explain and exploit its internal knowledge.**By slightly changing the input I can get completely different performances, if I use the same format they used during the fine tuning improves the quality of the response.

- **Interpretability and explainability: How do I understand how our models work internally? Self attention more or less gives us an idea, but by increasing the heads or stacks of encoders and decoders it is difficult to reproduce the reasoning to obtain a certain result.**

* also the input embedding could be wrong and cause problems

The shift to LLM was done using the ATTENTION MECHANISM.

Pre-training of LLMs: SELF SUPERVISED LEARNING

If we don't have text with label

I can split the sentence into two halves, I use the first one as input and the second one as output(what it has to generate)→gpt is pre-trained like this (Decoder) [the mask sets the token to predict to 0, so it doesn't calculate the correlation with itself but only with others. Its embedding vecotr is modified anyway]

I can also do next sentence prediction, I have two sentences and I want to understand if the second one is after the first one, so I train with two close sentences

With this self supervised pre-training the network learns (in terms of weights) the correlation between words, or syntax (even if it doesn't learn a specific task), LEARN THE RELATIONSHIP BETWEEN WORDS, the effects of words on others.

** so I use pre-training for weights*

THE MOST USED one is to predict a word contained between a word before and a word after. (encoder) . → The encoder does a classification task

There is also autoregressive (decoder predicts word) and seq-to-seq i.e. hybrid where the output is the sentinel token followed by the predicted tokens.

Masked words, I need an encoding for <mask>

Attention input and output is the correlation matrix (I think**), the correlation entropy between <mask> and "love" must be minimized during pre-training.**

- **Auto-encoding : predict the embedding of a word included among others, ENCODER ONLY (the dictionary labels it is compared with at the end have word embeddings) (not great for text generation because it works with entire sentences)**
- **Autoregressive : predicts the masked word that follows the previous ones, DECODER ONLY (does not have a global understanding but only of the past)**
- **Seq2Seq (span corruption) : A sentinel token takes the place of a random word, the output is the sentinel token and the associated text. ENCODER-DECODER. (predicts more than one masked word)**

Can I use it in zero shot knowledge?→ in training I have positive and negative sentences (like comments) even if not explicitly,**I have the opportunity to do few shot learning**. Ex: you are bad. I tell him it is negative. You are good I tell him it is positive,**then I send a new sample and he will classify without having to tow the model FOR THE SPECIFIC TASK** .

** What it does is consider the correlation between words, if they have more or less the same embedding I give them the same label.*

Pre-training tasks:

- It can be invented
- Creates general models to be trained on specific tasks, builds embeddings with correlation between words and syntax
- Learn to generate language without relying on a specific task
- It can be used as a basis for zero-shot learning on a range of tasks.

Datasets and data pre-processing

The**data preprocessing**It is used to have high quality data for pre-training purposes.

- bookCorpus and gutenberg are good datasets for books
- commonCrowl, web page data, preprocessing needed because data is not always clean
- Wikipedia

** pubmed gpt for medical questions*

The**pre-processing**is to remove low quality, toxic or unethical text. Removes duplicate or redundant content.

- Quality filtering, it is done with heuristics
- Deduplication, removes repetitions
- Privacy scrubbing, removes things that can identify people
- Filtering out toxic and biased text

Using LLMs after pre-training

We can fine-tune it to specialize it on a specific task , it is difficult to fine-tune the entire network, I use PARAMETER EFFICIENT FINE TUNING(PEFT) to focus only on specific aspects →

I'm changing the model.

** I might not get a better result either*

Or**prompting**, we specify rules for answering questions and using the knowledge that already has, the model is able to understand the "meaning" of the text → **the model does not change**

(it compares words for classification I imagine and chooses the meaning class most similar to those we give it)

When we talk about few-shot learning (and zero shot) we are referring to the model's ability to prompt, and therefore to make inference.

Instead, fine tuning also includes changing the weights on the new dataset and on the new task.

Hugging Face

It's a repository where I can find new models and datasets and spaces for demoes and code (with information about for example how the model was trained)

There are 2 important libraries: transformers and dataset, and evaluate which is used to evaluate the performance of LLMs (although standard evaluation metrics for models are not very valid for LLMs)

** the pipeline is not only valid for llm but also for other types of tasks*

I also have an interface to test the selected model

I can look at the dataset card to understand if it is suitable for example for zero-shot or for a certain task.

