ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI
SCIENZE STATISTICHE "PAOLO FORTUNATI"

# 8 - Pre-trained Language Models

Giovanni Della Lunga

giovanni.dellalunga@unibo.it

Halloween Conference in Quantitative Finance

Bologna - October 25-26-27, 2023

# Transfer Learning

# Introduction

- Transfer learning is a machine learning and deep learning technique where a model trained on one task is adapted for use on a second related task.
- It leverages knowledge gained while solving one problem and applies it to a different, but related, problem.
- Transfer learning is particularly powerful when you have a small amount of data for the target task because it allows you to utilize the knowledge learned from a larger, source task.

# Introduction

- **Pre-trained Models**: In transfer learning, a pre-trained model, often trained on a large dataset for a different task (the source task), is used as the starting point. This pre-trained model already has learned features, representations, or weights that capture general patterns and knowledge.
- **Fine-tuning**: After initializing with the pre-trained model, you adapt it to your specific task (the target task) by making further adjustments or fine-tuning. You typically replace or modify the final layers of the model to suit the new task while keeping the initial layers, which capture general features, intact.

# Introduction

If not differently specified the source of picture are reported in the Reference and Credits section of this presentation.

# Pre-trained Language Models

# What are pre-trained language models?

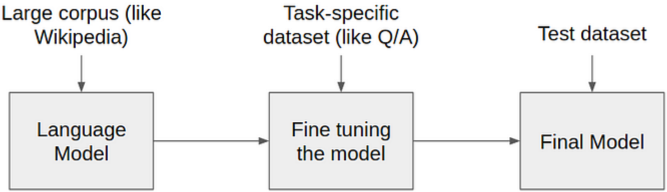
- Pre-trained language models are machine learning models that have been trained on large amounts of text data and can be fine-tuned for specific natural language processing (NLP) tasks.
- These models learn general language features, such as grammar, syntax, and semantics, which can be adapted to various NLP tasks, such as sentiment analysis, named entity recognition, and text summarization.

# What are pre-trained language models?

- Some popular pre-trained language models include:
- - BERT (Bidirectional Encoder Representations from Transformers)
- - GPT-3 (Generative Pre-trained Transformer 3)
- - RoBERTa (Robustly optimized BERT approach)
- - T5 (Text-to-Text Transfer Transformer)
- - OpenAI Codex

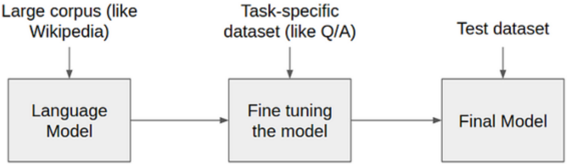
# What are pre-trained language models?

The intuition behind pre-trained language models is to create a black box which understands the language and can then be asked to do any specific task in that language.


Large corpus (like Wikipedia)
Task-specific dataset (like Q/A)
Test dataset
Language Model
Fine tuning the model
Final Model

# What are pre-trained language models?

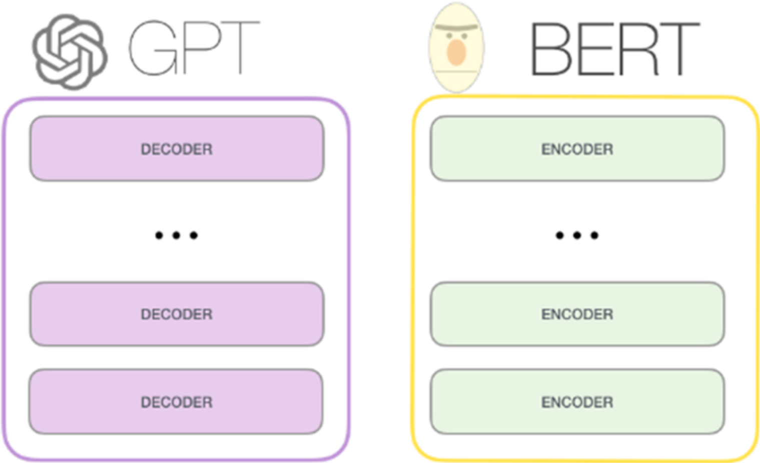
- The language model is first fed a large amount of unannotated data (for example, the complete Wikipedia dump).
- This lets the model learn the usage of various words and how the language is written in general.
- The model is now transferred to an NLP task where it is fed another smaller task-specific dataset, which is used to fine tune and create the final model capable of performing the aforementioned task.


