Brief intro to Large Language Models.

Part 1: observations

Contents

- Recall the main points of LLMs on surface level
- Recall neural networks
- Transformers
- Using LLMs

Recall the main points of LLM on the surface level

LLM (Large Language Model / Foundation model) like GPT: what does it do?

Try to find the next word for the sentences:

They are leaving in about fifteen …
Dog ate my …
The Panama-registered tanker was carrying 136,000 tons of …
North Korea has agreed to send a delegation to next month's Winter Olympics in …
Officials from the two nations met …
2 + 2 = ...
cat => kass, dog => ..

What does it do? Predict tokens (letters, words) one at a time:

- John sat in the car and …
- John sat in the car and started …
- John sat in the car and started to …
- John sat in the car and started to drive …
- John sat in the car and started to drive home …

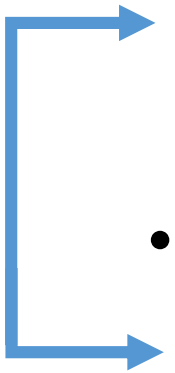
How to predict? A simplified idea:

- Collect statistics from a lot of texts: how did the sentences continue

  John sat in the car and ... *(started, thought, …)*

- When you see a sentence you have not seen before, use similar ones:

  Mike sat on a bike and …



Large LLMs?

GPT4 (March 2023) rumors / speculation (no official data given):

- 16 separate models (experts)
- Each expert has ca 111 billion params (ca 222 GB of memory?)
- Trained on ca 13 TB of text (ca 3 million large books)?
- OpenAI spent spent more than $100 million training

LLM like any neural network is an interpreted computer program

- Parameters (small floating point numbers) are the “code” .
- An interpreter program reads the code and produces results.
- Example interpreter programs: llama.cpp, mistral.rs, …

Differently from “conventional program”, neural networks are built by training on examples.

How can I run / use an LLM?

- Open / free ones like Llama from Meta:

  - On your own laptop (cpu: very slow; gpu: needed 6 GB … 200 GB).
  - Some Taltech server with a gpu with a large video memory.

- Large / proprietary like GPT or Claude: on their servers either via

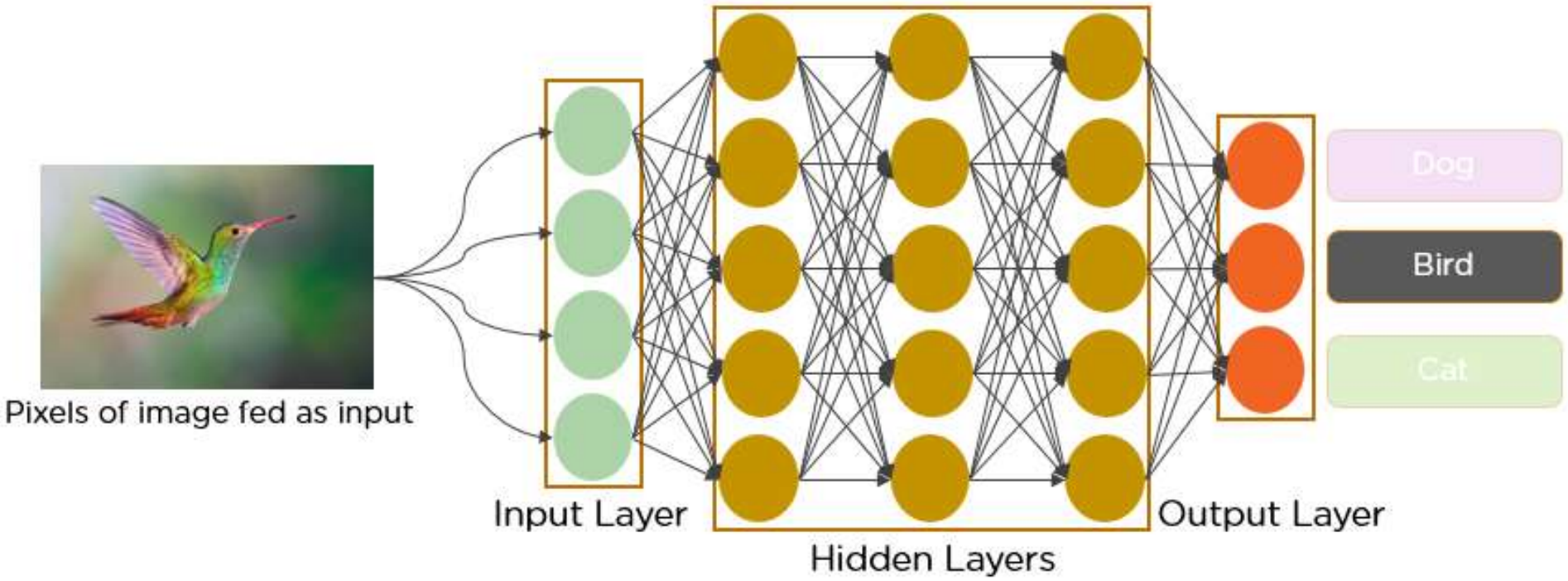
  - Web page for human use.
  - API for calling from command line or your own program.

“Large Language Model” aka LLM ?

- Given text (prompt), predicts next word, then next, then next, …
- Is a foundation model.
- Built using the transformer architecture.
- Large amounts of compute have been used for training
- Not necessarily “very large” (like, Phi-3-mini: 3.8 billion params, 4 bits per parameter, ca 1.8 GB of memory; trained on 3.3 TB of words / tokens)

Recall neural networks

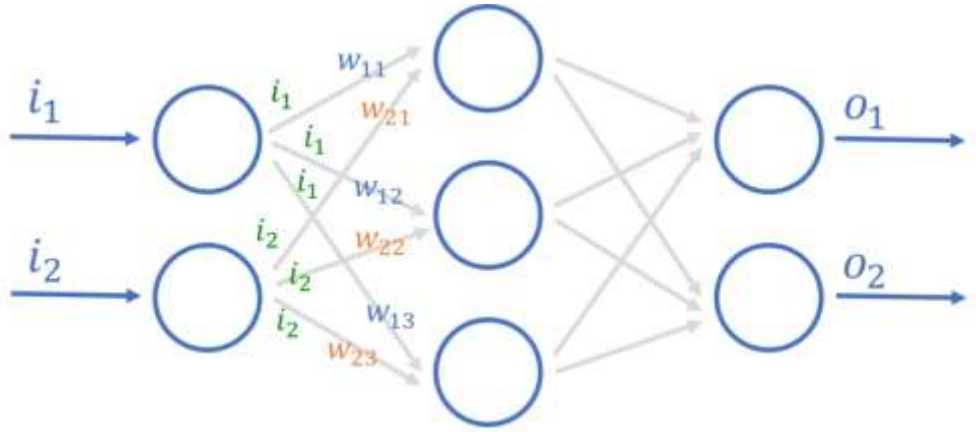
Deep neural networks is the most popular M.L. technique among many



Matrices and calculations

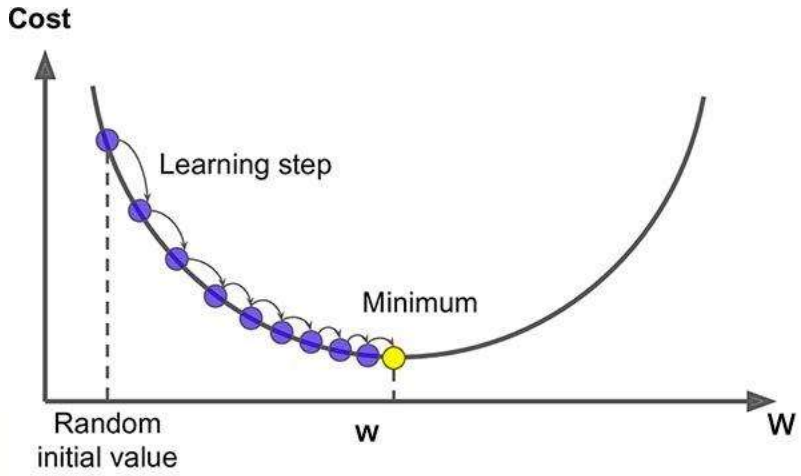
Connections are simply numbers (weights) you can store in a large table (matrix).

