Natural Language Processing

NLP with Deep Learning

Sebastian Stabinger — DeepOpinion.ai

April 22, 2023

Introduction

Who am I?

- Sebastian Stabinger (sebastian.stabinger@deepopinion.ai)
- PhD in Deep Learning (computer vision) from the University of Innsbruck
- Machine Learning Lead @ DeepOpinion.ai
  - Intelligent process automation

A First Step

If you want to follow along a few of the later experiments.
Create an account:
`https://platform.openai.com/signup`

Overview of Today

- Introduction into NLP
- Introduction into the Transformer Architecture
- Introduction into GPT and BERT
- Practical experiments
  - Experiments with ChatGPT, GPT-4

What is NLP?

- Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and humans through natural language.
- Goal: enable computers to understand, interpret, and generate human language in a way that is both meaningful and useful.

Why a Lecture about NLP?

- NLP is probably the most hyped topic in ML currently
- NLP is highly business relevant

Why you should be sad

- I have been working in NLP professionally for 5 years
- I probably know 5% of what there is to know (and more stuff is coming out faster than I can learn)
- So .. 4h won't cut it . . .

Why you should be excited

- You will get a an overview
- You will understand the foundations of the two most practically relevant technologies in NLP after these 4h (GPT and BERT)

Examples for NLP tasks

Text Classification Assigning categories or labels to a piece of text

Machine Translation Translating text from one language to another

Summarization Generating a shorter version of a piece of text while retaining the most important information

Question Answering Providing answers to natural language questions

Text Generation Generating new text based on a given input or prompt

Information Extraction Identifying and extracting structured information from unstructured text

Chat Systems As a natural interface to computer systems

Speech Recognition Converting spoken language into written text

Speech Synthesis Generating spoken language from written text

History: Traditional NLP → Transformer

NLP existed long before machine learning!

Traditional NLP

Rule-based systems and hand-crafted features (regular expressions, context-free grammars, finite state automata, parse trees, . . .)

Modern NLP

Still uses hand crafted scaffolds (e.g. the notion of parts of speech, parse trees) but uses data to fill those scaffolds

NLP and Deep Learning

Uses neural networks and learns everything from data

NLP and Transformer Architectures

A specific neural network architecture that has taken over NLP since 2018

Successes of NLP before Deep Learning

- Conversational Agent, starting in the 1960s (e.g. ELIZA, Winograd SHRDLU)
- Summarization, starting in the 1990s (e.g. Gisting)
- Spell/grammar checking, starting in 1970s/80s (e.g. Word/Grammatik)
- Document retrieval, starting in the 1990s (e.g. AltaVista)
- Support systems for translators, starting in 1960s (e.g. Systran, Lingo24)
- Text-to-speech for vision impaired people, starting in 1950s (e.g. Kurzweil Reading Machine)
- Speech recognition, starting in 1980s (e.g. Dragon NaturallySpeaking)
- . . . and many more . . .

Types of ML and where to put modern NLP?

Supervised (classification)

- Training on labeled data

Self-Supervised Learning (pre-training of GPT, BERT, etc.)

- Works like supervised training but ...
- ... the labels can be automatically created from the data

Unsupervised

- Trained on unlabeled data

Reinforcement Learning (part of e.g. ChatGPT)

- Trained using feedback only (output was good/bad)

How to Bring Text into Neural Networks?

We have a slight problem

The Problem

We have words, but neural networks need numbers ...

Solution 1

- Just give every letter a number and use those
- Does not work very well, although modern systems start to be able to handle this

Better solution

Tokenization

Tokenization

Classical

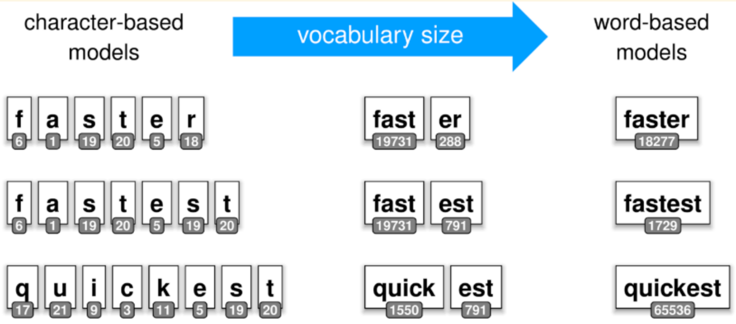
1. Define a vocabulary size
2. Count how often a word appears in some big text corpus
3. Assign a token-id to the most frequent words

Problem: Rare words have to be left out → Subword Tokenization

Subword Tokenization

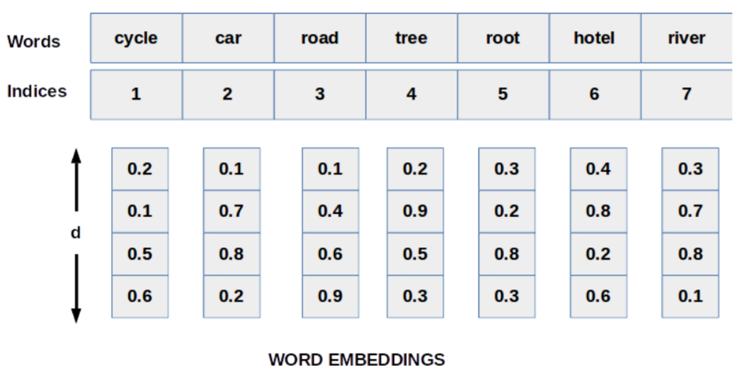
- Instead of counting words, the words are also split into sub-words (down to single characters)
- The goal is to fill the vocabulary in a way so that words have to be split into as few tokens as possible
- Prominent algorithms: Byte Pair Encoding (BPE), WordPiece, and SentencePiece
- Advantage: No out-of-vocabulary words
- Problem: The neural network has to be able to handle that