Pipeline

Features to create and use the model

Pipeline() function

Connects a model to preprocessing and postprocessing, intermediate steps are hidden

Pre-processing→model→post processing

Classifier = pipeline("sentiment-analysis")

Finally choose the task to do to create your application, you can also do fine tuning or zero shot

I can also chain multiple pipeline objects together which builds me a custom model which would be a collection of other models.

Model Selection

I can select models by task, license (if I have to use them for commercial products), language (I have to know the language with which it was trained otherwise in the pipe I have to add a translator)

I can also see if the model is stable or still under review

Common models

GradeIO

This interface allows us to create a web application with python.

I create a model

And with gradio I can: containerize and deploy the model, store samples to create a dataset with the format for training or fine tuning, and create an interactive front end

I have the ability to put my gradio interface on hugging face

Can I publish my gradio-made applications online?

CREATE EXAMPLES WITH LLMs WITH DIFFERENT FEATURES

Encoder only transformers

I need all the transformers if I need to transform one sequence into another of different length.

If I want to transform it into one of the same length or of a single value I can also use just the ENCODER.

Same length→I can use the vector at the end and do the loss function on this, eg: named entity recognition, noun pronoun or verb for each word.

Single value→classification, sentiment analysis, I add a special value START as the first element x1 of the input.

Then I take the final element z1 as output and do the loss function only on this.

BERT

Bidirectional encoder representation→ I have the ability to analyze both word first that after to do word embedding, I can use knowledge of the word before and after to embedding a word . (such as self-supervised training).

That's why the encoder is bidirectional,**the decoder is autoregressive instead**:I can only use what comes before and not what comes after.

Due to the fact that MASKED ATTENTION is not used.

Bert has 12 stacked encoder blocks with 110 million parameters, bert large has 24

I can use it for other tasks with fine tuning

Bert uses a WordPiece tokenizer→ subword based, it can therefore know both words common than rare by dividing real words into small pieces.

Vocabulary→vocabulary made of common words and their subwords

If I don't know the word "unhappiness" I can still get the encoding by joining the encoding of un, happy and ##ness

BERT INPUT ENCODER

Bert needs specific tokens:

- **CLS**→ **classification token placed at the beginning of each sequence.**
- **SEP**→ **separator token to define the end of a sequence** (to make next sentence prediction for example)

[CLS] tokens [SEP]

Token to ID after tokenization to map them with the vocabulary and then feed them into the model.

Efficiency, unfamiliar words

CLS (classification token) TOKEN

Used in the output as a summary representation of the entire frequency

So cls contains ALL THE CONTEXT OF THE SEQUENCE

I can use it for classification:

- **Single-sentence classification**→The embedding is passed to the classifier layer to make predictions.
- **Sentence-pair tasks**→uses CLS to encode both sequences, maintains the context of both sequences.
  I can pass two sentences that both start with CLS (it's as if it considers them as one sentence), separated by [SEP]

BERT pre-training

It is pre-trained using self-supervised learning

MASKED LANGUAGE MODELING→predict which word is missing in a sentence. Masks 15% of the elements at random.

NEXT SENTENCE PREDICTION→determine if a sequence B is a possible completion of sequence A (can do this if they have the same length)

Bert fine tuning

Instead of cls there is class

With general knowledge Bert understands the context of a sentence while the final tuning does various tasks using the cls.

Text classification, named entity recognition, question answering by adding only a few layers for each task

It's as if the representation capacity of the CLS itself were particularized. As if I'm training the CLS.

** even without fine tuning we could have zero shot knowledge by exploiting internal knowledge*

Advantages:

bidirectional context awareness, flexible in transfer learning, very performant on popular datasets.

Limitations:

large model size, high pretraining cost, fine tuning requires a lot of data and time.

ROBERTA→different from BERT because it was**trained on more data, the goal was to decrease bert's computational time**.

Removed next sentence prediction in pretraining to improve performance . Training and**longer batch**(I calculate the loss on multiple samples, I have an update of the weights that take into account more information).

Dynamic Masking means changing the number of masks over time.

** the shuffle is done because I might have batches with only men's faces for example*

** I get a memory error if I use the whole dataset instead of the betch because I have to store all the gradients.*

ALBERT→**the differences are that I have reduction of parameters** , shares parameters between layers, and does SENTENCE ORDER PREDICITON (sentence 1 before sentence 2 and vice versa).

