# Natural language processing: Summarization

# The problem: Summarization

- Let us say you have a long newspaper article and you want to get a short version of it, of a paragraph or two, which still captures the main points of the article.
- This is a natural language processing task called summarization.
- Differs from the sentiment analysis task:
    - sentiment analysis: input is a text, output is a label (positive, negative, neutral)
    - summarization: input is a text, output is a shorter text

# What makes it a summarization problem?

- **Inputs**:
  - A long text (e.g., a newspaper article)
- **Outputs**:
  - A short text (e.g., a summary of the article)

# How do I know that it works?

- This is a difficult question. How do you know that the summary is good?
  - In contrast to the sentiment analysis task, where you can easily check if the output label is correct or not, in summarization you need to check if the output text is a good summary of the input text.
  - There is no single correct answer. Different people might write different summaries of the same article, and all of them might be good.
- The gold standard for evaluating summarization is to have a human read the input text and the output summary and judge if the summary captures the main points of the input text.
  - Very expensive and time consuming.

# Automatic evaluation of summarization

- There are some automatic metrics for evaluating summarization, such as ROUGE (Recall-Oriented Understudy for Gisting Evaluation).
  - ROUGE compares the output summary with a reference summary (a human-written summary) and calculates the overlap of n-grams between the two summaries.
  - The higher the overlap, the better the summary is.
- These days: if you have access to a very good summarization model, you can use it to:
  - generate a reference summary for the input text, and then use ROUGE to evaluate the output summary against the reference summary.
  - or just as an LLM to evaluate the output summary

# How it works: let's try to hack together a summarization model

- How did people do summarization before the advent of large language models?
- The first idea: take the first few sentences of the input text and use them as the summary.
    - This is called an extractive summarization approach. It works reasonably well for some types of texts, such as news articles, where the main points are often summarized in the first few sentences.
    - But it does not work well for other types of texts, such as scientific papers, where the main points are often summarized in the last few sentences.
- A more sophisticated approach: score the sentences in the input text based on their importance and then select the top few sentences as the summary.

# How it works: large language models

- Human language has some regularities:
  - for instance, some words are often followed by other words, such as "the" is often followed by a noun, and "is" is often followed by an adjective or a noun.
  - notice the "often": these are not hard rules, but probabilities
- A **language model** captures these probabilities.

# A simple language model: n-grams

- An n-gram is a sequence of n words. For instance, a bigram is a sequence of 2 words, and a trigram is a sequence of 3 words.
- An n-gram language model estimates the probability of a word given the previous n-1 words. For instance, a bigram model estimates the probability of a word given the previous word, and a trigram model estimates the probability of a word given the previous two words.
    - You can build an bigram model by counting the number of times each bigram appears in a text.
- The n-1 previous words are called the **context**. The language model estimates the probability of the next word given the context.

# What can you do with a bigram model?

- You can use it to predict the next word in a sentence:
  - Typing suggestions in your phone or in an email client.
- You can use it for spelling correction:
  - If you type "teh", the model can suggest "the" as a correction, because "the" is a more common bigram than "teh".

# Large language models

- A large language model (LLM) is a language model which:
  - is larger and more sophisticated
  - is trained on a large amount of text data (e.g., the entire internet)
  - it can handle a much larger context (e.g., the entire article) and can generate a much longer output (e.g., a summary of the article)

# How to build a large language model?

- Is it complicated?
  - The code is not **that** trivial, but it is not very long.
  - Andrej Karpathy's video "Let's reproduce GPT-2" is 4 hours long.
  - The code is about 500 lines long, a single file and it is available on GitHub:
    - https://github.com/karpathy/build-nanogpt
- The critical idea: the architecture is relatively simple
  - The power of the LLM comes from the training data and the training procedure

# The key idea: the transformer architecture

- The transformer architecture is a neural network architecture that was introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017.
- It is based on the idea of self-attention, which allows the model to attend to different parts of the input text when generating the output text.
- The transformer architecture is the basis for almost all large language models

# Self-attention

- Let us say you have an input text "The brave chicken swiftly crossed the dangerous road."
    - We call this the **context**.
- Now you want to reason about this text
    - Generate a summary
    - Answer questions etc.
- If we are asking "What is the subject of the sentence?", we need to attend to the "chicken" or maybe "brave chicken"
- The transformer architecture performs a mathematical operation that creates a map that would put a high value on the "chicken" and a low value on the other words in the context.

# Multi-head self-attention

- For some tasks, you need to attend to more than one word in the context.
- So we can just use multiple attention heads, each focusing on a different part of the context.
- And we stack them on top of each other, so that the model can attend to different parts of the context at different layers of the network.

# The architecture of a GPT-3 models

| Model | Parameters | Transformer Layers | Attention Heads | Hidden Size |
|---|---|---|---|---|
| Small | 125M | 12 | 12 | 768 |
| Medium | 350M | 24 | 16 | 1024 |
| Large | 760M | 24 | 16 | 1536 |
| XL | 1.3B | 24 | 32 | 2048 |
| 2.7B | 32 | 32 | 2560 | |
| 6.7B | 32 | 32 | 4096 | |
| 13B | 40 | 40 | 5120 | |
| **175B (largest)** | **96 layers** | **96 heads** | **12288** | |

# How it works: pre-training an LLM

- An LLM is trained to predict the next word in a sentence given the previous words (the context) - prediction task
- Sometimes, it is trained to fill in a blank in a sentence given the previous and the following words (the cloze task)
- What do we train on?
  - A large amount of text data, such as the entire internet, books, articles, etc.
  - Eg. Wikipedia: 6 million articles, 3.5 billion words, mostly well chosen, good quality text
  - Eg. Common Crawl: 60 billion web pages, 25 trillion words, very noisy, but a lot of data

# What can we do with a pre-trained LLM?

- After pre-training, the LLM can give me continuations of a text, such as:
  - "One a fine day "
    - "the brave chicken swiftly crossed the dangerous road."
    - "I went bungee jumping"
    - "my wallet was stolen"
- All of them are valid continuations of the text
  - At least, grammatically. I did **not** go bungee jumping, and my wallet was **not** stolen.
- Interesting, but not very useful
  - Maybe except for spelling correction, typing suggestions, etc.
  - It cannot be used as a chatbot!

# Following instructions: InstructGPT

- What is the continuations for:
  - "What is the capital of France?"
    - The capital of France is Paris
    - Funny you ask that
    - I would also like to know that
    - I don't know, but I can tell you a joke about France
- And we have a preference over the continuations: we want the model to give us the correct answer, not a joke about France.
- We want the model to follow instructions, not just generate continuations of a text.

# Fine-tuning an LLM to follow instructions

- We can fine-tune a pre-trained LLM on a dataset of the following type:
  - A number of questions, with multiple continuations for each question
  - A human annotator would rank the preferred continuations
- The model is then fine-tuned to generate the preferred continuations given the questions.
  - This is called supervised fine-tuning (SFT)
  - Had been originally introduced in a paper referred to as "InstructGPT"
  - The most frequently used method is called "reinforcement learning from human feedback" (RLHF)
    - Because it is using a reinforcement learning algorithm (PPO)
    - But other choices are possible

# Building a chatbot with an instruction tuned LLM

- The conversation with a chatbot is a sequence of questions and answers.
- All these go into the **context** of the LLM
- This is an instruction tuned LLM, so it will try to follow the instructions in the context and generate a response that is appropriate for the conversation.

# Summarization with an LLM

- We can use an instruction tuned LLM to generate a summary of a long text.
- Typical process:
  - We provide the LLM with a prompt that includes instructions on how to summarize the text
  - We provide the LLM with the long text as part of the context
  - The LLM generates a summary of the long text based on the instructions and the context.
  - We might add constraints on the summary, such as a maximum length, or specific points that should be included in the summary.

# Possible problems: context size

- The size of the context: how long can the input text be?
  - The size of the text is limited by the size of the context that the LLM can handle.
  - For instance:

    -GPT-3: 2048 tokens, which is about 1500 words.
    - GPT-3.5 Turbo: 16,385 tokens
    - GPT-5: 400,000 tokens in the API
    - Anthropic's Claude: 200k tokens in the API
    - Google Gemini 3.1 Pro: 1M tokens

# Possible problems: hallucinations

- LLMs can generate information that is not true or not supported by the input text.
- This is known as "hallucination" in the context of LLMs.
- Hallucinations can be problematic, especially in applications where accuracy is critical.

# Two kinds of knowledge in LLMs

- LLMs stores knowledge in two ways:
  - In the parameters of the model, which are learned during pre-training. This is called "parametric knowledge".
  - In the context, which is provided as input to the model. This is called "contextual knowledge".
- When we ask the model to summarize a long text, we want it to use the contextual knowledge from the input text, not the parametric knowledge from the pre-training.
- However, the model might use the parametric knowledge to fill in gaps in the input text, which can lead to hallucinations.

# Knowledge cutoff

- The parametric knowledge in the model is based on the data that was used for pre-training, which has a cutoff date. For instance, if the model was pre-trained on data up to June 2024, it will not have any knowledge of events that happened after that date.
- This can lead to hallucinations if the model tries to generate information about events that happened after the knowledge cutoff date, based on its parametric knowledge.

# Example with setting an alternative word

- Set the world of The Man In the High Castle, a novel by Philip K. Dick.
- Ask questions about the President of US, President of Russia etc.
- Ask question about whether Mac's use Intel processors.
- An inconsistent word!

# Starting with an empty context

- If you start a new chat, the context is "empty", so the model will rely on its parametric knowledge to generate a summary.
- You can use this feature to avoid polluting the reasoning with things you said in the context.
- Be aware that the system already has some text in the context, the **system prompt**
  - Defines behavior, rules, or background information.
  - Example: “You are a helpful assistant...”
  - But it also can include other instructions or context that influence the model's behavior. For instance it might provide specific guidelines on how to respond to certain types of questions.

# Instructions about style, language and format

- You can ask an LLM to express their answers in a particular style:
  - pirate
  - villain from Shakespeare
- Some of these styles can be useful:
  - explain like I'm five (ELI5)
  - use simple language
  - use bullet points
  - use a language (e.g., English, Spanish, French)

# Pitfalls and dangers: Deep vs. shallow reading

- The opening lines of the Iliad, by Homer:

> "Sing, O goddess, the anger of Achilles son of Peleus, that brought countless ills upon the Achaeans. Many a brave soul did it send hurrying down to Hades, and many a hero did it yield a prey to dogs"

# What does deep reading gets us?

- The first important word is **anger** (often translated from the Greek mēnis). This is not ordinary irritation. In Greek literature, mēnis usually describes the **divine wrath of gods**, not the emotion of humans.
- A deep reading asks why this word is used for Achilles. It suggests that Achilles' rage is **almost godlike in scale**, capable of shaping the fate of entire armies. The poem immediately signals that the central subject is not the Trojan War itself but the destructive power of rage.
- And we are just halfway through the first line!

# Shallow reading

- Summarize Romeo and Juliet in one sentence:

> Two young lovers from feuding families fall in love and die tragically.

- Ok, then.
- But what about the themes of love, fate, family, and conflict? What about the poetic language and the dramatic structure of the play? What about the historical context in which it was written?

# Try it out: summarizing newspaper articles

- Summarizing newspaper articles
- ELI5 style
- Combine it with background information