Tokenization — Character → Subword → Word



Token Embeddings

- We now converted words into numbers, but neural networks can't handle integers very well
- Each integer is assigned a high-dimentional (e.g. 765D) vector (called embedding vector) in the first, so called embedding layer, of the neural network
- The values of that vector are trained with the rest of the network and the network works with these vectors



Demo

The Transformer Architecture

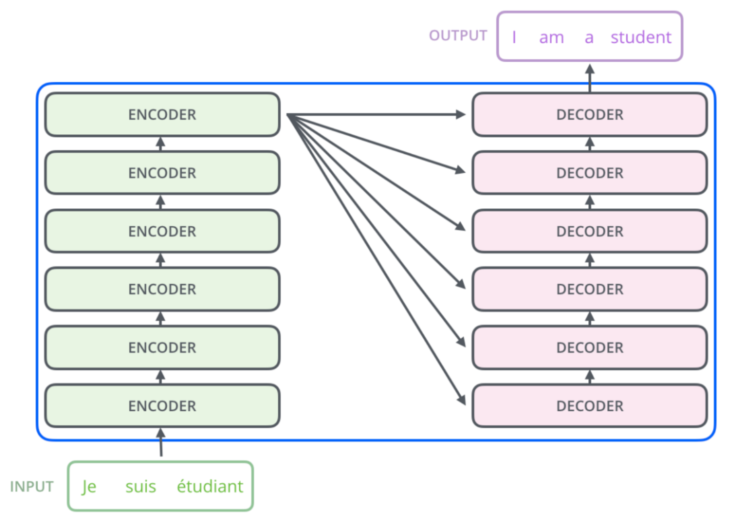
General Info

- Introduced by Vaswani et al. in 2017 (at Google) in the paper *"Attention is All You Need"*
- Originally developed for machine translation
- Revolutionized NLP
- Replaced Recurrent Neural Networks (RNNs) i.e. mostly Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks for most NLP tasks
- Scales exceptionally well with model size and amount of training data (we haven't found a limit yet)

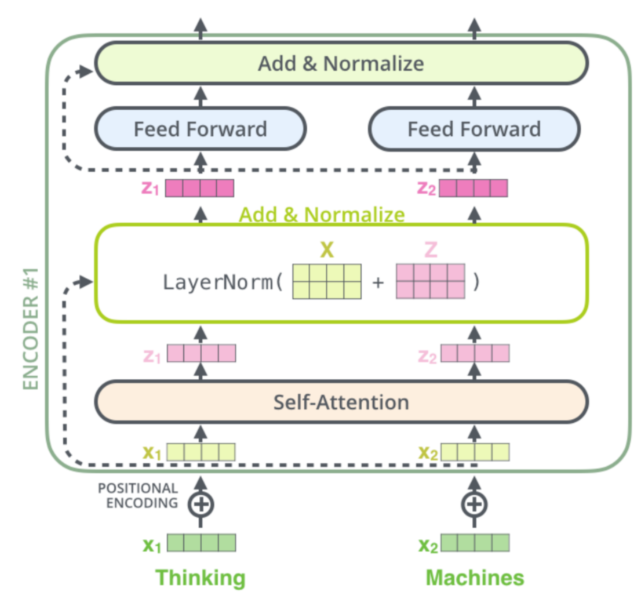
Key Innovations (well ... at least they combined all of it)

- Self-attention mechanism (!!!)
- Multi-head attention
- Positional encoding

Overview


OUTPUT
I am a student
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant

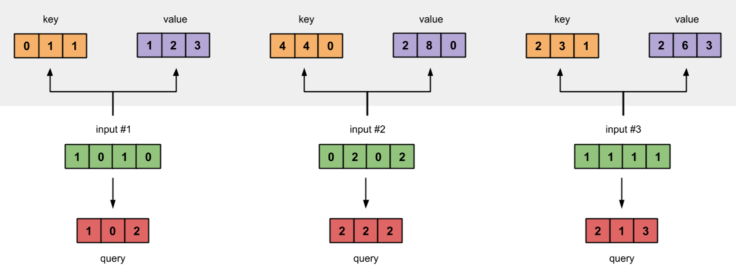
Encoder



Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
z1
z2
Self-Attention
x1
x2
ENCODER #1
POSITIONAL
ENCODING
x1
x2
Thinking
Machines

How does Self-Attention work?

Key, Query, and Value vectors

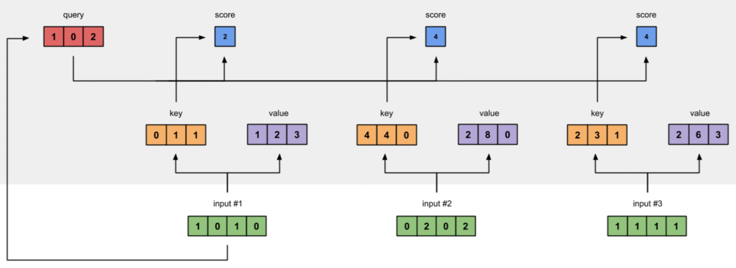


- Using matrices M_K, M_V, and M_Q the vectors `key`, `value`, and `query` are generated out of all input vectors
- M_K, M_V, and M_Q are the same for each position
- M_K, M_V, and M_Q are learned during training

Images from https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

How does Self-Attention work?