DistilBERT→knowledge distillation (or student-teacher method):**use the labels made by an expert (the real BERT) and these labels are used to train a smaller model that performs a simpler task**, try to learn to**mimic the behavior of the real model** instead of trying to understand the actual dataset.

I get a replica of the large model with fewer parameters(teacher-student approach). Reduced by 40% while keeping 97% accuracy, fewer layers, faster.

TinyBERT→2 step knowledge distillation, between pretraining and fine tuning, in pretraining they used bert and in fine tuning using THIS MODEL. Lighter than distilbert and used on mobile, similar accuracy.

ELECTRA→classification, replaced token detection, instead of using a mask they removed some tokens.**More efficient pretraining, high performance.**

SciBERT→academic research applications,**THEY HAVE DONE A SPECIFIC DOMAIN PRETRAINING** , the dictionary is always built on the basis of that domain, better performance on scientific tasks than bert.

BioBERT→biomedical applications

ClinicalBERT→hospital applications or helps with clinical decisions

mBERT→used for multilingual support. NLP tasks between multiple languages

Other variants: camembert, finbert, legalbert, IMAGE ENCODING→a batch of images can be thought of as a word

EXERCISES

(I can do fine tuning with 1 training epoch)

Decoder-only transformers

It is a transformer that uses only the decoding part. It is autoregressive and does generation tasks.

It is used for language generation, summary and question answering tasks.

GPT(not all versions are well documented) and LLAMA(I have all the info on the architecture)

Each token is generated sequentially based only on tokens from the same sequence generated before, instead of using encoder-decoder attention.

Since it is autoregressive I can't use encoder keys and values.

The prompt is a continuous sequence, each word is added to this sequence.

The context is taken from the prompt which itself becomes QUERY KEY AND VALUE, each time a word is generated it is added to the prompt. The final output will be a probability distribution on the dictionary from which the most probable word will be taken and then added to the prompt.

By giving the whole sequence as input to the masked self attention we can obtain both the context encoding and the decoding , without using a separate encoder block.

So we have self-attention with causal masking, we can use only that word for each token and the previous ones. I hide the future . **So this MECHANISM IS USED FOR ENCODE THE PROMPT CONTEXT.**

The difference is that in the multi head attention I put the newly generated words and not the encoding ones.

The masked is the same.

Context is built as it generates words.

Pre-training:next word prediction

Context:**process tokens one at a time**

Output: sequential token generation

Text generation, conversation ai, programming help, summary

GPT

Generates text, can do various tasks without specific training

GPT INPUT ENCODER

BPE→Byte-Pair encoding, use subwords to balance between letters and words, fixed vocabulary, we have more flexibility, predict out-of-vocabulary words.

GPT pre-training

Minimizes cross-entropy loss on the classification task. Adam optimizer with learning rate scheduling with warm up. Large batches.

It predicts the next word based only on previous words, compares the predicted word to the expected word, and updates the weights.

* batches are large to not allow examples of only one type within a single batch

*** wastes a lot of memory because it has to remember many gradients to then do back propagation at the end of the batch**

GPT FINE TUNING

Use specific datasets for a given task

GPT strengths

- Fluent in language and coherent
- general knowledge
- good for few shot and zero shot learning
- creative writing
- quick adaptation with fine tuning so I don't need a lot of data
- scalability for larger models in terms of performance (bigger, better performance).

GPT LIMITATIONS

- Lack of true reasoning (relies on patterns)
- It's hard to find a good prompt for the best answer.
- ethical and bias issues
- cannot reason or do complex calculations unless I give specific instructions in the prompt
- high computational requirements
- can't keep memory between sessions
- vulnerable to adverse prompts.

GPT Variants

- Codex→gpt3 model fine tuned for coding
- MT-NLG→nlp generation, specialized in summarization and few shot learning
- GLaM→uses less resources than gpt3 but with less performance using multiple models together

- PanGU-A→nlp in mandarin
- Chinchilla→training model
- OPT→built to support transparency in AI models by releasing weights, codes etc.
- BLOOM→NLP for different cultural contexts to strengthen inclusiveness

LLAMA

4 different models

LLAMA Input encoding

Relative positional encoding, the value that is added to the input embedding is not absolute but relative, for positional encoding, is based on relative positions rather than absolute ones.