Large corpus (like Wikipedia)
Task-specific dataset (like Q/A)
Test dataset
Language Model
Fine tuning the model
Final Model

# Transformers, GPT and BERT

- A transformer uses Encoder stack to model input, and uses Decoder stack to model output (using input information from encoder side).
- But if we do not have input, we just want to model the ”next word”, we can get rid of the Encoder side of a transformer and output ”next word” one by one. This gives us GPT.
- If we are only interested in training a language model for the input for some other tasks, then we do not need the Decoder of the transformer, that gives us BERT.

# Transformers, GPT and BERT



# BERT

## Bidirectional Encoder Representations from Transformers

Subsection 1

## Introduction to BERT

# Introduction to BERT

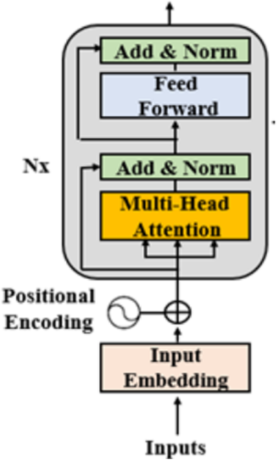
- BERT, or Bidirectional Encoder Representations from Transformers, is a groundbreaking natural language processing (NLP) model that has reshaped the landscape of language understanding in AI.
- Developed by Google's AI Research team in 2018, BERT represents a significant leap in the field of NLP due to its unique architecture and training approach.
- At its core, BERT is a transformer-based model, a neural network architecture that excels at capturing contextual relationships in language.

# Introduction to BERT

- What sets BERT apart is its **bidirectional training**, allowing it to consider both the left and right context of a word simultaneously, thereby gaining a deep understanding of word meanings and context.
- BERT's capabilities are largely attributed to its **pre-training** on massive text corpora, comprising vast amounts of web text.
- During pre-training, BERT learns to predict masked words within sentences, forcing it to understand context, grammar, and semantics.

# Introduction to BERT

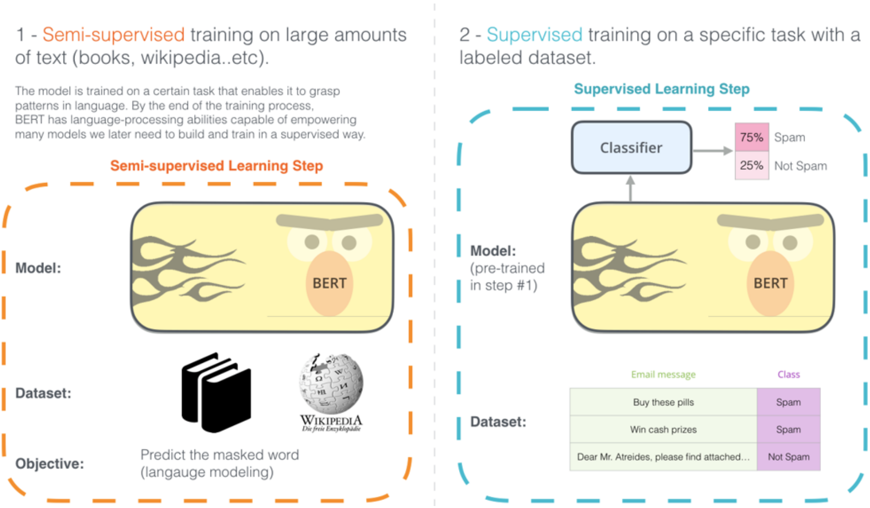
- The BERT model does not use decoder layers. A BERT model has an encoder stack but no decoder stacks.
- The masked tokens (hiding the tokens to predict) are in the attention layers of the encoder.



# Introduction to BERT

- BERT embodies the essence of **transfer learning**.
- After pre-training on extensive data, it can be fine-tuned on specific NLP tasks with relatively small task-specific datasets.
- This transferability of knowledge has proven invaluable for a myriad of NLP applications.
- BERT's versatility extends to multiple languages. It can be fine-tuned for various languages, making it a powerful tool for cross-lingual NLP tasks.
- **BERT is open source**, allowing researchers and developers worldwide to build upon its architecture and leverage pre-trained models for their own applications.