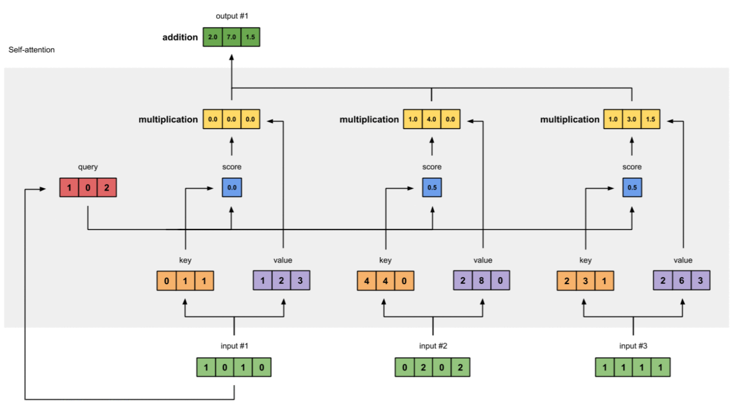
Determine a Score



- For each position
  - Calculate the dot-product between it's `query`-vector and all `key`-vectors giving a `score`
  - The `score` values are normalized with a softmax so their sum $= 1$

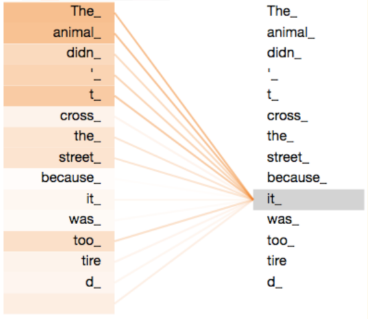
How does Self-Attention work?

Add the `value` vectors using the `score` values as weights



What does Self-Attention do in the end?

Self-attention enables a model to weigh and incorporate context from different parts of an input sequence, allowing it to capture dependencies and relationships between words efficiently.

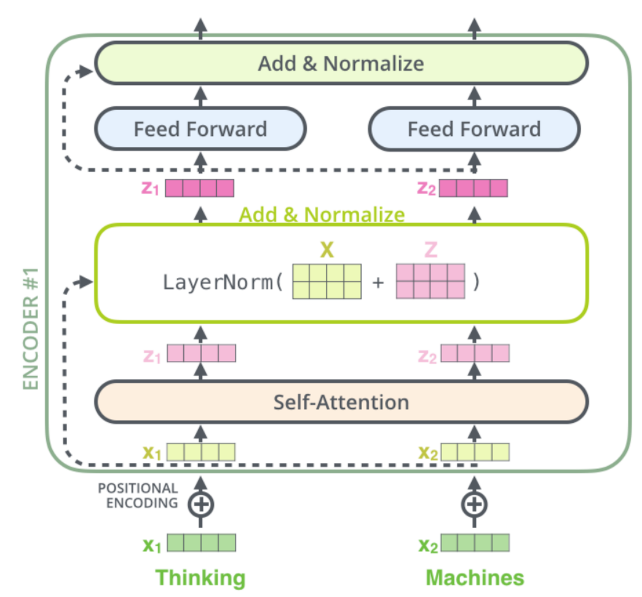


- Put tokens in relation to each other
- E.g. to resolve homonyms (spring $\rightarrow$ season, coiled piece of metal, natural source of water)
- Handle words which consist of multiple tokens

Multi-Head Attention

We apply this self-attention multiple times in parallel

Encoder



Add & Normalize
Feed Forward
Feed Forward
z_1
z_2
Add & Normalize
X
Z
LayerNorm(+)
ENCODER #1
z_1
z_2
Self-Attention
x_1
x_2
POSITIONAL
ENCODING
x_1
x_2
Thinking
Machines

Token Positions

The Problem

- The transformer architecture is position invariant (i.e. the position of a token does not matter to the system)
- This is bad for an NLP system. The order of words is important:
  - The dog chased the cat
  - The cat chased the dog

The Solution

Positional Encoding

Positional Encoding

Basic Idea

- Add a vector to the token embeddings that is different for each position
- The neural network can use this information to orient itself

Implementation

```
"""
token_nr: The token number, starting at 0
i: Iterator from 0 to d_model/2
d_model: The max number of dimensions in the embedding
"""
function pe(token_nr::Integer, i::Integer, d_model::Integer)
  if mod(i,2) == 0
    sin(token_nr / 10000^(2i / d_model))
  else
    cos(token_nr / 10000^(2i / d_model))
  end
end
```

Positional Encoding — Example

Example for token embeddings with 8 dimensions and max 4 tokens

```
[pe(token_nr, i, 16) for i in 0:15, token_nr in 0:3]
```

0.0	0.841471	0.909297	0.14112
1.0	0.995004	0.980067	0.955336
0.0	0.00999983	0.0199987	0.0299955
1.0	1.0	0.999998	0.999996
0.0	0.0001	0.0002	0.0003
1.0	1.0	1.0	1.0
0.0	1.0e-6	2.0e-6	3.0e-6
1.0	1.0	1.0	1.0