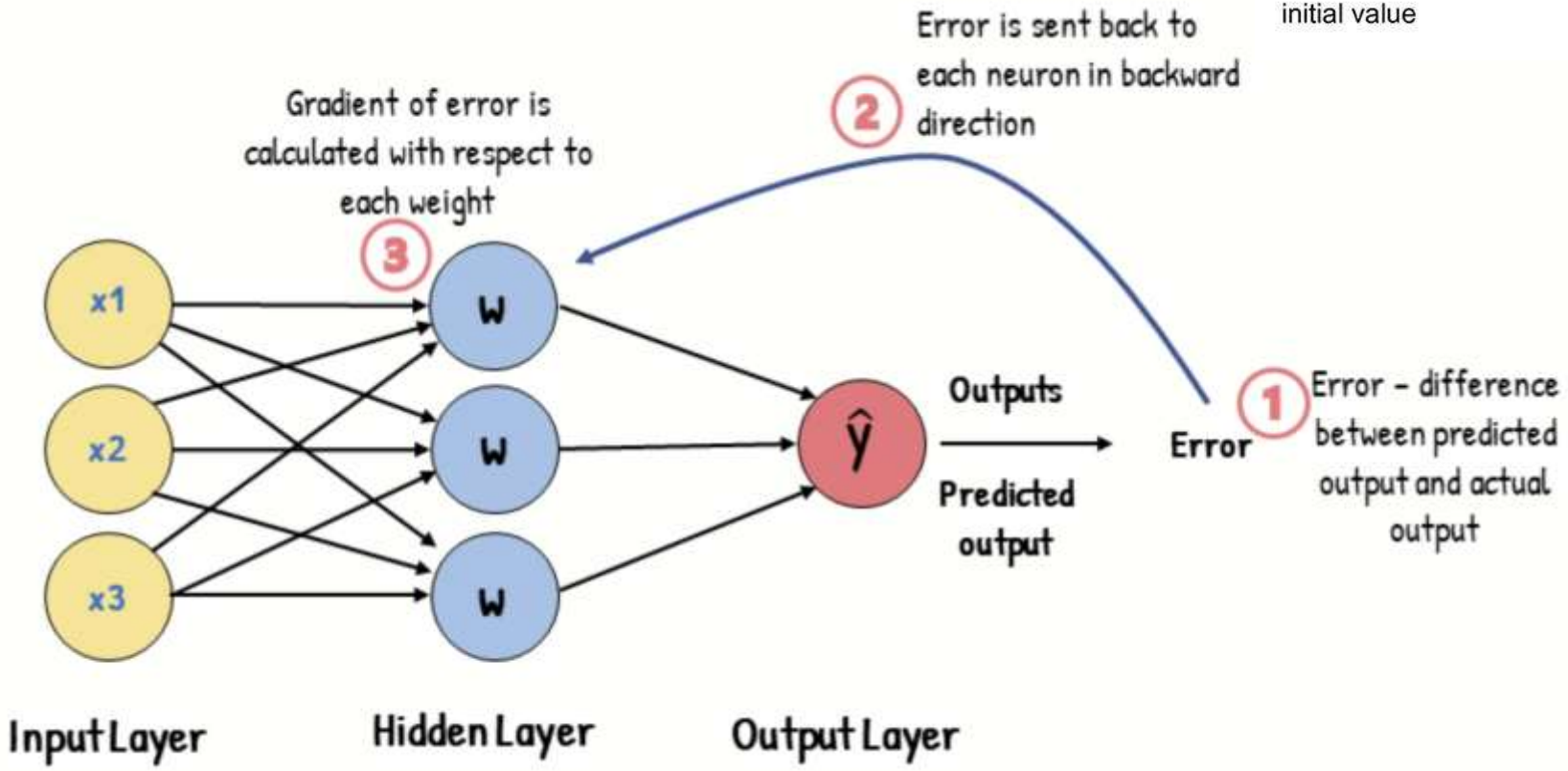
Calculating values for nodes can be done in several ways. One simple approach is to first multiply weights with inputs, sum all of them together and then normalize, using some popular simple function for this.



$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} (w_{11} \times i_1) + (w_{21} \times i_2) \\ (w_{12} \times i_1) + (w_{22} \times i_2) \\ (w_{13} \times i_1) + (w_{23} \times i_2) \end{bmatrix}$$

Learning by backpropagation and gradient descent





Stuff to read / look at

SLP3 slides about simple neural networks for text:

https://web.stanford.edu/~jurafsky/slp3/slides/7_NN_Apr_28_2021.pdf

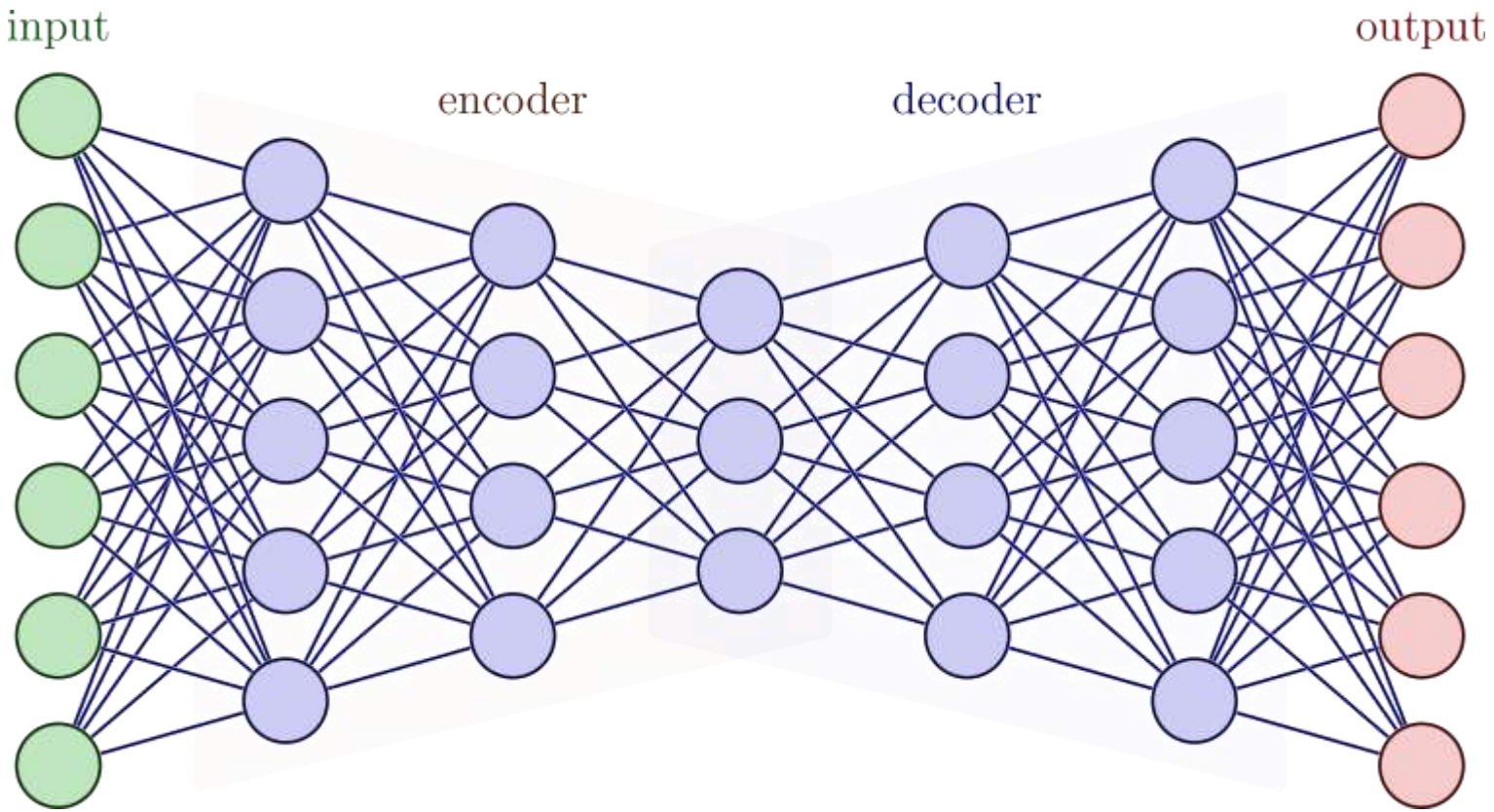
Gradient descent:

https://medium.com/data-science/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

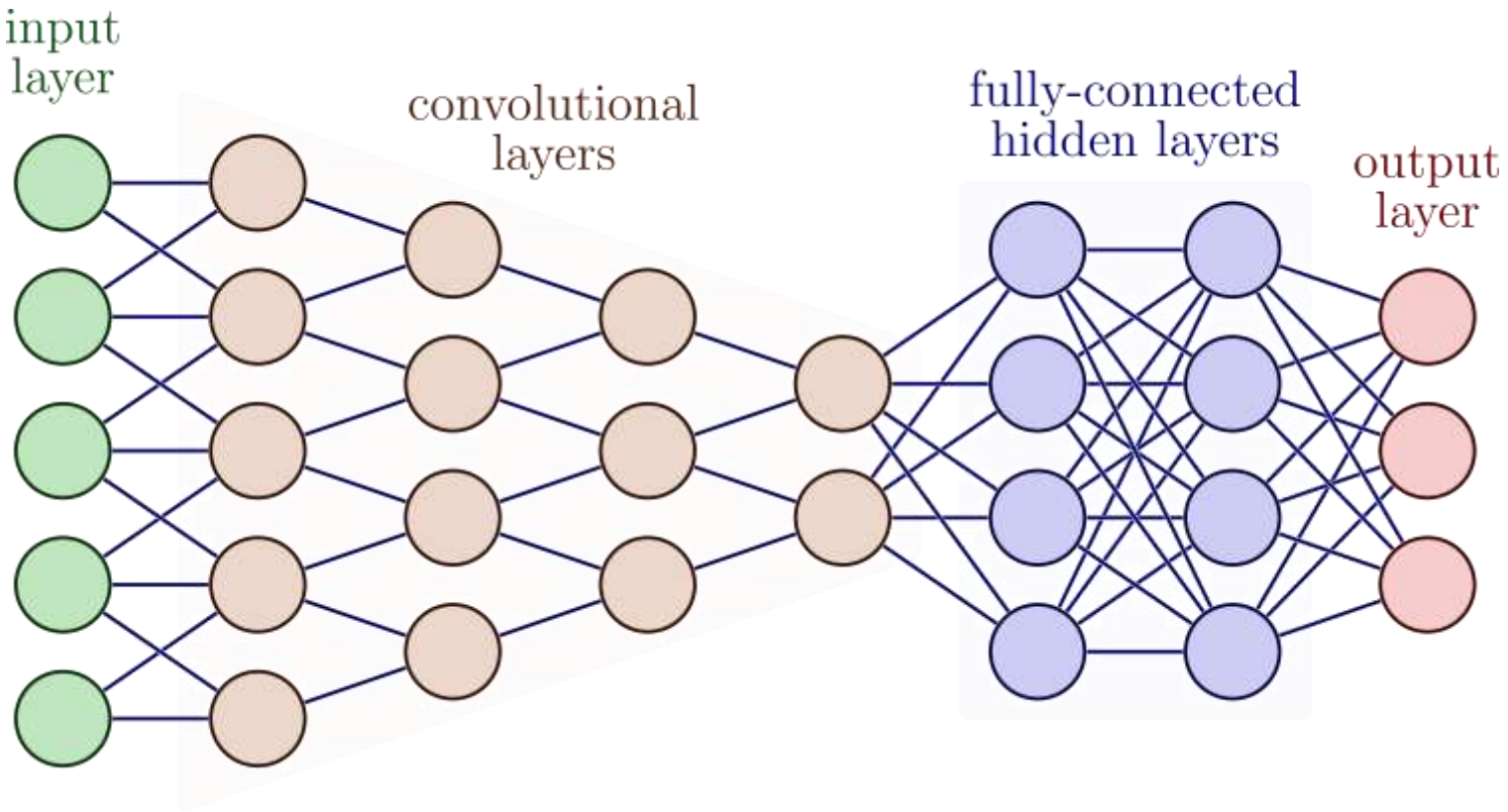
2003 paper of next word prediction with simple neural networks:

https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

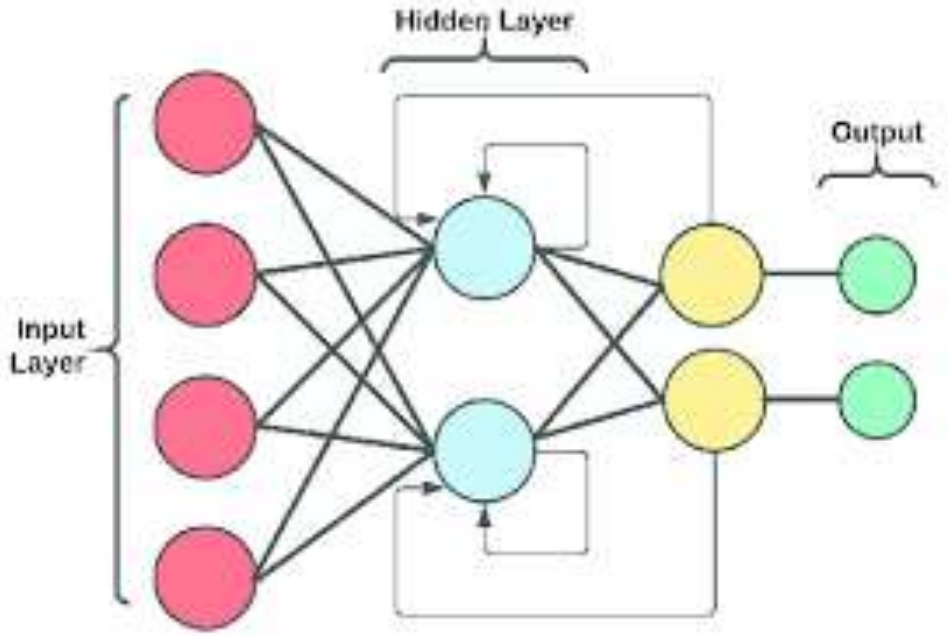
Autoencoder neural network (denoising, compression, generation)



Convolutional neural network (image recognition)



Recurrent neural network (time series, etc)



Transformers

Transformer (large language models aka LLMs like BERT / GPT / Llama)

Originally developed by Google in 2017 for machine translation. BERT model uses the encoder.