# Introduction to BERT



1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

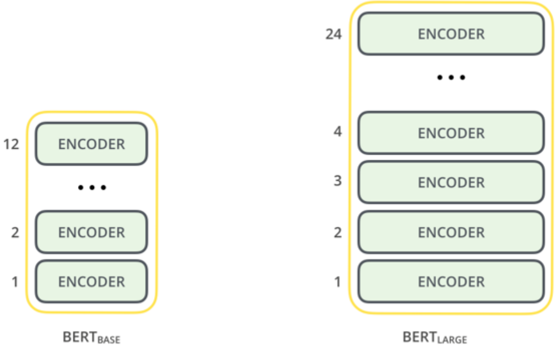
The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

2 - Supervised training on a specific task with a labeled dataset.

| Email message | Class |
|---|---|
| Buy these pills | Spam |
| Win cash prizes | Spam |
| Dear Mr. Atreides, please find attached… | Not Spam |

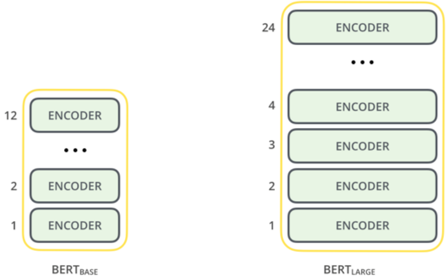
# Introduction to BERT

Basically BERT model cames into two sizes that have a large number of encoder layers (which the original paper calls Transformer Blocks) - twelve for the Base version, and twenty four for the Large version.

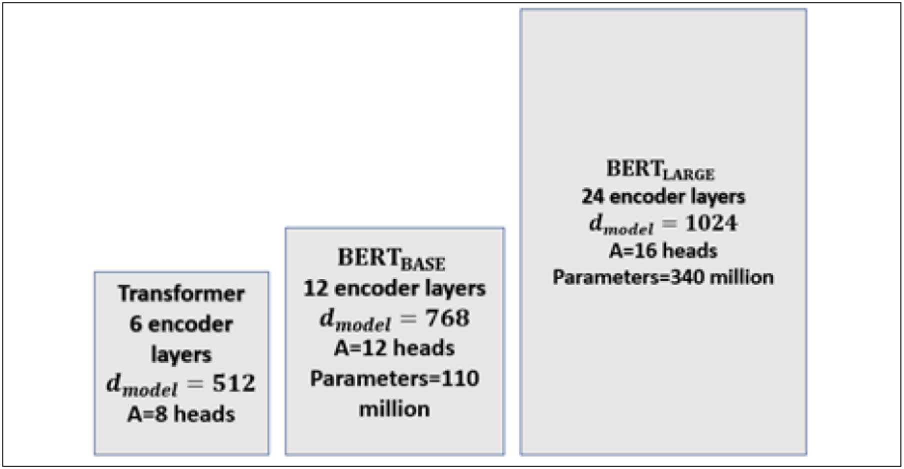


# Introduction to BERT

These also have larger feedforward-networks (768 and 1024 hidden units respectively), and more attention heads (12 and 16 respectively) than the default configuration in the reference implementation of the Transformer in the initial paper (6 encoder layers, 512 hidden units, and 8 attention heads).



# Introduction to BERT



Transformer
6 encoder layers
$d_{model} = 512$
A=8 heads

BERT$_{\text{BASE}}$
12 encoder layers
$d_{model} = 768$
A=12 heads
Parameters=110 million

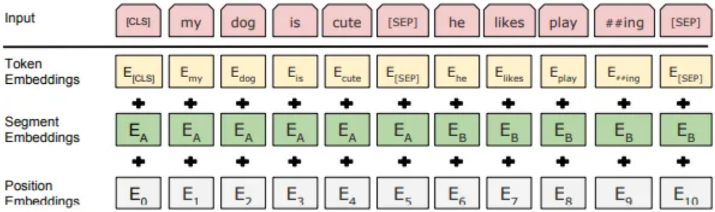
BERT$_{\text{LARGE}}$
24 encoder layers
$d_{model} = 1024$
A=16 heads
Parameters=340 million

Subsection 2

## BERT Input

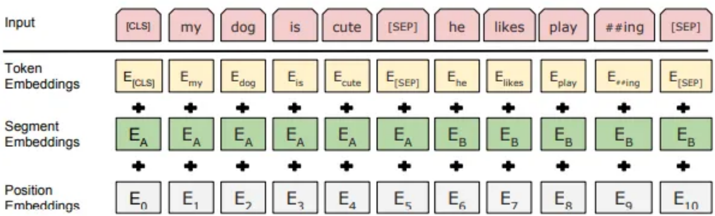
# BERT Input

- Given a sequence of tokens X = (x1, x2, . . . , xn), BERT pads the input sentence with two special [CLS] and [SEP] tokens.
- Then it trains an encoder that produces a contextualized vector representation for each token.
- Finally, the representation of each token in the sequence is constructed by summing the corresponding token, segment, and position embeddings.