Nice Property

Adding two positional encoding vectors preserves their distance

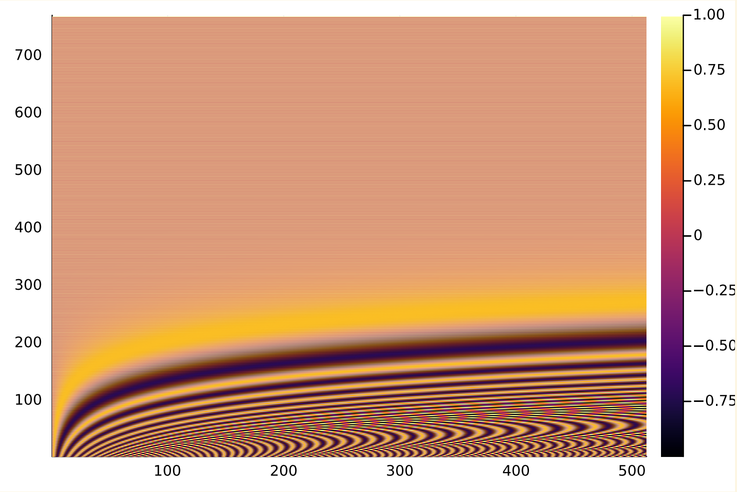
Alternatives

There are many alternatives for positional encoding by now

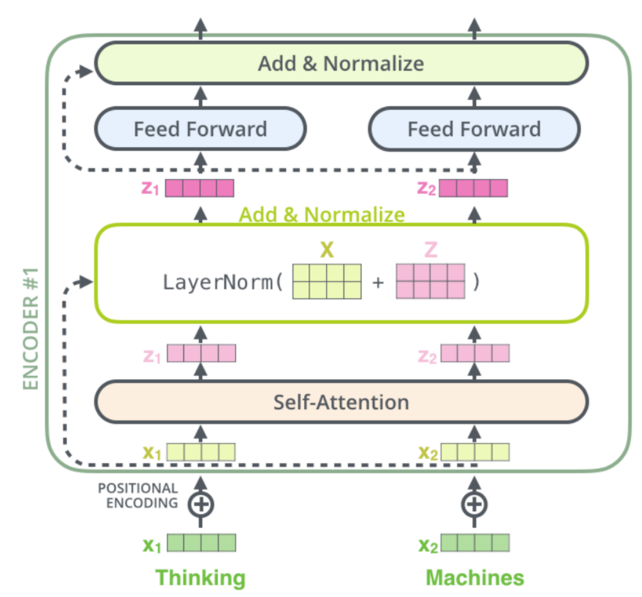
Positional Encoding — Realistic Example

Example for token embeddings with 768 dimensions and max 512 tokens (used in BERT base)

```
using Plots
heatmap([pe(token_nr, i, 768) for i in 0:767, token_nr in 0:511])
```

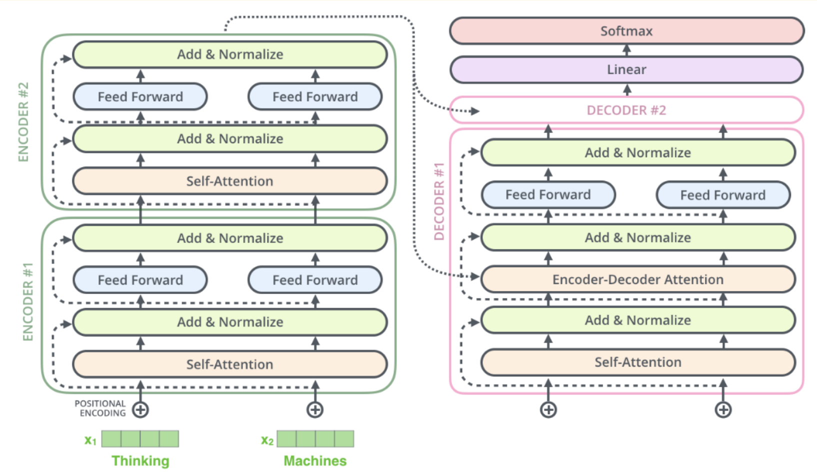


Encoder



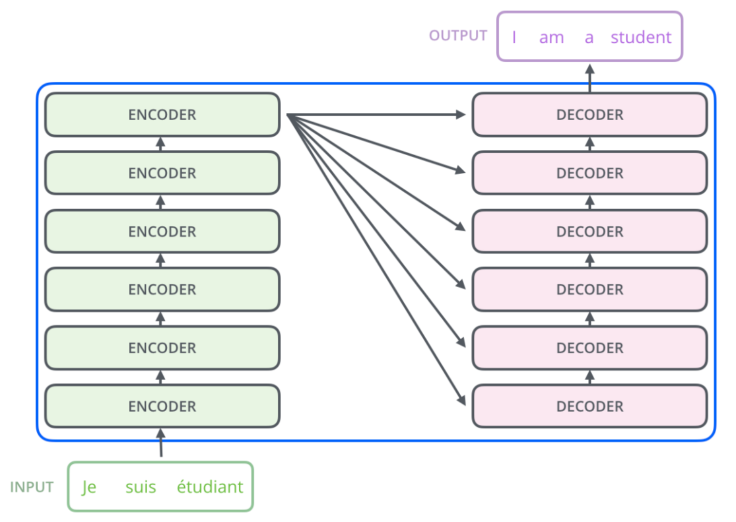
Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
z1
z2
Self-Attention
x1
x2
ENCODER #1
POSITIONAL ENCODING
x1
x2
Thinking
Machines

Encoder extracts semantic meaning of the text → Forwarded to the Decoder → Produces text with the same semantic meaning in a different language



How is a traditional Transformer Trained?

- Present many pairs of text and their translation
- Train end to end



How is a traditional Transformer Trained?

- You need a lot of sentence pairs in two languages (e.g. English sentences and their correct translation in German)
- You present the encoders with one sentence in English
- You calculate what words the decoders would choose as a translation
- You compare to the correct translation with a loss
- You calculate the gradient of the weights with respect to this loss
- You change the weights of the whole system so the next time the correct translation is a bit more likely
  - That is pretty easy since the whole system is differentiable. I.e. we can directly calculate in which direction the parameters have to be changed to improve things.
- Rinse and repeate . . . a few billion times

The Groundbreaking Insight

Insights

- People found out that using the encoders and decoders separately can be used for all kinds of NLP tasks
- They seems to learn something more general about language while learning to translate from one language to another

What is Transfer Learning?

- Train a neural network on one task
- Take this trained network (maybe modify it a little bit) and train it on another task
- What has been learned on the first task is transferred to the second task

Why is this useful?

You need much less training data for the second task

When does that work best?