GPT uses only the decoder part.

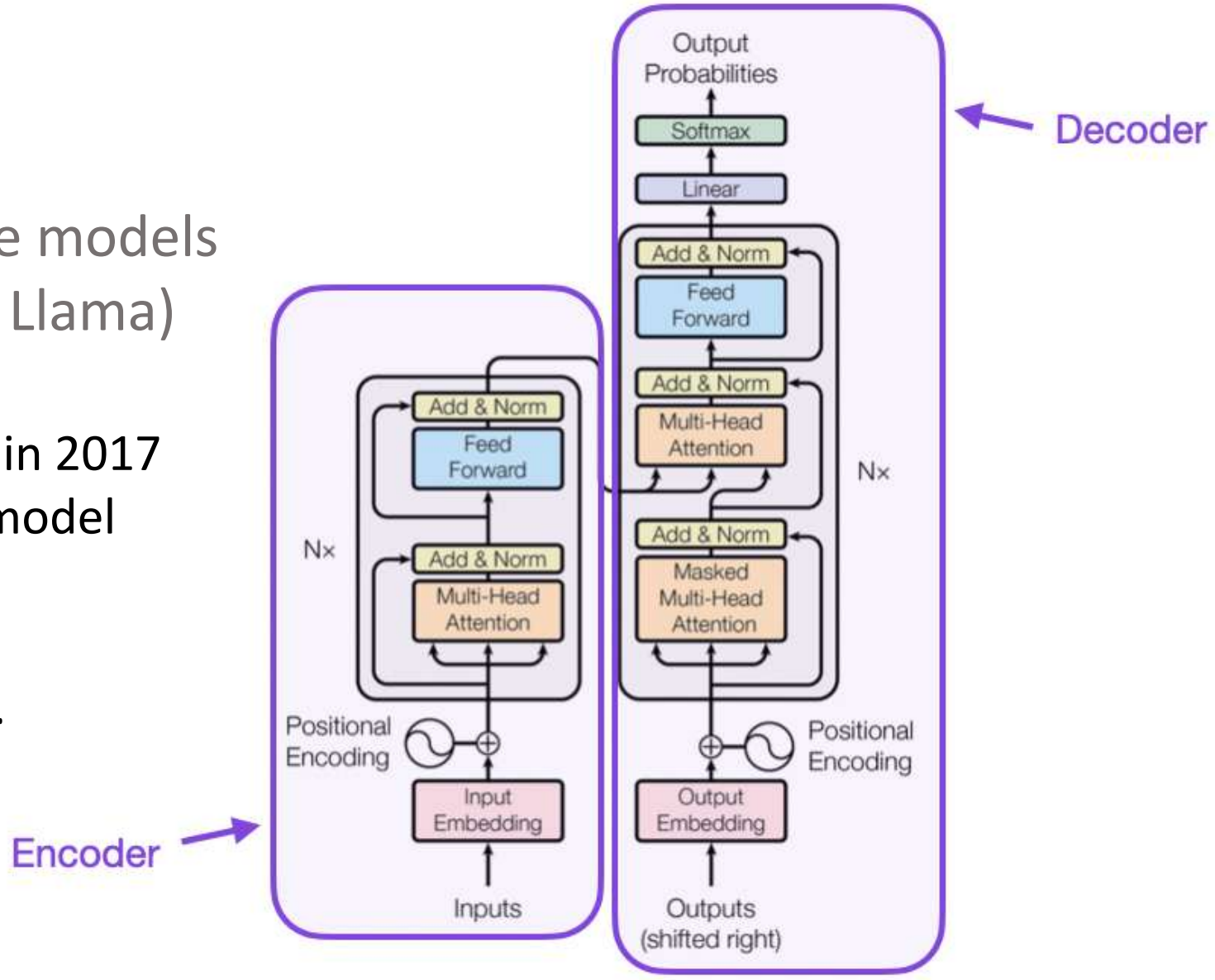


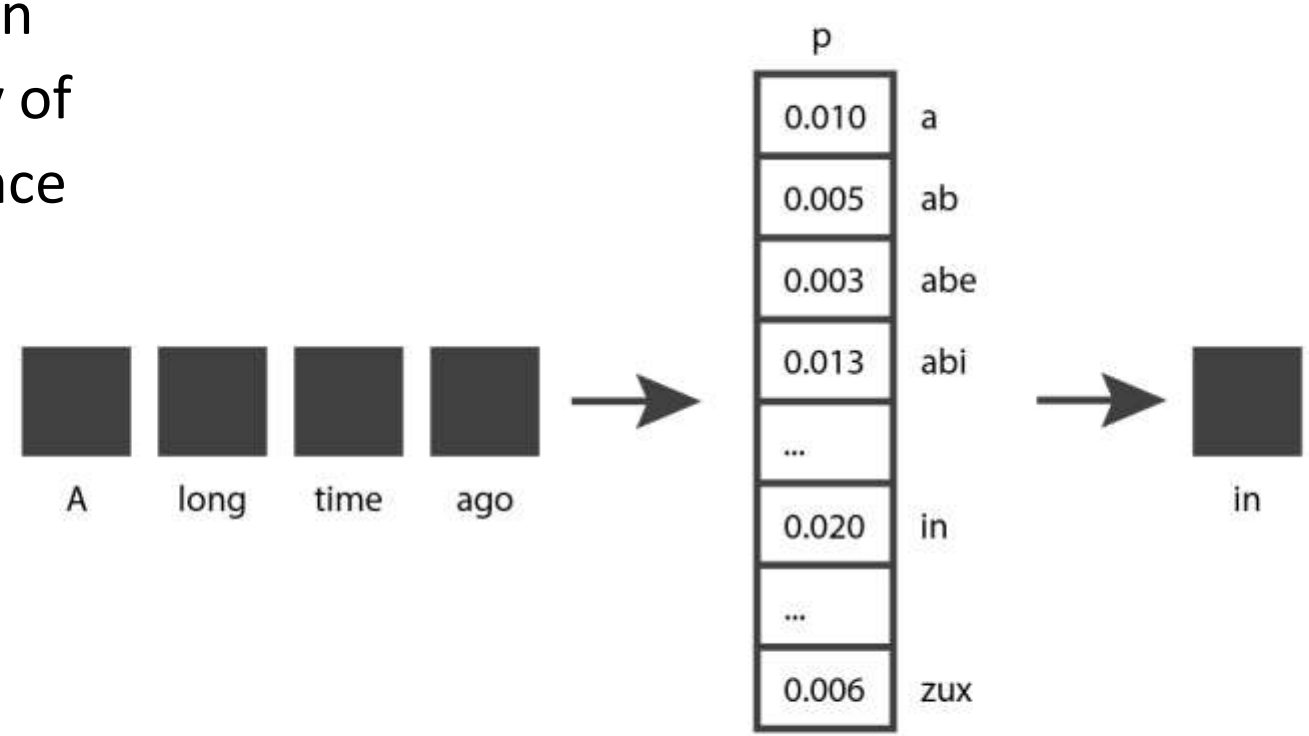
Figure 1: The Transformer - model architecture.