Input
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
Token Embeddings
$E_{[CLS]}$ $E_{my}$ $E_{dog}$ $E_{is}$ $E_{cute}$ $E_{[SEP]}$ $E_{he}$ $E_{likes}$ $E_{play}$ $E_{\#\#ing}$ $E_{[SEP]}$
+ + + + + + + + + + +
Segment Embeddings
$E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_B$ $E_B$ $E_B$ $E_B$ $E_B$
+ + + + + + + + + + +
Position Embeddings
$E_0$ $E_1$ $E_2$ $E_3$ $E_4$ $E_5$ $E_6$ $E_7$ $E_8$ $E_9$ $E_{10}$

# BERT Input

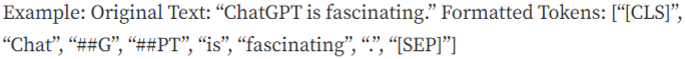
- The **[SEP]** token indicates when the next sentence starts for the NSP task.
- The **[CLS]** token is added to sequence A and sequence B to form the input, where the target of [CLS] is whether sequence B indeed follows sequence A in the corpus.
- **[CLS]** stands for **classification** tasks and **it supposes to summarize the sentence and the output of the last layer of this token will be used for the classification step**.



| Input | [CLS] | my | dog | is | cute | [SEP] | he | likes | play | ##ing | [SEP] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Token Embeddings | $E_{[CLS]}$ | $E_{my}$ | $E_{dog}$ | $E_{is}$ | $E_{cute}$ | $E_{[SEP]}$ | $E_{he}$ | $E_{likes}$ | $E_{play}$ | $E_{\#\#ing}$ | $E_{[SEP]}$ |
| | + | + | + | + | + | + | + | + | + | + | + |
| Segment Embeddings | $E_A$ | $E_A$ | $E_A$ | $E_A$ | $E_A$ | $E_A$ | $E_B$ | $E_B$ | $E_B$ | $E_B$ | $E_B$ |
| | + | + | + | + | + | + | + | + | + | + | + |
| Position Embeddings | $E_0$ | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | $E_7$ | $E_8$ | $E_9$ | $E_{10}$ |

# BERT: Tokenization

- As other LLM, BERT needs text to be broken down into smaller units called tokens.
- But here‘s the twist: BERT uses **WordPiece** tokenization.
- It splits words into smaller pieces, like turning ”running” into ”run” and ”ning.”
- This helps handle tricky words and ensures that BERT doesn‘t get lost in unfamiliar words.
- We add special tokens like [CLS] (stands for classification) at the beginning and [SEP] (stands for separation) between sentences (see next slides on the second form of BERT Training).



Example: Original Text: “ChatGPT is fascinating.” Formatted Tokens: [“[CLS]”, “Chat”, “##G”, “##PT”, “is”, “fascinating”, “.”, “[SEP]”]

Subsection 3

## BERT Pre-Training

# BERT Pre-training

- The authors of BERT came up with bidirectional attention, letting an attention head attend to all of the words both from left to right and right to left.
- In other words, the self-attention mask of an encoder could do the job without being hindered by the masked multi-head attention sub-layer of the decoder.

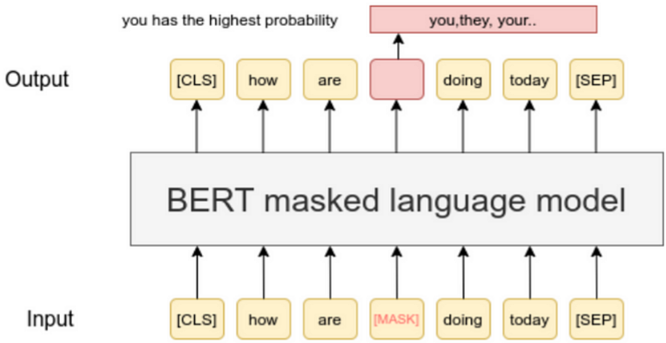
# BERT Pre-training

The model was trained with two tasks.