When you can easily get training data for the first task

Example

- Train a transformer network on translation
- Re-use the network for sentiment classification
- The network learns about language in the first task and only task-specific things in the second task

But we Have a Problem

The Problem

To train the encoders we have to train on a language translation task and not that much data is available

The Result

People tried to train parts of the Transformer architecture without translation:

- Generative: G enerative P re-trained T ransformer
- Discriminative: B idirectional E ncoder R epresentations from T ransformers

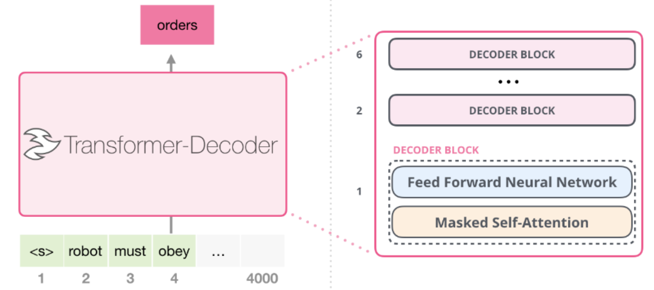
GPT

Architecture

- Basically a stack of Transformer blocks

Task

Given a text, predict a probability distribution over the next tokens



Basically: Auto-complete on steroids

Images from https://jalammar.github.io/illustrated-gpt2/

Full GPT-2 implementation in Python (!!!)

```python
def gelu(x):
    return 0.5 * x * (1 + np.tanh(np.sqrt(2 / np.pi) * (x + 0.044715 * x**3)))

def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=-1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=-1, keepdims=True)

def layer_norm(x, g, b, eps: float = 1e-5):
    mean = np.mean(x, axis=-1, keepdims=True)
    variance = np.var(x, axis=-1, keepdims=True)
    return g * (x - mean) / np.sqrt(variance + eps) + b

def linear(x, w, b):
    return x @ w + b

def ffn(x, c_fc, c_proj):
    return linear(gelu(linear(x, **c_fc)), **c_proj)

def attention(q, k, v, mask):
    return softmax(q @ k.T / np.sqrt(q.shape[-1]) + mask) @ v

def mha(x, c_attn, c_proj, n_head):
    x = linear(x, **c_attn)
    qkv_heads = list(map(lambda x: np.split(x, n_head, axis=-1), np.split(x, 3, axis=-1)))
    causal_mask = (1 - np.tri(x.shape[0], dtype=x.dtype)) * -1e10
    out_heads = [attention(q, k, v, causal_mask) for q, k, v in zip(*qkv_heads)]
    x = linear(np.hstack(out_heads), **c_proj)
    return x

def transformer_block(x, mlp, attn, ln_1, ln_2, n_head):
    x = x + mha(layer_norm(x, **ln_1), **attn, n_head=n_head)
    x = x + ffn(layer_norm(x, **ln_2), **mlp)
    return x

def gpt2(inputs, wte, wpe, blocks, ln_f, n_head):
    x = wte[inputs] + wpe[range(len(inputs))]
    for block in blocks:
        x = transformer_block(x, **block, n_head=n_head)
    return layer_norm(x, **ln_f) @ wte.T
```

Training

Pretty simple ... (self-supervised training)

- Get a lot of text
- Send in a piece
- Let the model predict the next token
- Compare prediction to actual next word using a loss
- Repeat (a lot)

Amount of data

GPT-2 40 GiB

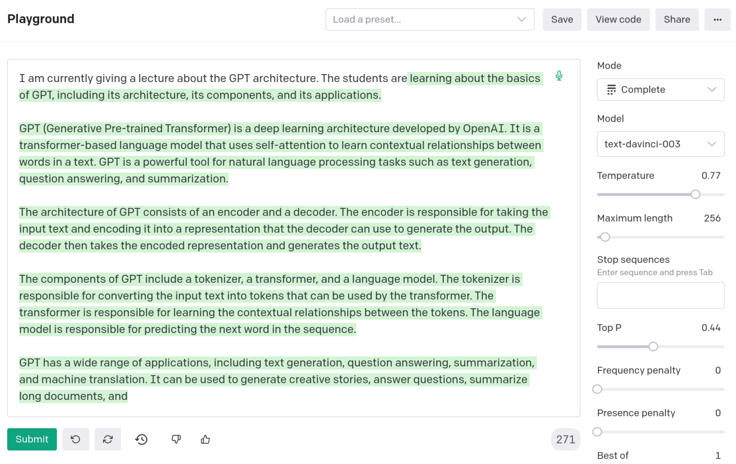
GPT-3 570 GiB

So what do all GPT models do?

Given a text, they give a propability distribution over the vocabulary for the next word/token ... Doesn't sound super exciting?

Doesn't sound that exciting . . .

Playground Demo



Some facts about GPT-3 ...

- The model is HUGE
  - 175 billion parameters
    - Model size: 652 GiB (FP32), 326 GiB (FP16)
- Such Large Language Models (LLMs) are by far the largest models used in deep learning
  - Up to over 1 trillion parameters for research models
- Training and Inference Infrastructure
  - Needs 5xA100 80GiB GPUs to run 16 bit inference (~17,000 € per A100)
  - A server that can run inference is approx. 120.000€
  - Renting costs around 14€ per hour
  - Training cost around 4.6 millon $ in compute

GPT-4?

Unfortunately basically nothing is known

The Big Surprise

- Training such huge GPT architectures on a lot of data show **emerging properties**. Happened during the switch from GPT-2 to GPT-3.
- They can do things they were not explicitly trained on . . .

Input: Das Haus ist groß.
Output: The house is big.

Input: Wir gingen zum Hafen
Output: We walked to the harbor

Input: Ich halte gerade einen Vortrag über GPT and der Universität Innsbruck.
Output: I am currently giving a lecture on GPT at the University of Innsbruck.

Say what . . . . ????? This is still just a system predicting the next tokens!

Why is this happening?

No One Knows!

But we like it!

Demo Time!