Dataset "The Pile"

LLAMA PRE-TRAINING

Cross-entropy. Use SGD or ADAM, mixed precision: 32f, more complex float calculation.

Difference between LLAMA and GPT

We don't know anything about gpt training while we know a lot about llama. Llama is easier to use, and it's open while gpt is commercial.

EXAM: TEXT GENERATION

Encoder-Decoder transformers

Sequence to sequence tasks

T5

First LLM proposed for this task, although there are new ones

Each encoder block has 8 attention heads

Input encoding

SentencePiece, based on subwords.

Unigram Language Model, during training, selects the subwords that maximize the likelihood.

Ci sono vari special tokens: <pad> per allineare sequence nel batch, <unk> parole fuori dal dizionario, <eos>, <sep> ovvero specifici prefissi per task per indicare il task da fare.

T5 pre-training

Span-corruption→We put tokens in place of random words in the sequence. The model is trained to predict the missing tokens.

Generate tokens and learn context simultaneously.

Predicts SPAN instead of TOKEN to encourage learning global context, speaks fluently and coherently, and is more versatile across tasks.

C4 dataset, clean.

Cross-entropy loss

Adafactor optimizer which is memory efficient

Learning rate with warm up

Fine tuning

Prepare a dataset for the tasks of interest.

Ex: input is key word “summarize” <document>→output <summary>

These are also examples of possible prompts

mT5

multilingual T5, the dataset covers 101 languages

limitations: performance varies by language

FlanT5

Fine tuned of t5 with insturction tuning on various tasks, in zero shot and few shot it is better than T5.

Limitations: Needs precise wording to perform well on tasks.

ByT5

They tried to process text at the byte level

It works well for noisy or rare sentences.

Limitations: Very slow and computationally intensive

Usable when I need to generate character by character and not long texts

T5-3B and T5-11B

Many parameters

Ul2

Different learning paradigms, unidirectional, bidirectional and seq to seq.

Limitation: complex

Multi-Modal T5

Processes images and text, enabling tasks that combine them

Limitations: computationally complex

Efficient T5

T5 with less performance but faster: t5 smaal, distilT5 obtained with knowledge distillation.

Widely used to put them in small devices such as smartphones, they must be able to communicate with a CLOUD.

Project

I can use the exercise codes almost as they are

Justify each solution in the REPORT

The system is in ENGLISH

Check ROBUSTNESS (Questions like: ARE YOU SURE?) and OUT OF CONTEXT

I can use patterns like hugging face or llama

But I'll probably use a model only decoder

Fine Tuning

Full fine tuning→updates all model parameters, accurate but expensive and risks overfitting if I have small datasets

**** hugging face tutorial fine tuning on squad dataset see***

But we are not obliged to use the whole model, we have to adapt a pretrained llm to a specific task maybe only on some parts while keeping others done with self supervised pretraining.

I want to optimize performance on a small, precise dataset.

- **Parameter-efficient**fine tuning (**peft**):It only updates a subset of parameters, and thus fine-tuning is faster and requires less computational resources.
- **Instruction fine-tuning: (the one in flat t5)**:I train the model to return useful results with specific instructions , I can write a guide that says use these keywords or fill these slots. It is used to give a single LLM the ability to perform multiple tasks.
- **Reinforcement learning for human feedback (rlhf):**combines supervised learning with Reinforcement learning, reward the model when it generates correct answers .

Parameter efficient fine tuning – peft

It needs fewer parameters than standard methods.

Peft is used for large llm models that need updates in a short computational time

** peft library in hugging face*

It's great in situations where I have performance constraints or frequent updates.

Peft techniques

- **Low-Rank Adaptation (LoRA)**→ approximates weight updating by learning matrices
- **Adapters**→**These are modules that are added to the llm that keep the old weights fixed**
- **Prefix tuning**→I add new prefixes to the attention layer without touching the old pieces

LoRA

It assumes that the changes to be made reside in a lower-dimensional subspace.

A pre-trained transformer is represented by its weight matrix

A mathematical decomposition is made which is capable of obtaining two low rank matrices DW = AxB

A = mxr , B = rxn

r is the low rank which is smaller than mon

A and B are the minor matrices into which the major matrix W decomposes.