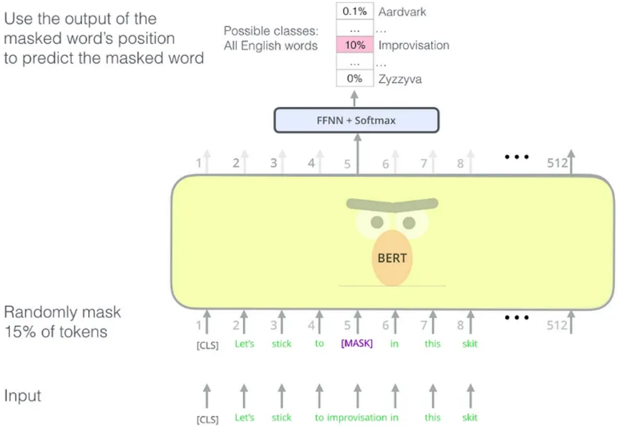
- The first method is **Masked Language Modeling (MLM)**.
- The second method is **Next Sentence Prediction (NSP)**.

# BERT Pretraining: Masked Language Model

- During its training, some words are masked (replaced with [MASK]) in sentences, and BERT learns to predict those words from their context.
- This helps BERT grasp how words relate to each other, both before and after.



# BERT Pretraining: Masked Language Model



Use the output of the masked word's position to predict the masked word

Possible classes: All English words

| | |
|---|---|
| 0.1% | Aardvark |
| ... | ... |
| 10% | Improvisation |
| ... | ... |
| 0% | Zyzzyva |

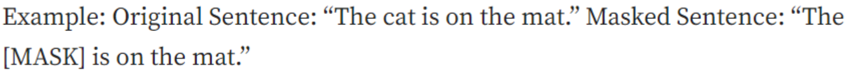
FFNN + Softmax

Randomly mask 15% of tokens

Input

# BERT Pretraining: Masked Language Model

- As we have said, during its training, some words are masked (replaced with [MASK]) in sentences, and BERT learns to predict those words from their context.
- This helps BERT grasp how words relate to each other, both before and after.



Example: Original Sentence: “The cat is on the mat.” Masked Sentence: “The [MASK] is on the mat.”

# BERT Pretraining: Masked Language Model

The input tokens were masked in a tricky way to force the model to train longer but produce better results with three methods:

- **Surprise** the model by not masking a single token on 10% of the dataset; for example: "the cat sat on it [because] it was a nice rug"
- **Surprise** the model by replacing the token with a random token on 10% of the dataset; for example: "the cat sat on it [often] it was a nice rug"
- **Replace** a token by a [MASK] token on 80% of the dataset; for example: "the cat sat on it [MASK] it was a nice rug"

# BERT Pretraining: Next Sentence Prediction

- The second method found to train BERT is Next Sentence Prediction (NSP).
- The input contains two sentences.
- BERT doesn't just understand words; it grasps the flow of sentences.
- In the NSP objective, BERT is trained to predict if one sentence follows another in a text pair.
- This helps BERT comprehend the logical connections between sentences, making it a master at understanding paragraphs and longer texts.

# BERT Pretraining: Next Sentence Prediction

Two artificial tokens are very important:

- **[CLS]** is a binary classification token added to the beginning of the first sequence to predict if the second sequence follows the first sequence. A positive sample is usually a pair of consecutive sentences taken from a dataset. A negative sample is created using sequences from different documents.
- **[SEP]** is a separation token that signals the end of a sequence.

# BERT Pretraining: Next Sentence Prediction

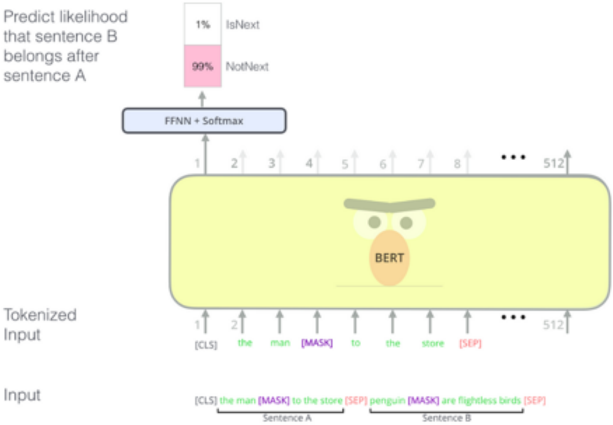
- For example, the input sentences taken out of a book could be:

  `"The cat slept on the rug.  It likes sleeping all day."`

- These two sentences would become one input complete sequence:

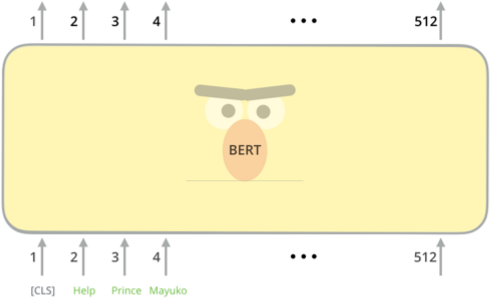
  `"[CLS] the cat slept on the rug [SEP] it likes sleep ##ing all day[SEP]"`