Known about GPT3 (predecessor of GPT4)

- Number of layers: 96
- Number of attention heads: 96
- Dimensions of its hidden layers: 12288
- Sequence length: 2048
- Number of parameters: 175 billion (ca 350 GB)

LLMs: predict next word / token

Each token / word has an approximate probability of being next in the sentence



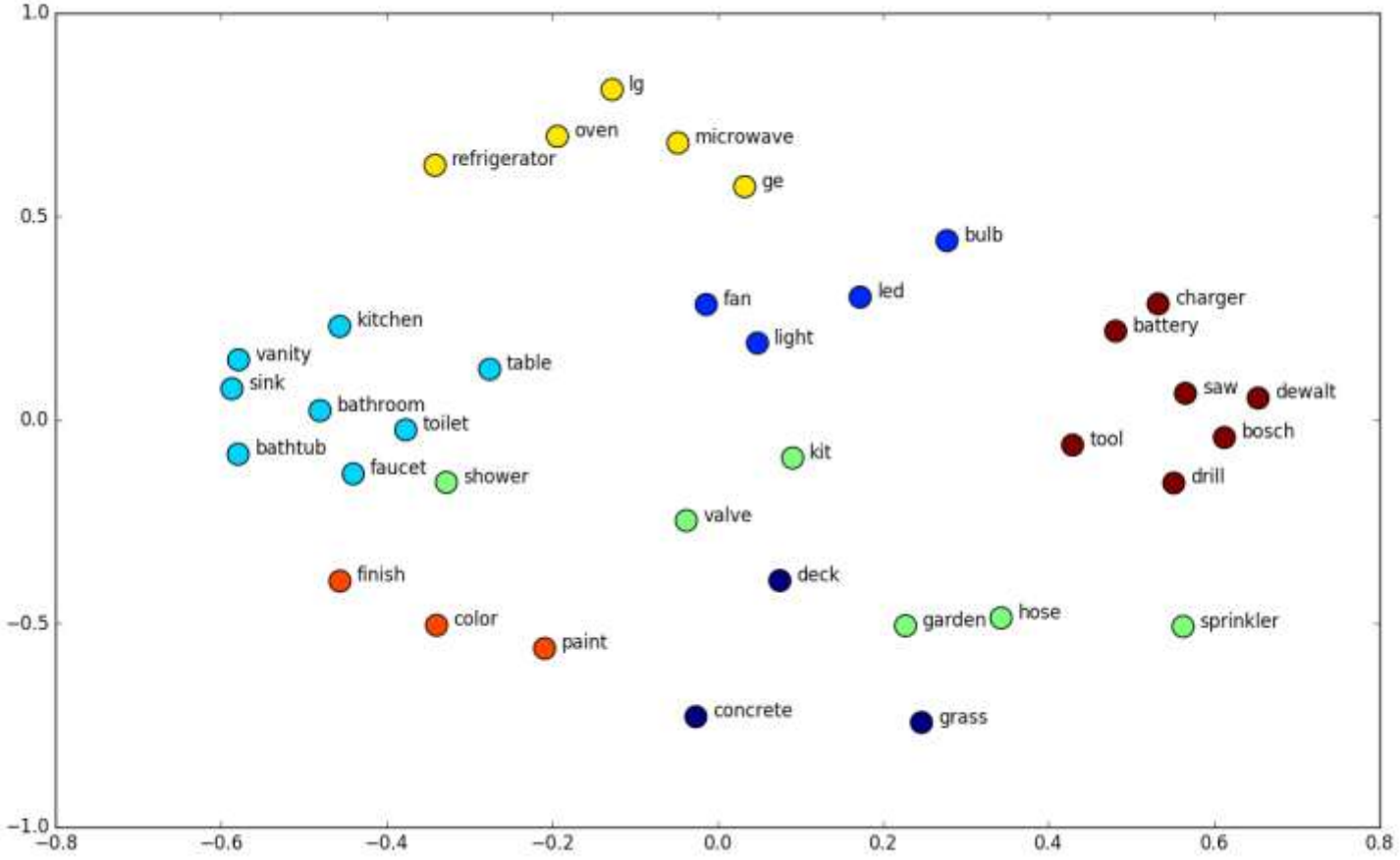
Word vectors (embeddings)

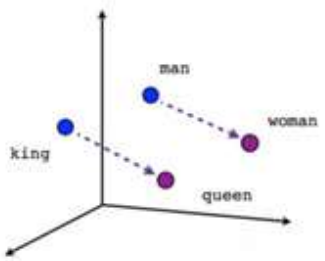
Each word or a phrase like “car”, “horse”, “jumped”, “fast runner”,
“weird colour of sky” is represented by a long list of numbers like

1, 99, 23, …., 85,12

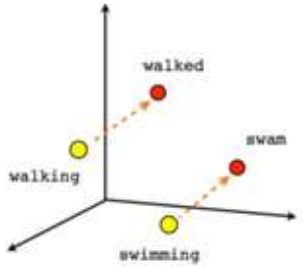
with each number indicating the “closeness” of the word/phrase to some specific concept or property.

GPT4 has ca 3000 numbers in a word vector.