W = W + DW = W + AxB

It is proven that if I learn A and B I can update the weights of the model by adding only A and B which are smaller in size.

I learn A and B, small number of parameters.The inner dimension r is a hyper parameter

It makes a copy of the original weight matrix and cuts out A and B from that, then updates only this cut out part and adds it to W.

Fixed pre-training weights, only weights A and B are optimized for the new task,knowledge of pre-training is preserved .**But if A and B are not good the NEW KNOWLEDGE TAKES OVER and something is lost.**

AxB changes less than 1% of the old parameters (meaning that it only changes part of the old weights)

Efficient because the number of parameters that can be pulled is proportional to arx(m+n) which is much smaller than mxn

Compatible inference

I add the gradient in my weight matrix, which is not a layer of the network. Later, during inference I add them to the real ones and use them. Technically the real ones

they are still fixed because I could remove A and B.!!!!!!!!!!!!!!!!!!!

Adapters

We insert into the LLM new pieces that are light, task specific

These modules are trainable**while the weights of the old ones are fixed**

We add fully connected layer with nonlinear activation

After multihead we don't need to change the meaning but we need to change it before,**change the context and make them interpret correctly before attention, so that attention is calculated differently.**

I need to train only the fully connected layers

** some adapters are in pft library*

Prefix tuning

I add a prefix to each input token. It guides the model towards the specific task.

Guide llm's attention and output generation as it is added BEFORE THE ATTENTION.

The input sequence is preceded by a prefix.

The prefix influences attention by modifying kev

Only the prefix embeddings are optimized during fine tuning → there is a fully connected that comes concatenated with the normal layers, only the fully connected one is trained.

The higher it is, the more expressive it is but also the more expensive.

* are weights that are INSERTED above the original input matrix, I can choose up to m

So it's normal that I lose some of the pretraining knowledge

During fine-tuning only the FULLY CONNECTED LAYER under the SOFT PROMPT is trained.

They are placed before attention because attention is also calculated with respect to them (they influence kev).

Fine tuning instructions

** it's not that efficient*

Adapt the model to better understand and respond to the user

I train the model on task specific prompts

I can train him with this format:

- **Instruction ("Summarize the following text in one sentence")**
- **context**(you can help to guide the llm to give a better answer)
- **output**(expected response)

During training the model is trained**to recognize intent in instruction and generate consistent, accurate outputs etc.**

I have to have a**dataset**specialized on tasks (**but as general as possible, in the sense that it must have tasks from various domains**) which is composed of prompts for a specific task and the related responses.

The goal is to fully exploit the model's ability to generalize and align it to instructions human to make it more usable.

STUDY PFT LIBRARIES (for now) AND TLR

EXAM: I COULD DO BOTH FINE TUNE AND RAG

SCRIPTS COULD HELP YOU UNDERSTAND HOW TO HANDLE CONTEXT AND INPUT

Prompt engineering

Optimize prompts to use large language models.

Goals:

- understand the possibilities and limitations of LLMs
- improve model performance on various tasks
- allows you to interface with LLMs and integrate it with various tools
- use external knowledge to enable the LLM to reason about these resources without to retrain.

Writing good prompts

- Start with simple prompts and make prompts that gradually add things to get better results .
- Use specific instructions at the BEGINNING OF THE PROMPT.
- Be detailed and descriptive for better output but not too detailed
- Do not include excessive information, llm has limited attention windows
- Use examples to drive the output, tune the behavior.

Elements of a prompt

- **Instructions**→what do we want from the model
- **Context**→external information or explanations to ensure a better response ,**It's very important.**
- **Data input**→input or question I want an answer to
- **Output indicator**→format I want the output in – ex: Sentiment: (puts the output)

EXAM: I CAN TELL HIM NOT TO ANSWER IF HE FINDS OUT THAT IT DOESN'T BELONG TO NLP, OR AS A CONTEXT I CAN GIVE HIM THE SLIDES AND HE WILL ANSWER ONLY BASED ON THE SLIDES.

In-Context Learning

Ability of an LLM to perform a task by interpreting the prompt using information of context given in the prompt not updating its parameters.

In the prompt I can specify: reference material, task examples to illustrate the desired pattern, I can specify step by step instructions, clarify ambiguous parts or templates to put the output in.