# BERT Pretraining: Next Sentence Prediction


Predict likelihood that sentence B belongs after sentence A
1%
IsNext
99%
NotNext
FFNN + Softmax
1
2
3
4
5
6
7
8
• • •
512
BERT
Tokenized Input
[CLS]
the
man
[MASK]
to
the
store
[SEP]
Input
[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]
Sentence A
Sentence B

# Remark: The special CLS token

- As we have seen, the first input token is always supplied with a special **[CLS]** token;
- **CLS** here stands for **Classification**.



# Remark: The special CLS token

- In order to better understand the role of [CLS] let's recall that BERT model has been trained on 2 main tasks:
  - **Masked language modeling**: some random words are masked with [MASK] token, the model learns to predict those words during training. For that task we need the [MASK] token.
  - **Next sentence prediction**: given 2 sentences, the model learns to predict if the 2nd sentence is the real sentence, which follows the 1st sentence. For this task, we need another token, output of which will tell us how likely the current sentence is the next sentence of the 1st sentence. And here comes the [CLS].
- You can think about the output of [CLS] as BERT's understanding at the sentence-level.
- This is why when we are interested in **classification problems** we are interested to the embedding representation of this single token and we can discharge all the others.

# BERT: Pre-training

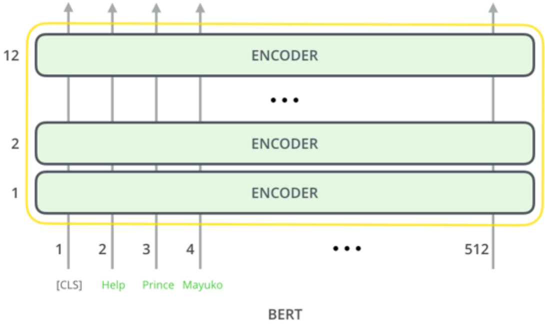
After BERT is trained on these 2 tasks, the learned model can be then used as a feature extractor for different NLP problems, where we can either keep the learned weights fixed and just learn the newly added task-specific layers or fine-tune the pre-trained layers too.

Subsection 4

## BERT Inference

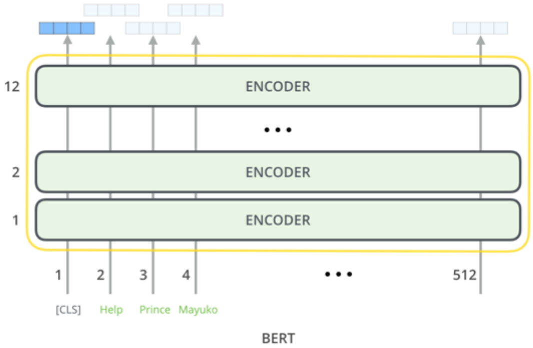
# BERT: Elaboration

- Just like the vanilla encoder of the transformer, BERT takes a sequence of words as input which keep flowing up the stack.
- Each layer applies self-attention, and passes its results through a feed-forward network, and then hands it off to the next encoder.



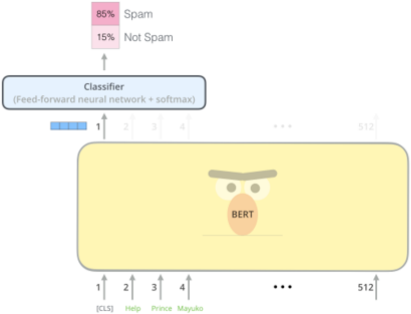
# BERT: Model Outputs

- Each position outputs a vector of size hidden_size (768 in BERT Base).
- For the sentence classification example, **we will focus on the output of only the first position** (that we passed the **special [CLS] token** to).



# BERT: Model Outputs

- That vector can now be used as the input for a classifier of our choosing.
- If you have more labels (for example if you‘re an email service that tags emails with ”spam”, ”not spam”, ”social”, and ”promotion”), you just tweak the classifier network to have more output neurons that then pass through softmax.



# GPT

## Generative Pre-trained Transformer

Subsection 1

## Introduction

# Introduction to GPT

- GPT stands for (G)enerative (P)re-trained (T)ransformer.
- By being generative, it means that it can generate new text, given an input of text. For example, if we give the model the following text: "The sky is", the model should be able to predict that the next word is "blue".
- If I give it another sentence such as "the quick brown fox", the model would first make a prediction "jumped," then using the previous sentence ("The quick brown fox jumped"), it should predict the word "over," and so on.

# Introduction to GPT

- ”Pre-trained” means that the model was simply trained on a large amount of text to model probabilities without any other purpose than language modeling.
- GPT models can then be fine-tuned, i.e., further trained, to perform more specific tasks.
- For instance, you can use a small dataset of news summaries to obtain a GPT model very good at news summarization.
- Or fine-tune it on French-English translations to obtain a machine translation system capable of translating from French to English.

# Introduction to GPT: GPT-3

- GPT-3 was announced in 2020. With its 175 billion parameters, it was an even bigger jump from GPT-2 than GPT-2 from the first GPT.
- This is also from GPT-3 that **OpenAI stopped to disclose precise training information about GPT models**.
- Today, there are 7 GPT-3 models available through OpenAI's API but we only know little about them.
- With GPT-3, OpenAI demonstrated that GPT models can be extremely good for specific language generation tasks if the users provide a few examples of the task they want the model to achieve.

# Introduction to GPT: GPT-3.5

- With the GPT-3 models running in the API and attracting more and more users, OpenAI could collect a very large dataset of user inputs.
- They exploited these inputs to further improve their models.
- They used a technique called reinforcement learning from human feedback (RLHF). I won‘t explain the details here but you can find them in a blog post published by OpenAI (https://openai.com/research/learning-from-human-preferences).
- In a nutshell, thanks to RLHF, GPT-3.5 is much better at following user instructions than GPT-3. OpenAI denotes this class of GPT models as "instructGPT".

# Introduction to GPT: GPT-3.5

- With GPT-3.5, you can "prompt" the model to perform a specific task without the need to give it examples of the task.
- You just have to write the "right" prompt to get the best result.
- This is where "prompt engineering" becomes important and why skilled prompt engineers are receiving incredible job offers.

# Introduction to GPT: GPT-4

- GPT-4 has been released in March 2023.
- **We know almost nothing about its training**.
- The main difference with GPT-3/GPT-3.5 is that GPT-4 is bimodal: It can take as input images and text.
- It can generate text but won't directly generate images. Note: GPT-4 can generate the code that can generate an image, or retrieve one from the Web.

# Introduction to GPT: chatGPT

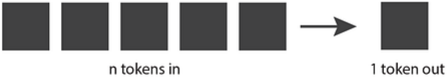
- ChatGPT is just a user interface with chat functionalities. When you write something with ChatGPT, it‘s a GPT-3.5 (or 4) model that generates the answer.
- A particularity of ChatGPT is that it‘s not just taking as input the current query of the user as an out-of-the-box GPT model would do. To properly work as a chat engine, ChatGPT must keep track of the conversation: What has been said, what is the user goal, etc.
- OpenAI **didn‘t disclose how it does that**. Given that GPT models can only accept a prompt of a limited length, ChatGPT can‘t just concatenate all the dialogue turns together to put them in the same prompt. This kind of prompt could be way too large to be handled by GPT-3.5.

## Subsection 2

## How GPT Models Work

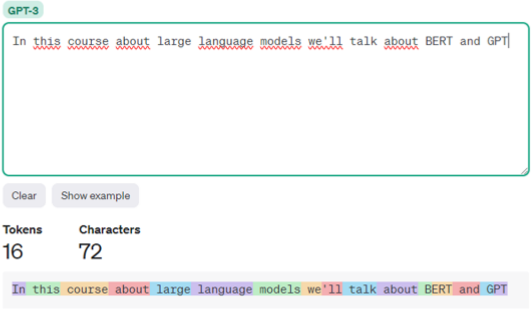
# How GPT Models Work

Let‘s start by exploring how generative language models work. The very basic idea is the following: they take n tokens as input, and produce one token as output.



# How GPT Models Work

- In order to understand the precise meaning of "token" for GPT, you can go to OpenAI's Tokenizer page, enter your text, and see how it gets split up into tokens.
- You can choose between "GPT-3" tokenization, which is used for text, and "Codex" tokenization, which is used for code. We'll keep the default "GPT-3" setting.