Examples — It can program

A function that finds the biggest and smallest element of a list of numbers and returns their sum can be written in Julia as follows:

```
function sum_extremes(list)
  # Find the biggest and smallest elements
  biggest = maximum(list)
  smallest = minimum(list)

  # Return the sum of the two
  return biggest + smallest
end
```

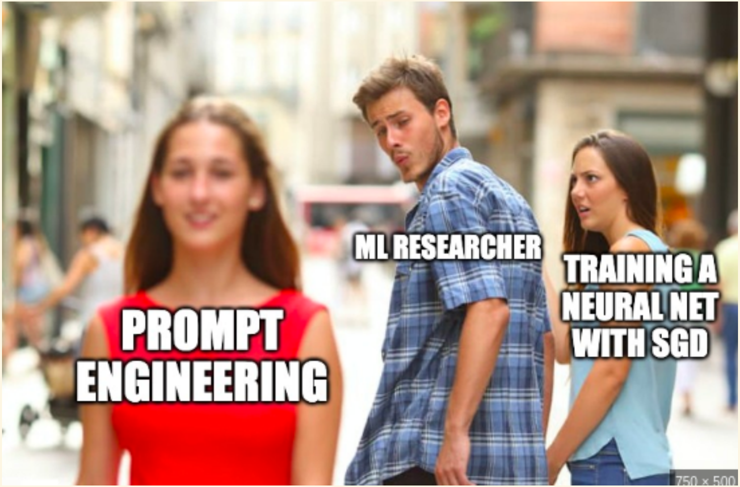
Exercise

Get GPT-3 to do sentiment analysis I.e. we want the system to tell us that

- *"The food was pretty bad"* should be negative
- *"I really enjoyed the drinks and the food was also great"* should be positive

Prompt Engineering

- Getting LLMs to do what you want them to do with text input is called Prompt Engineering
- Prompt Engineer is already a job title Salary: $250K - $335k
- `https://www.promptingguide.ai/`



How to Actually Produce Text With Language Models?

Just always selecting the most likely next tokens (greedy decoding) has a few problems:

- You will always get the same response (this is also good in some cases)
- The responses can be a bit boring
- The process tends to get stuck in a loop (producing the same sentences over and over)

Alternatives

- Pure sampling
- Temperature-based sampling
- Nucleus sampling
- Top-K sampling
- Beam-search

Alternatives to Greedy Sampling

Pure sampling

We select the next token following the probability distribution

Temperature-based sampling

- Like Pure Sampling, but the probability distribution is flattened/accentuated using a temperature value
- High temperature $\rightarrow$ the distribution is flattened $\rightarrow$ Less likely words are selected more often
- Low temperature $\rightarrow$ the distribution is accentuated $\rightarrow$ Less likely words are selected less often

Alternatives to Greedy Sampling

Nucleus sampling

Only most likely tokens whos probabilities sum to a parameter top-P

Top-K sampling

Like pure sampling but only the K most likely tokens can be selected

Alternatives to Greedy Sampling

Beam-search

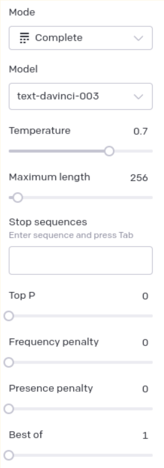
- Instead of the most likely token, keep the N (the beam width) most likely tokens
- Continue with all selected tokens in parallel
- For each of the tokens, again select the N most likely next tokens
- This gives us N*N possibilities
- Prune all except for the N sequences with the highest overall probability
- Repeat

What is used in practice?

Often a combination of temperature-based, nucleus, and Top-K sampling together with repetition penalties.

Experiment Time

Use the OpenAI Playground and get a feeling for how Temperature, Top-P, and the other parameters change the output

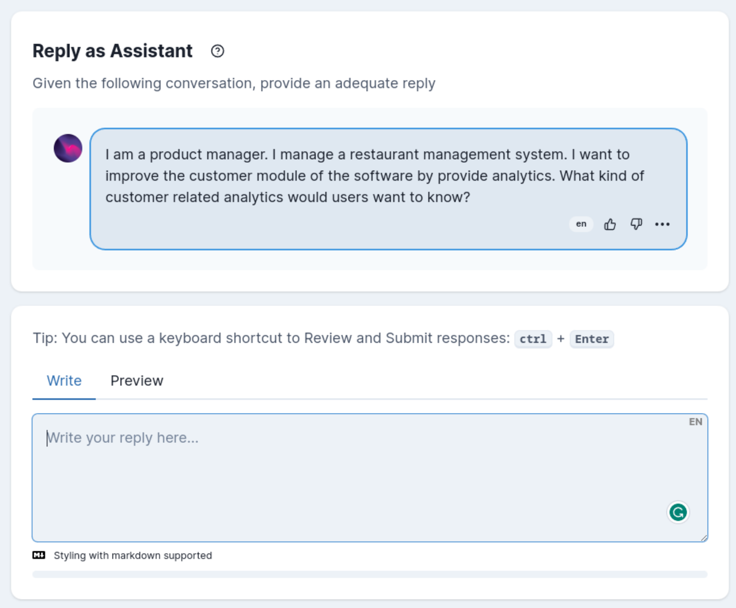


The Disadvantage of GPT-3 and Similar LLMs

- You have to prompt GPT in a very weird way and you often need to supply examples etc.
- I.e. it's not very easy to use
- The solution: Instruction Tuning

Instruction Tuning

- Explicitly train the model to generate helpful answers to instructions
- This was presented under the name InstructGPT
- Quite a lot of manual work, but improves the usefulnes quite a bit
- An open approach and dataset: www.open-assistant.io


Reply as Assistant
Given the following conversation, provide an adequate reply
I am a product manager. I manage a restaurant management system. I want to improve the customer module of the software by provide analytics. What kind of customer related analytics would users want to know?
en
Tip: You can use a keyboard shortcut to Review and Submit responses: ctrl + Enter
Write
Preview
EN
Write your reply here...
Styling with markdown supported