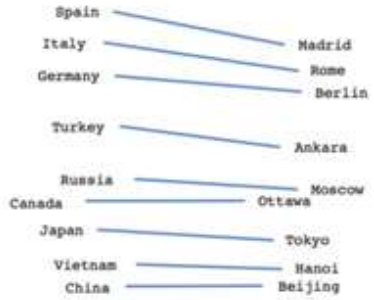




Male-Female

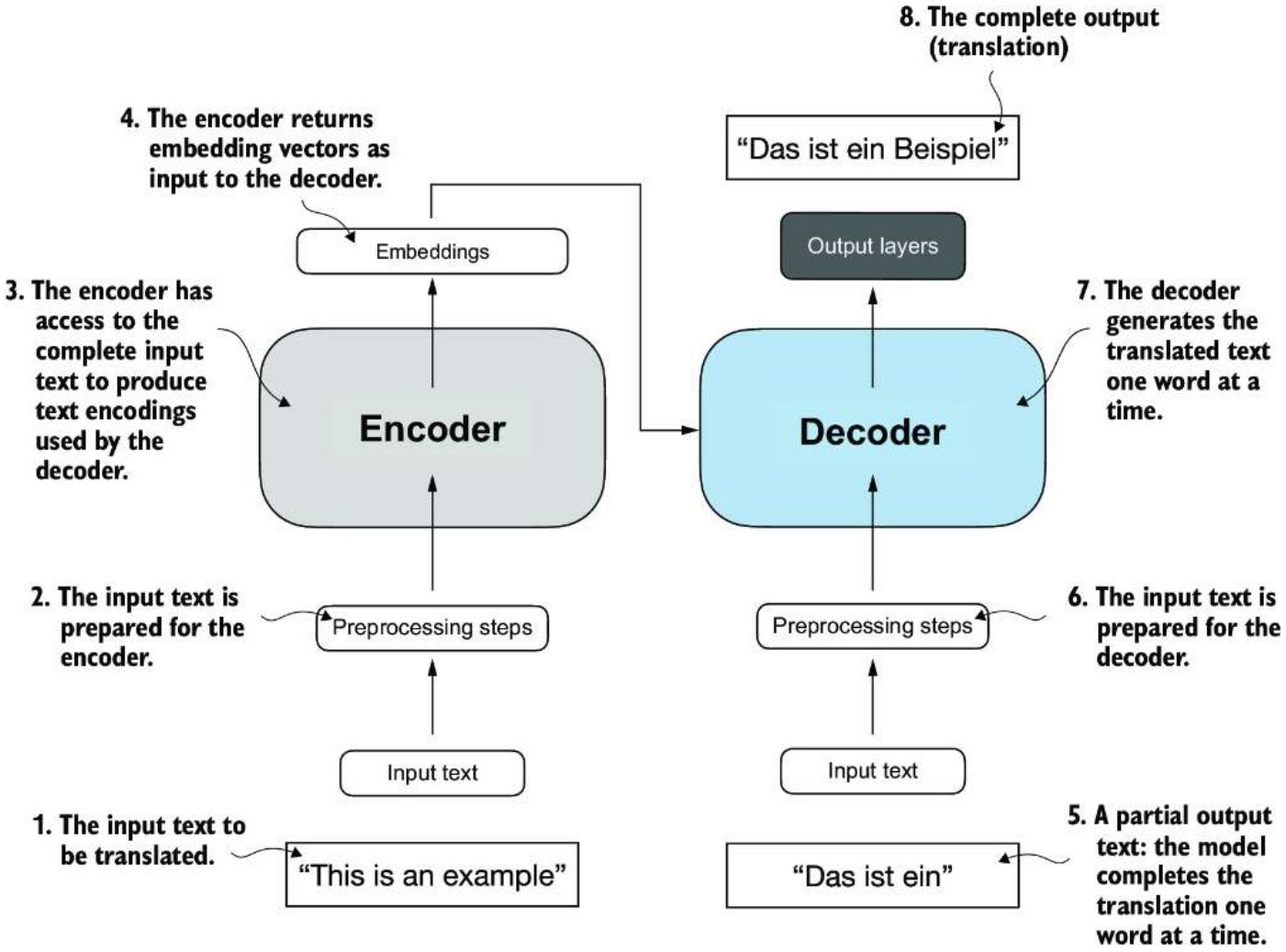


Verb tense

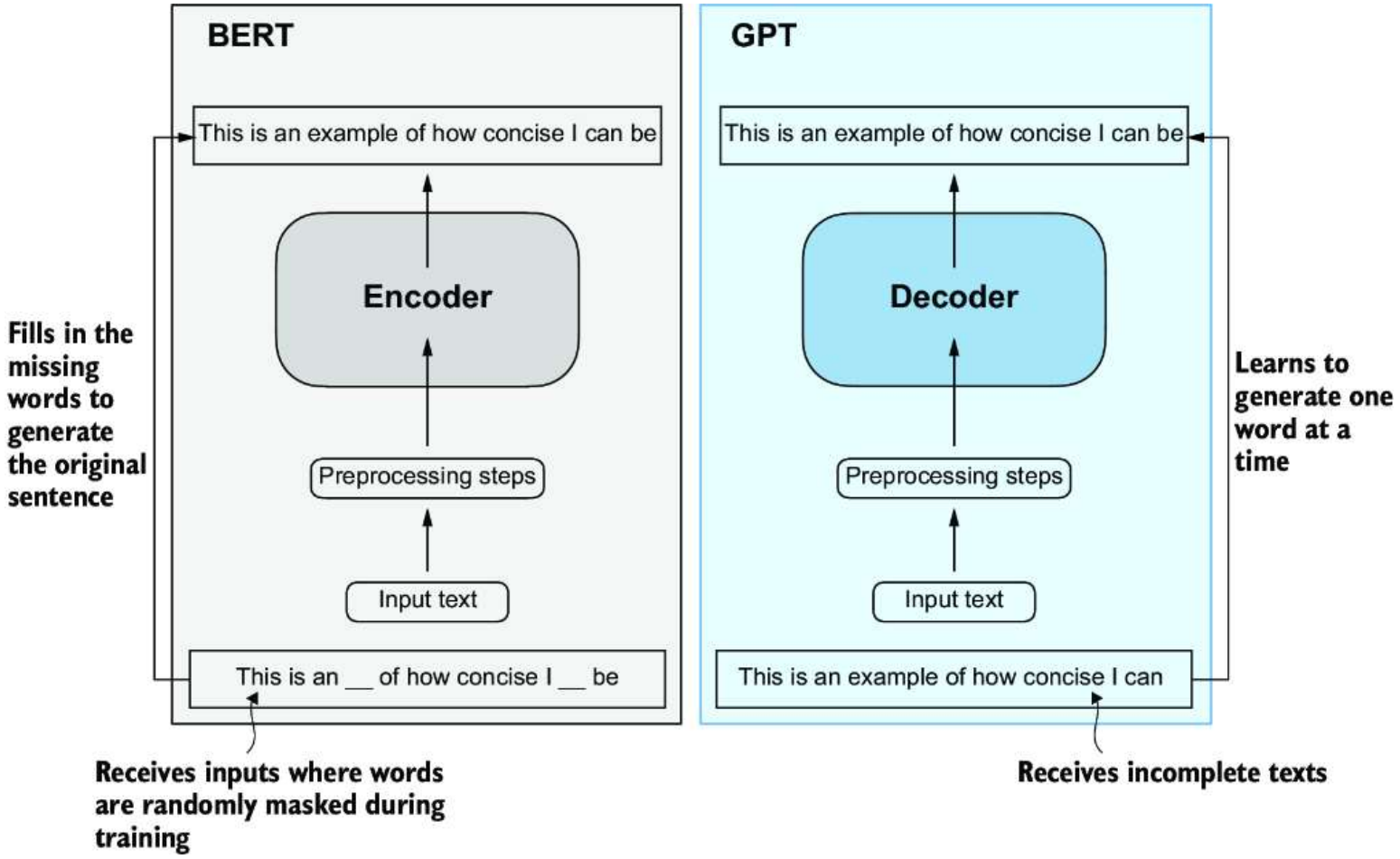


Country-Capital

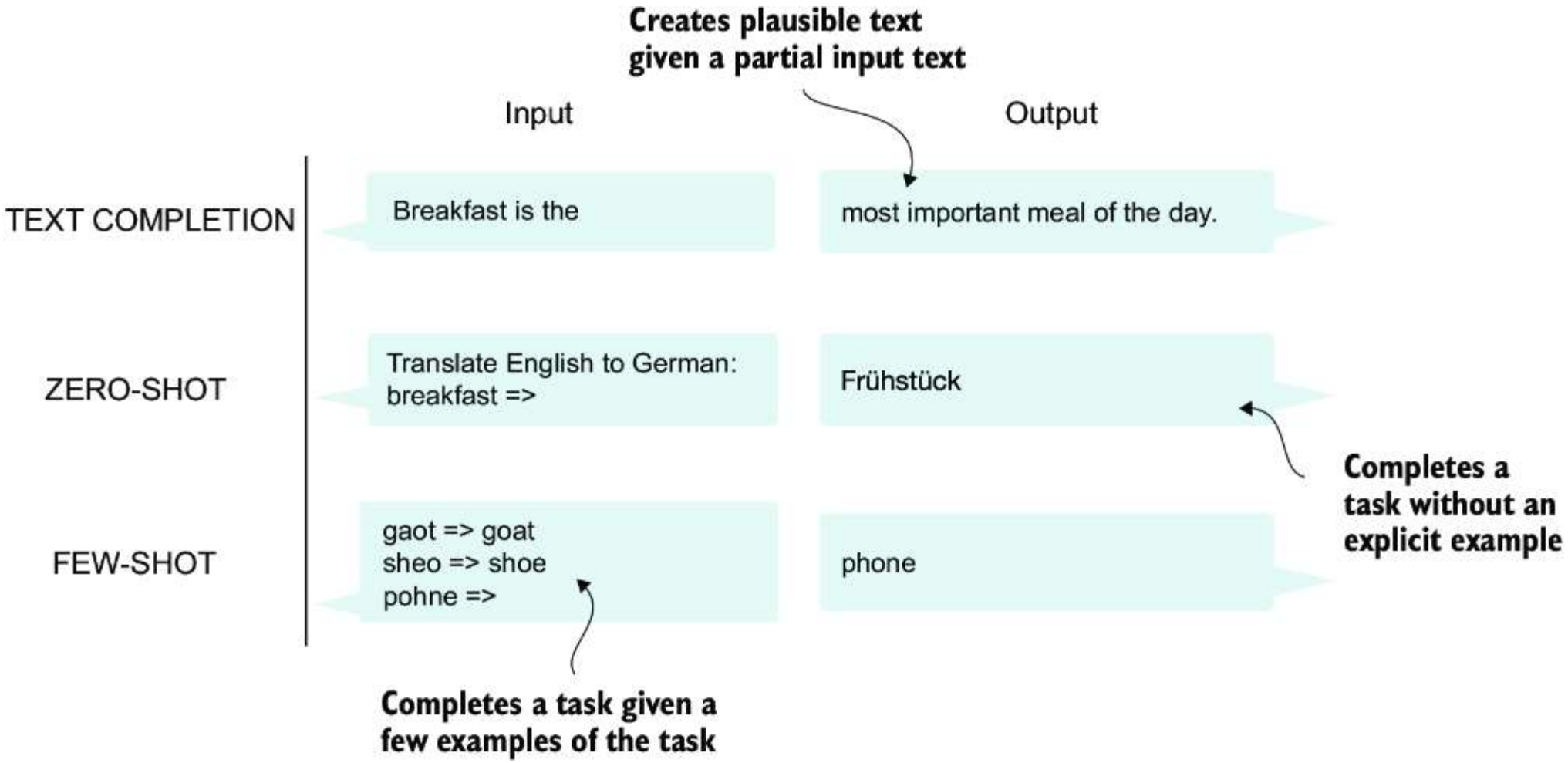
Encoder and decoder (original architecture for translation)