In this course about large language models we'll talk about BERT and GPT

Tokens 16

Characters 72

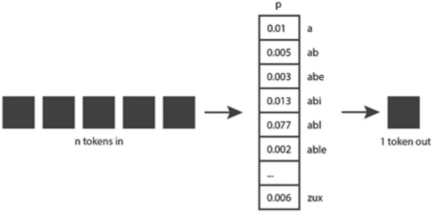
In this course about large language models we'll talk about BERT and GPT

# How GPT Models Work

- But if you‘ve played with OpenAI‘s ChatGPT, you know that it produces many tokens, not just a single token.
- That‘s because this basic idea is applied in an expanding-window pattern.
- You give it *n* tokens in, it produces one token out, then it incorporates that output token as part of the input of the next iteration, produces a new token out, and so on.
- This pattern keeps repeating until a stopping condition is reached, indicating that it finished generating all the text you need.
- While playing with ChatGPT, you may also have noticed that the model is not deterministic: if you ask it the exact same question twice, you‘ll likely get two different answers.

# How GPT Models Work

- That‘s because the model doesn”t actually produce a single predicted token; instead it returns a probability distribution over all the possible tokens.
- In other words, it returns a vector in which each entry expresses the probability of a particular token being chosen. The model then samples from that distribution to generate the output token.



# How GPT Models Work

- How does the model come up with that probability distribution?
- That‘s what the training phase is for.
- During training, the model is exposed to a lot of text, and its weights are tuned to predict good probability distributions, given a sequence of input tokens.
- GPT models are trained with a large portion of the internet, so their predictions reflect a mix of the information they‘ve seen.

# How GPT Models Work

- the T in GPT stands for transformer;
- As we already known, transformers are based on the "attention mechanism," which allows the model to pay more attention to some inputs than others, regardless of where they show up in the input sequence.
- For example, let's consider the following sentence:
- "She went to the store and ... [bought]"

# How GPT Models Work

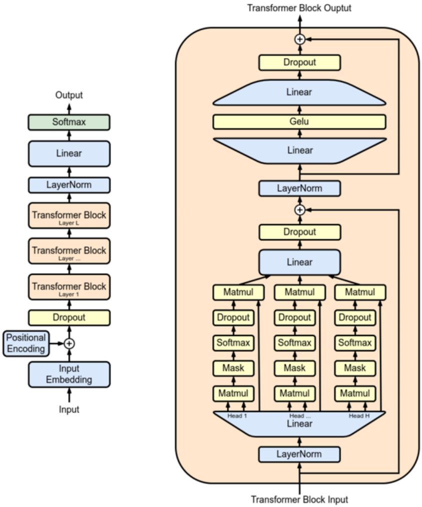
- In this scenario, when the model is predicting the verb "bought," it needs to match the past tense of the verb "went."
- In order to do that, it has to pay a lot of attention to the token "went". In fact, it may pay more attention to the token "went" than to the token "and", despite the fact that "went" appears much earlier in the input sequence.
- This selective attention behavior in GPT models is enabled by the use of a **masked multi-head attention** layer.
- Remember that **GPT use only the DECODER block** of the transformer architecture.

Subsection 3

## GPT Implementation and Architecture

# GPT Architecture

- Remember that the attention layer is ”masked” if the matrix is restricted to the relationship between each token position and earlier positions in the input.
- This is what GPT models use for text generation, because an output token can only depend on the tokens that come before it.



# GPT Implementation

- At the time of writing, the three latest text generation models released by OpenAI are GPT-3.5, ChatGPT, and GPT-4, and they are all based on the Transformer architecture.
- GPT-3.5 is a transformer trained as a completion-style model, which means that if we give it a few words as input, it‘s capable of generating a few more words that are likely to follow them in the training data.

# GPT Implementation

- ChatGPT, on the other hand, is trained as a conversation-style model, which means that it performs best when we communicate with it as if we‘re having a conversation.
- It‘s based on the same transformer base model as GPT-3.5, but it”s fine-tuned with conversation data.
- Then it‘s further fine-tuned using Reinforcement Learning with Human Feedback (RLHF), which is a technique that OpenAI introduced in their 2022 InstructGPT paper.

# GPT Implementation

- In this technique, we give the model the same input twice, get back two different outputs, and ask a human ranker which output it prefers.
- That choice is then used to improve the model through fine-tuning.
- This technique brings alignment between the outputs of the model and human expectations, and it's critical to the success of OpenAI's latest models.

Subsection 4

## Reference and Credits

# References and Credits

- Beatriz Stollnitz, "How GPT Models Work", Toward Data Science
- Benjamin Marie, "A Gentle Introduction to GPT Models", Towards Data Science
- Vonage Dev Team, "Introduction to GPT-3", Medium
- Jay Alammar, "A Visual Guide to Using BERT for the First Time", https://jalammar.github.io/