From InstructGPT to ChatGPT

The Problem

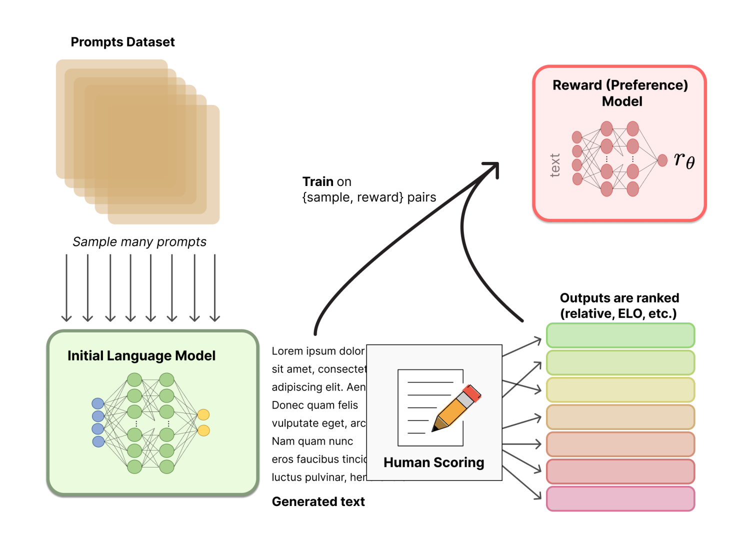
- We would like to tell the system what responses we expect from it
- But manually writing responses as demonstrations is very time consuming

The Solution

Reinforcement Learning from Human Feedback (RLHF)

- For a prompt (question, request, etc.), let the model generate multiple answers
- Let human subjects rank those answers
- Train a Reward Model to learn which answers humans prefer (text to scalar reward)
- Use this Reward Model to train the InstructGPT using reinforcement learning

From InstructGPT to ChatGPT



Open-Assistant Example

Sure!
An artery is carrying

en

Can you please elaborate more, and provide a description of both and the main differences please.

en

Arteries and veins are the major blood vessels. If they are all essential, the veins and the arteries do not have the same role at all. Because we often confuse them, we take stock of their differences

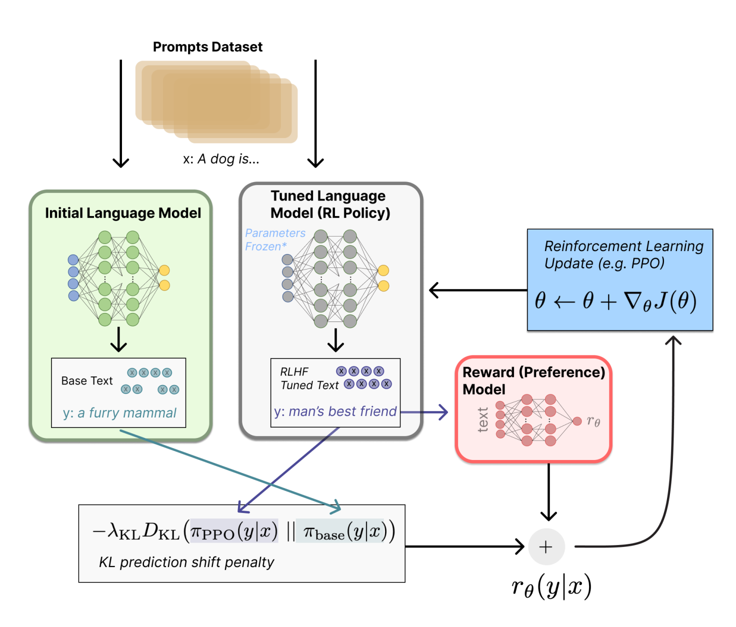
It is generally believed that arteries differ from veins only in size, and that they are larger than veins. But the reality is a little more complex. Veins and arteries differ above all in their role in blood circulation. Arteries carry blood from the heart to the organs, while veins carry blood from the organs back to the heart.

The blood system is organized into two circulations. The small one between the heart and the lungs loads the blood with oxygen. The pulmonary arteries go from th...

Arteries and veins are both types of blood vessels in the circulatory system, but they have several important differences. Arteries are responsible for carrying oxygenated blood away from the heart to the rest of the body, while veins carry deoxygenated blood back to the heart.

Arteries are typically thicker and more muscular than veins, as they must withstand the pressure of the blood being pumped by the heart. They also have a smaller internal diameter than veins. Arteries have walls that contain three layers: the intima, media, and adventitia. The intima is the inner lining of the artery and is made up of a thin layer of endothelial cells. The media is the middle layer and contains sm...

From InstructGPT to ChatGPT



ChatGPT

- ChatGPT is GPT-3 that is instruction fine-tuned and then fine-tuned further using RLHF
- It has caused quite a stirr!

Open-Source Language Models

- Llama (llama.cpp)
- BLOOM
- OPT
- GPT-J
- GPT-NeoX
- ...

Open-Source Alternatives to ChatGPT

- Alpaca
- Vicuna
- GPT4All
- Open-Assistant
- ...

ChatGPT Demo

Model: GPT-4

Write an opening speech for a 4h university course on natural language processing

Ladies and gentlemen, esteemed professors, colleagues, and dear students, I am honored to welcome you to this 4-hour course on Natural Language Processing, or NLP for short. As we embark on this fascinating journey together, we'll dive into the world of computational linguistics, machine learning, and artificial intelligence to better understand the intricate and powerful ways in which we communicate as human beings.

Language is the very cornerstone of our society, providing a framework for sharing knowledge, building relationships, and expressing our deepest emotions. Through the ages, it has evolved to meet our ever-changing needs and desires, and today, we stand on the cusp of another transformative shift in human communication with the emergence of NLP technologies.

Let's do some experiments

BERT

Yes, It is Named after Bert of Sesame Street . . .

. . . because Facebook released ELMO and . . . nerds will be nerds

It got a bit crazy

- BigBird
- ERNIE (two different models . . .)