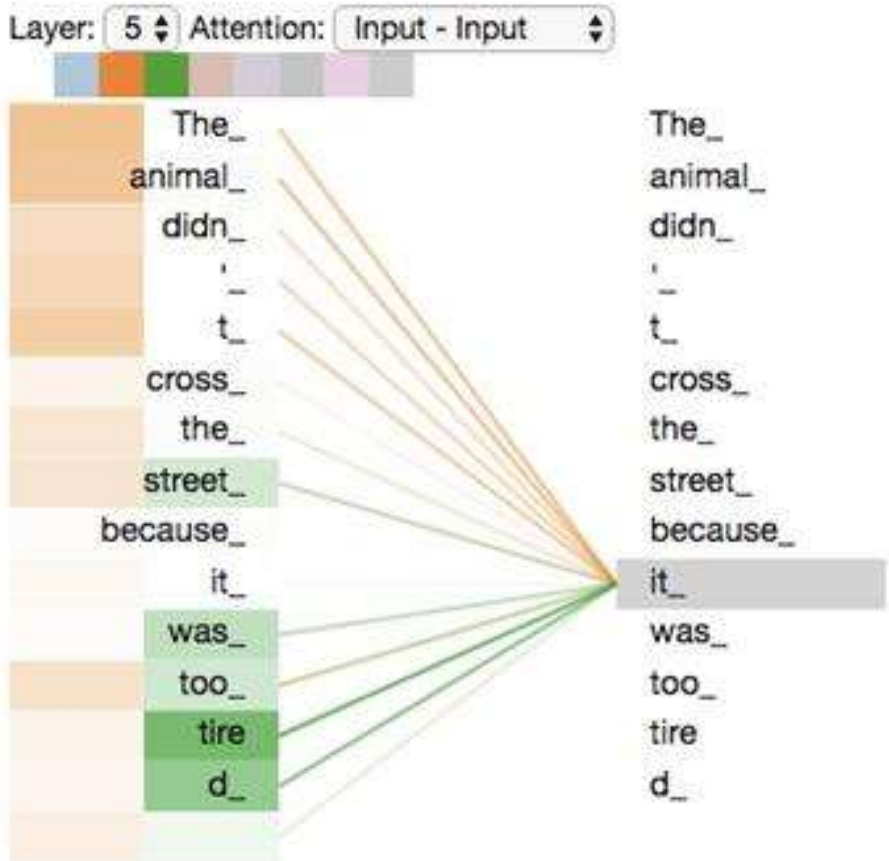
BERT vs GPT


BERT
This is an example of how concise I can be
Encoder
Preprocessing steps
Input text
This is an __ of how concise I __ be
Fills in the missing words to generate the original sentence
Receives inputs where words are randomly masked during training
GPT
This is an example of how concise I can be
Decoder
Preprocessing steps
Input text
This is an example of how concise I can
Learns to generate one word at a time
Receives incomplete texts

Shots (examples)



Attention



As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

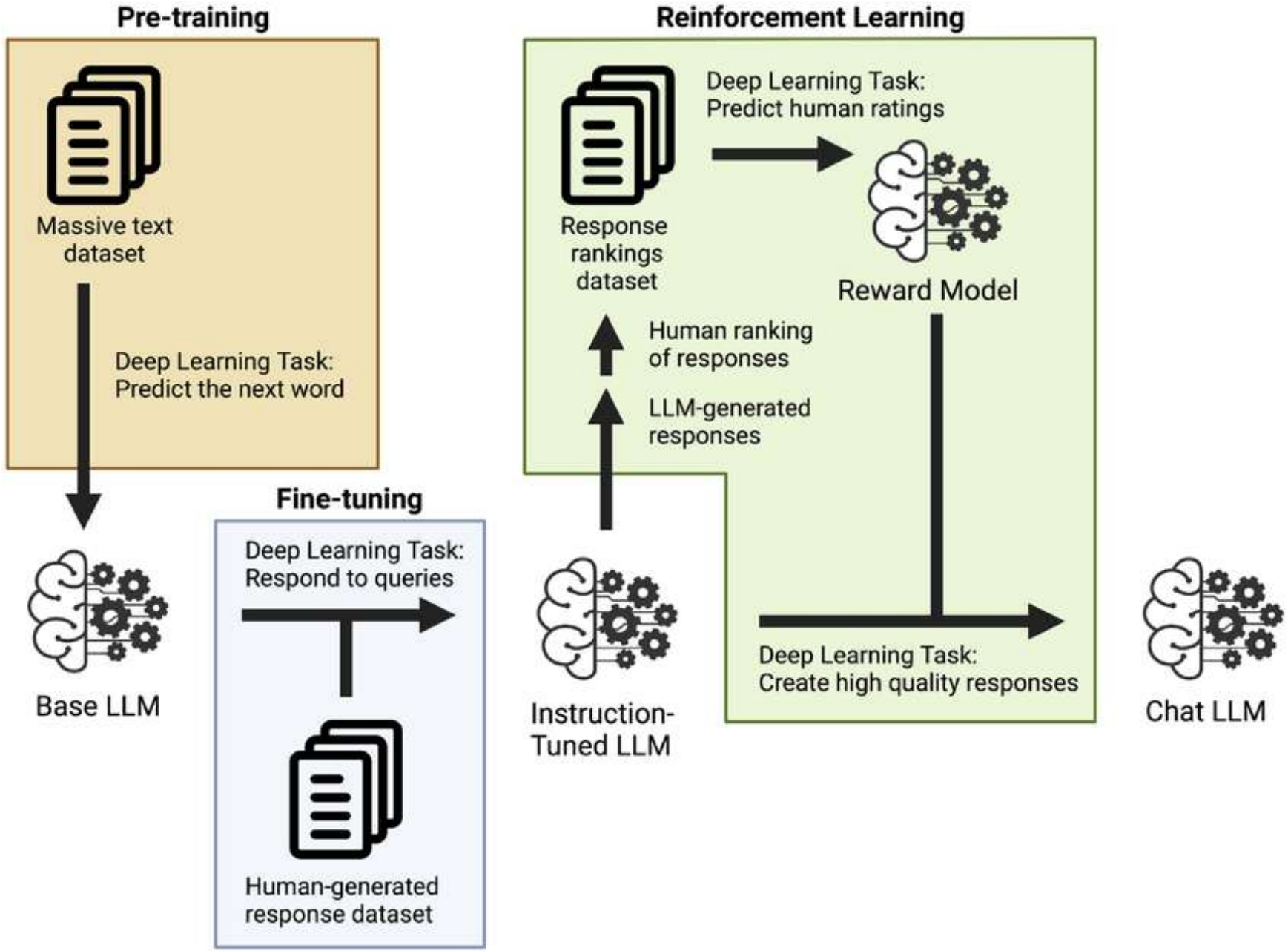
Attention as transforming information of word vectors: example for GPT2 (predecessor of GPT3)

When Mary and John went to the store, John gave a drink to … GPT2 predicts: Mary.

Three types of attention heads contributed to this prediction:

- Three heads copied information from the Mary vector to the final input vector (for the word to). GPT-2 uses the information in this rightmost vector to predict the next word.
- How did the network decide Mary was the right word to copy? A group of four attention heads marked the second John vector in a way that blocked the first three heads from copying the name John.
- How did the second head group decide John shouldn't be copied? Two attention heads marked the second John vector as a duplicate of the first John vector, which helped the second head group to decide that John shouldn't be copied.

These nine attention heads enabled GPT-2 to figure out that "John gave a drink to John" doesn't make sense and choose "John gave a drink to Mary" instead


Pre-training
Massive text dataset
Deep Learning Task: Predict the next word
Base LLM
Fine-tuning
Deep Learning Task: Respond to queries
Human-generated response dataset
Reinforcement Learning
Response rankings dataset
Deep Learning Task: Predict human ratings
Reward Model
Human ranking of responses
LLM-generated responses
Instruction-Tuned LLM
Deep Learning Task: Create high quality responses
Chat LLM

Stuff to read / look at

The illustrated transformer:

https://jalammar.github.io/illustrated-transformer/

Using LLMs

Reasoning: partially OK

- LLMs perform limited reasoning.
- Nobody knows where exactly is the plateau or limit.
- Deep / uncommon reasoning fails, though.
- LLMs do not contain search / planning / simulation algorithms.

Recent paper by Apple research: “adding seemingly relevant but ultimately inconsequential information to the logical reasoning of the problem led to substantial performance drops of up to 65% across all state-of-the-art models. “

The case of machine learning: gaining new knowledge by machine learning a software system to predict answers or actions similar to what has been observed

Issues:

- A very large number of examples is necessary, as well as a lot of compute
- Errors are inevitable; more generalization leads to more errors
- LLMs: hallucination (errors presented with confidence)
- Inability to explain the reasons for answers given
- Inability to perform complex search, for example, nontrivial planning or finding proofs
- Hard efficiently incorporate software tools
- Issues with memory: learning new information is very hard; forgetting is even harder

Issue: hallucinations (confidently generating wrong answers)

Human: Who was the mother of Afonso II, the third king of Portugal?

ChatGPT: The mother of Afonso II was Queen Urraca of Castile

Comment: *totally invented wrong "fact"*

Issue: hallucinations (confidently generating wrong answers)

Human: how James Joyce and Vladimir Lenin first met?

ChatGPT: James Joyce and Vladimir Lenin met in Zurich, Switzerland in 1916. Both men were living in exile in Zurich during World War I. Joyce was a writer and Lenin was a revolutionary. They met at the Cafe Odéon, a popular gathering place for artists and intellectuals in Zurich.

Comment: an encounter of these people has never been confirmed

Issue: not much of a reasoner (unless you use a well-known example)

Human: A man and a goat are on one side of the river. They have a boat. How can they both go across?

ChatGPT: The man takes the goat across the river first, leaving the boat on the original side. Then, the man returns alone with the boat. He leaves the goat on the other side and takes the boat back to the original side. Finally, he takes the cabbage across the river.

Issue: no short-term memory, very hard to teach anything, extremely hard to forget

Does not know anything about your specific context / company / organization

Two main approaches to alleviate:

- Retreival-Augmented Generation aka RAG: use special software to find relevant snippets from the docs of your context / organization and add these automatically to the prompt.

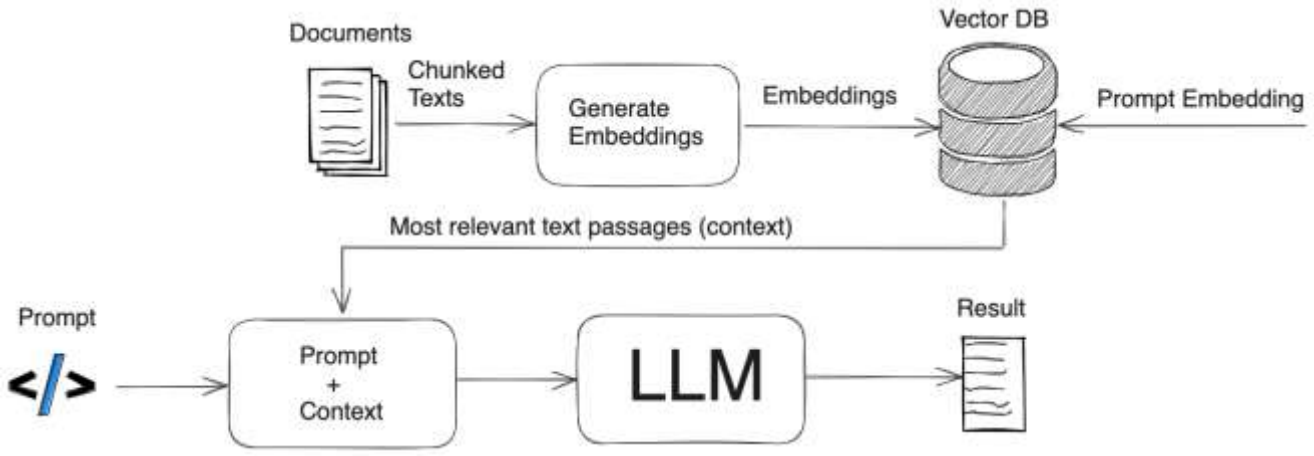
- Fine-tuning: perform additional teaching (like 1000 examples or so) to specialize for your context.

RAG (retrieval augmented generation): the main principle of using LLMs in an organization

Idea: search for relevant text snippets and add these to the LLM prompt.

The standard process with specialized RAG software:

- Collect all potentially relevant texts of an application domain and index all their short sections/paragraphs.
- Upon seeing a question, automatically select these sections/paragraphs which seem to be relevant. Use vector similarities / knowledge graphs and final evaluation.
- Automatically add likely relevant text snippets to the question prompt as background.



Another use case of LLMs: word / phrase vectors for RAG and similar

- Give GPT API a word, sentence or a large piece of text.
- Ask for the vector aka embedding (ca 3000 floating point numbers) for this.
- Suppose you have stored a large number of texts along with vectors.

Then you can, for example, use the vector of the question to find your existing texts / phrases with a similar vector.

Recommendations for specialized use cases

- A large prompt is better than a small prompt, up to a degree.
- Give context: explain briefly what role the LLM should take and what you expect.
- Give examples: even a few examples help, but tens / a hundred help even more.
- Explicitly prohibit the behaviour / output you observe but do not want.
- For reasoning-requiring tasks, it helps to instruct like “Reason step by step”.
- The longer the output, the more likely that LLM goes astray / makes errors.