Prompt engineering relies heavily on**CONTEXT LEARNING**→**I DON'T NEED TO FINE TUNE THE MODEL FOR ALL TASKS.**

LLMs are not yet very skilled in REASONING, they use particular techniques to obtain better results.

System Prompts

I give prompts once to guide subsequent interactions.

Subsequent responses will be influenced by this behavior.

"You are a helpful and knowledgeable assistans.."

Prompt engineering techniques

Zero shot prompting→prompt without including examples , uses its information

Few-shot prompting→I give some examples to guide the model in a better response. EX: The model understands that I am trying to use a dummy word and responds correctly .

Limitations: it is limited when there is complex reasoning, it fails neither with fewshot nor with zero-shot. I need more prompt engineering techniques.

PROMPTING TECHNIQUES TO AID REASONING

Chain of thought prompting→It allows for complex reasoning by specifying the intermediate steps that lead from the input to the output.

I can combine it with few-shot prompting to get better results even on complex tasks.

** this is a skill that no one expected an LLM to have*

Self-consistency prompting→use an iterative chain of thought, I give the same prompt several times, consider the different answers and take the most frequent answer.

Meta prompting→ I guide the model through the logical steps to solve the problem, not concretely but in an abstract way (the context is not a specific example but a general procedure). I explain the process by specifying for example the formulas without results. **THE CONTEXT IS THE PROCESS EXPLAINED IN THIS CASE.**

TASK AGNOSTIC META-PROMPTING→It has been shown that by adding the word "think step-by-step" I force the model to think of a procedure and therefore I get new results. It identifies the steps and then responds. I have to say**"start the answer with lets think step by step". Then I can also better define the structure of the answer I want.**

Meta meta prompting→I ask the model to generate a prompt for a particular problem, then I use that prompt to answer my problem. I can say "generate a step-by-step procedure" then use that in the next prompt.

** IT'S AN EMPIRICAL FIELD, things are discovered experimentally*

Prompt chaining→I use multiple prompts to get the result I want, I divide the task into specific parts with the specific prompt. I take the result of the first one and put it as the result of the second prompt until I reach the final result.

Role Prompting→**I ask the model to impersonate a specific role while responding, usually improving accuracy because the tone, style and depth of information changes accordingly. "You are a food critic".**

Structured prompting→I split the prompt into parts, and use DELIMITERS ###, ===, >>> to delimit the parts. I can also use XML tags as delimiters because usually llms are trained on web content <classes> </classes>.

One of the frameworks used to do structured prompting is**COSTAR FRAMEWORK**, divides the prompting in different sections:

- context
- Objective
- Style
- Tone
- Audience, the public to whom the answer must be addressed
- response , response format

Generate knowledge prompts→first use the llm to generate task-related knowledge and then incorporate that into the final answer prompt.

- Part 1→generates particular knowledge
- 2nd part→use what you generated to respond

Generates knowledge not included in training

Retrieval Augmented Generation – RAG

Combine LLM with search engines, overcome LLM limitations by accessing up-to-date documents or specific data (e.g. updated after training, also no need to change weights)

I use the query to find this information in the search engine and then use it to answer.

Useful because I can also use it WITH PRIVATE DOCUMENTS to build THE CONTEXT DYNAMICALLY, AT RUNTIME.

It is done in 2 steps.

- 1st step→I have documents, like books
- 2nd step→**I divide the documents into chunks, I have to decide how big they are,the chunk embedding is generated and are stored in a vector DB.**

When I do prompt first I generate the embedding, and put the chunk as context and it gives me similar answers.

** much requested because it responds**based on PRIVATE DATA**, is the future of CHATBOTs*

EXAMINATION

There are many other prompting techniques...

Prompt testing

I need to test the prompt before using it in my application→ PROMPT TESTING TOOLS, does various attempts to find the best structure and format of the prompt (depending on the model)

I can change various parameters such as temperature to control the output style, tone or precision.

OpenAI playground, google ai studio, lm studio

LLM Settings

By switching to a model I can change various parameters while designing a prompt like the temperature or**the top p**which adjusts the diversity of the answers by limiting the choice of tokens with a certain threshold. (As the next token is chosen according to this threshold, the tokens that can be the next word added must make a maximum of this threshold, instead the top k takes the first k and that's enough).

Temperature instead increases the basic probability of words. If t is low, 1/t is high so the prob of the first value becomes even higher.

** it is used because sometimes you pay for each token generated*

- **Stop sequences:**if this token is generated, generation stops
- **Frequency of penalty:**reduces repetitions by penalizing the model every time it repeats a word
- **Presence penalty**:penalizes the model EVEN IF THE WORD IS REPEATED ONLY ONE TIME.
- **Response format**:how do i expect the output.

LM STUDIO

I have access to every hugging face model, I can also use it as a model server.

Retrieval Augmented Generation (RAG)

LLm have knowledge limited only to the data used during training,They cannot access new information introduced after training, they cannot reason about private or proprietary data.

A RAG application has 2 main components:**indexing and retrieval**

Indexing:a pipeline to take data from a source and index it, it is done offline

Retrieval and generation: **takes the user query, retrieves the relevant data from the index, generates a prompt that contains the query and the retrieved data as context, sends this prompt to an LLM to generate a response.**

Indexing

LOAD:First of all we need to load the data , we can upload pdf,csv,html.

SPLIT:**I divide documents into small chunks**, does not use the whole document because the answer is usually contained in a part of the document,**Usually these chunks overlap to avoid information being spread across multiple chunks.** .

** The attention window of an LLM is finite so putting large documents may produce unimportant information.*

AFTER THE SPLIT I AN EMBEDDING OF THE CHUNKS IS GENERATED.

STORE:we store each chunk and index in a search engine,**vector store**

VECTOR STORES

Documents that talk about the same topic are placed next to each other based on their embeddings, **SEARCH BY SIMILARITY**

RETRIEVAL AND GENERATION

Relevant information is retrieved, a prompt is generated, passed to the network**and it comes produced the response.**

Prompt: Based on this {retrieved chunk} what is word embedding?

INTRODUCTION TO LANGCHAIN

Framework that simplifies the development of applications using llm.

These are building blocks for incorporating LLMs into applications.

I can connect third party LLMs, data sources and external tools.

Various use cases like chatbots, rag, document search..

- **Prompt templates**:facilitates the translation of user input into llm instructions
- **LLMs**: llm models that take and return strings
- **Chat models**:**keeps the history of the conversation**
- **Example selector**:select and create concrete examples to put in the prompt and improve performance.
- **OutputParser**: transform text into structure templates like csv
- **Document loaders**: upload documents from various data sources
- **Vector stories**: system for storing and retrieving documents
- **Retirees**: Document and data recovery interface
- **Agents**:*llm based system to reason and decide action based on input. It is based on previous steps.*

TRYING OUT VARIOUS TEXT SPLITTERS

EXERCISE

I can use LCEL to create modular pipelines,**also includes ChainPredefined for QUESTION ANSWERING for example.**

 # I COULD USE A CHAT MODEL IN THE PROJECT THAT KEEPS MEMORY OF THE CONVERSATION

AND USE PREVIOUS ANSWERS TO REFINE

Reinforcement learning for human feedback

The model responds, I give a score and I get a new dataset made up of questions, answers and score.

We could tow a**classifier**that given the question is able to give the score dragged with this dataset:**REWARD MODEL (scores are classes)**

Or I can give it to you constantly : I add a new sample to the REWARD MODEL

We use reinforcement and not supervised fine tuning because we don't have ground-truth ,**I use one policy.**

Instead of the reward function I use a score from the user.

Instead of always asking the user for a reward,**if I have it I use it otherwise I use the reward model driven with the scores already made to do CONTINUOUS LEARNING.**

Then the LLM loss is also influenced by the weight score which updates the weights in the direction of the best responses.

EXAM: OPTIMIZE THE PROMPT

It is a strategy to balance the model's performance and align it with human preferences.

Using RLHF can be helpful in concentrating the focus of the LLM and improving the safety and the rejection of some incorrect terms.

Pipeline:

1. pre-train a model with self supervised learning, I can consider this data LOW QUALITY DATA
2. **Decoder only model to obtain a pretrained llm**
3. Then I can do a supervised finetuing with less examples, I have HIGH QUALITY DATA or concentrate on the task (IN THIS CASE DIALOGUE, general question answering)
4. I can use this data to train a REWARD MODEL, plus finetuning data. I can also use a reward model with the dataset already used for finetuning giving scores.
5. The reward model is used to reinforce the model's learning to take the prompt, reward it with the reward model and get the final model.

*** EXAM: USE BERT, GPT, T5 ALREADY SPECIALIZED IN QUESTION ANSWERING AND WITH KNOWLEDGE OF NLP**

Use a secondary model reward model or human score.

Do reinforcement learning

Reward model

Input: question+answer, score

Train a model to predict the score

PPO proximal policy optimization

1. I generate answers using llm
2. I give a score
3. **Update llm to maximize score**

Pros:

- Iterative improvements to improve both the model and the reward model
- **I can start from an initial model and align it to my preferences.**
- I can use it to get epic responses
- **Update it per user preferences**

Against:

- subjectivity
- Scalability,**Collecting feedback is resource-intensive, how do we know if the feedback is right?**
- If we use a non-robust reward model it can lead to non-optimal fine tuning

It can be applied to all tasks:

- **text generation**,generate a better response by changing the style for example
- **dialogue systems**
- **language translation**,improve translation accuracy
- **summarization**
- **question answering**
- **sentiment analysis**,maybe zero-shot is not good and I can do it with reinforcement
- **computer programming**

GPT-3.5 and GPT 4

They were also fine-tuned with RLHF

They reported getting fewer unsafe outputs and more user-like interactions. human ones.

Transformers trl library

He's in hugging face

SFT Trainer can be used to train it on my dataset.

RewardTrainer, question+answer and score

PPOTrainer, improve sft model using reward from reward model and ppo algorithm

We generate a response to maximize the score

- **LOOK CAREFULLY AT PPO TRAINER AND REWARDTRAINER**
- **LOOK WELL DETOXFING LLM WITH PPO.**

EXAM: ALSO GIVE CODE EXAMPLES AS INPUT

Guardrails for LLMs

They are mechanisms or policies that regulate the behavior of LLMs.

They ensure that responses are confident, accurate, and appropriate to the context.

Ex:

block dangerous content

restrict output to specific domains

Types of guardrails

- Safety guardrails
- Domain-specific guardrails
- Ethical guardrails: eliminate bias, misinformation and ensure parity
- Operational Guardrails: Limit output to align with business or user goals.

5 main techniques

- **Rule-based filters** , specific rules that allow us to ensure a certain result
- **Fine tuning** with custom data
- **Prompt engineering** : if I automatically change the prompt from the one given in user input, I can use some sentences to guide the network to some answers
- **External validation layers** : output as is but there is another module to determine if the output is correct, post processing
- **Real-time monitoring and feedback** : reinforcement learning, to make the network evolve in the direction of the user's needs

Rule-based filters

Keyword blocking, regex pattern for sensitivity filtering

I have a dictionary of forbidden words and I add rules that if the output contains these words I do not return, I replace them or I censor.

Difficult to use for content filtering

Fine-tuning with custom data

I train the model on a specific domain dataset.

I work on weights, not output. I need a dataset.

Prompt engineering

I modify the LLM input, I force the LLM to generate an output that respects certain limits

** a network may answer wrongly because there are too few samples on that topic in the training dataset, this often happens for new topics*

External validation layers

Here I work on output, additional systems that do toxicity detection and fact-checking

Allows for modular and scalable implementation of guardrails

But there are more modules in the pipeline

Real time monitoring and feedback

In many applications I can use human feedback

THE TYPICAL BEST PRACTICE IS TO COMBINE MULTIPLE TECHNIQUES TOGETHER

Rule based filter to remove certain words, then I use an external validation but first I have to do a fine tuning to guide the model only on the topics of interest.

EXAM: WORKING ON HOW TO OPTIMIZE THE PROMPT

** I usually learn what is the best prompt to give in input to get the best response with reinforcement*

Guardrails AI (Hugging Face), LangChain, OpenAI Moderation (Validation Layer)

There are predefined and customizable guardrails on the ruleset

AI Guardrails

It allows you to validate against specific guidelines, you can help format the output structure or filters to remove or block unprotected content.

Langchain

I can decorate the prompt

I can merge tools in langchain and in Guardrails AI

Proceed incrementally, add complexity if I don't get there with the simple ones
my goal.

We can choose various models, do some tests and say:

“this one has these problems, this one has this other problem” and then choose the best one