Architecture

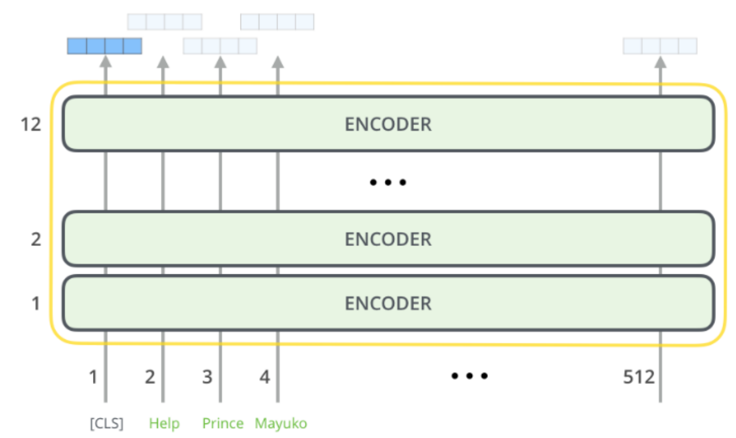
Misc

- BERT = Bidirectional Encoder Representations from Transformers
- BERT was presented in October 2018 by Google

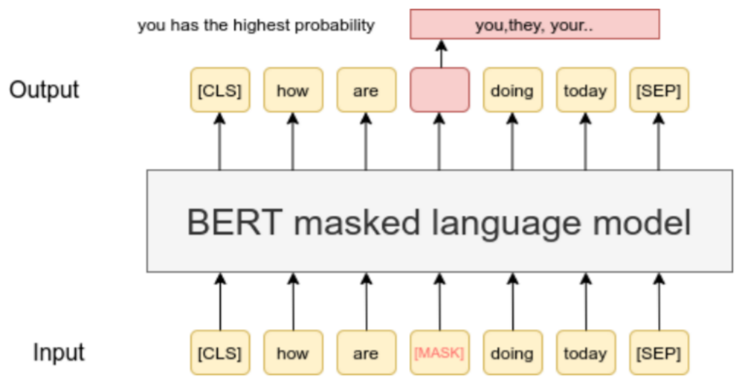
What is BERT?

- BERT is a specially trained and modified version of the Encoder-Part of the Transformer architecture
- BERT can be used as a core for many tasks in NLP

What does BERT do? High Level Overview?



Self-Supervised Pre-Training



Self-Supervised Pretraining

Masked Word Prediction (!)

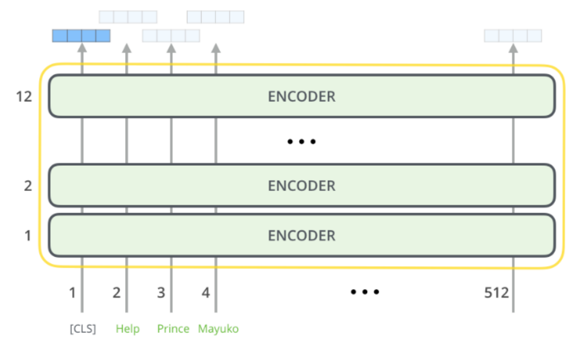
- Words are masked out by a special token and BERT is trained to correctly fill in the blanks
- `I hope my [MASK] is going [MASK]`

Next Sentence Prediction

- Two sentences are given to BERT and it predicts whether the sentences are following each other in the document or if they were randomly selected.
- `Peter bought a nice house [SEP] The knights defended the castle successfully`
- Not used anymore in more modern architectures

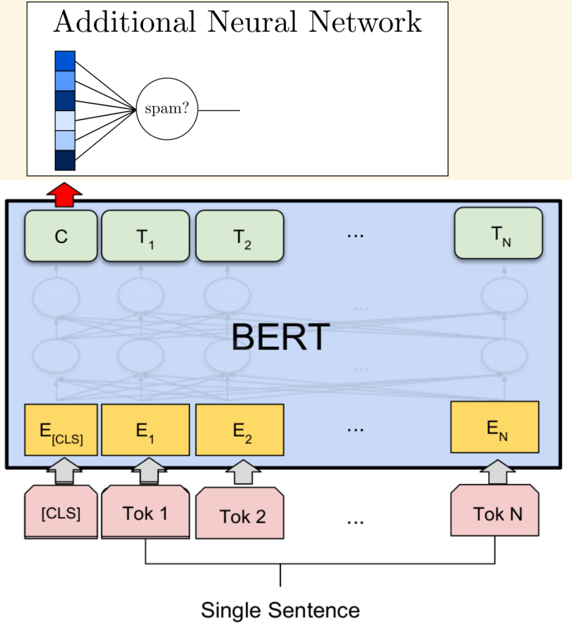
Solving Actual Tasks using BERT

After BERT has been pre-trained we can use it to solve supervised tasks by fine tuning

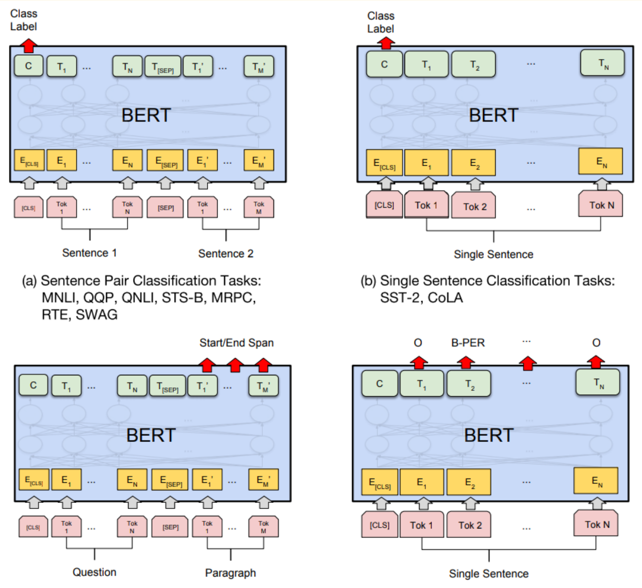


- BERT produces one output vector for each word/token given to the system
- We can attach arbitrary neural networks to process those vectors further

Supervised Fine Tuning For Sentence Classification



Supervised Fine Tuning for Different Tasks



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER