This is CS4045

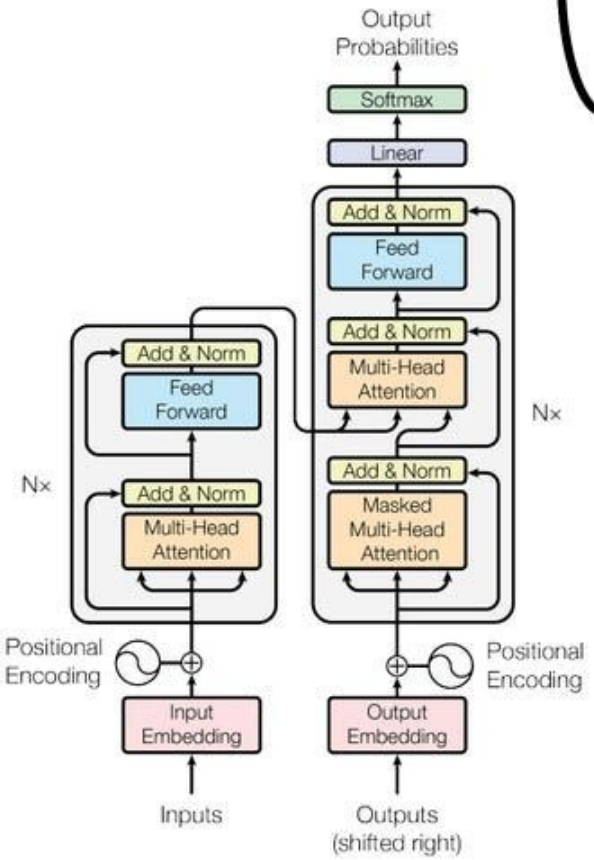
GCR : dxuxugo

References

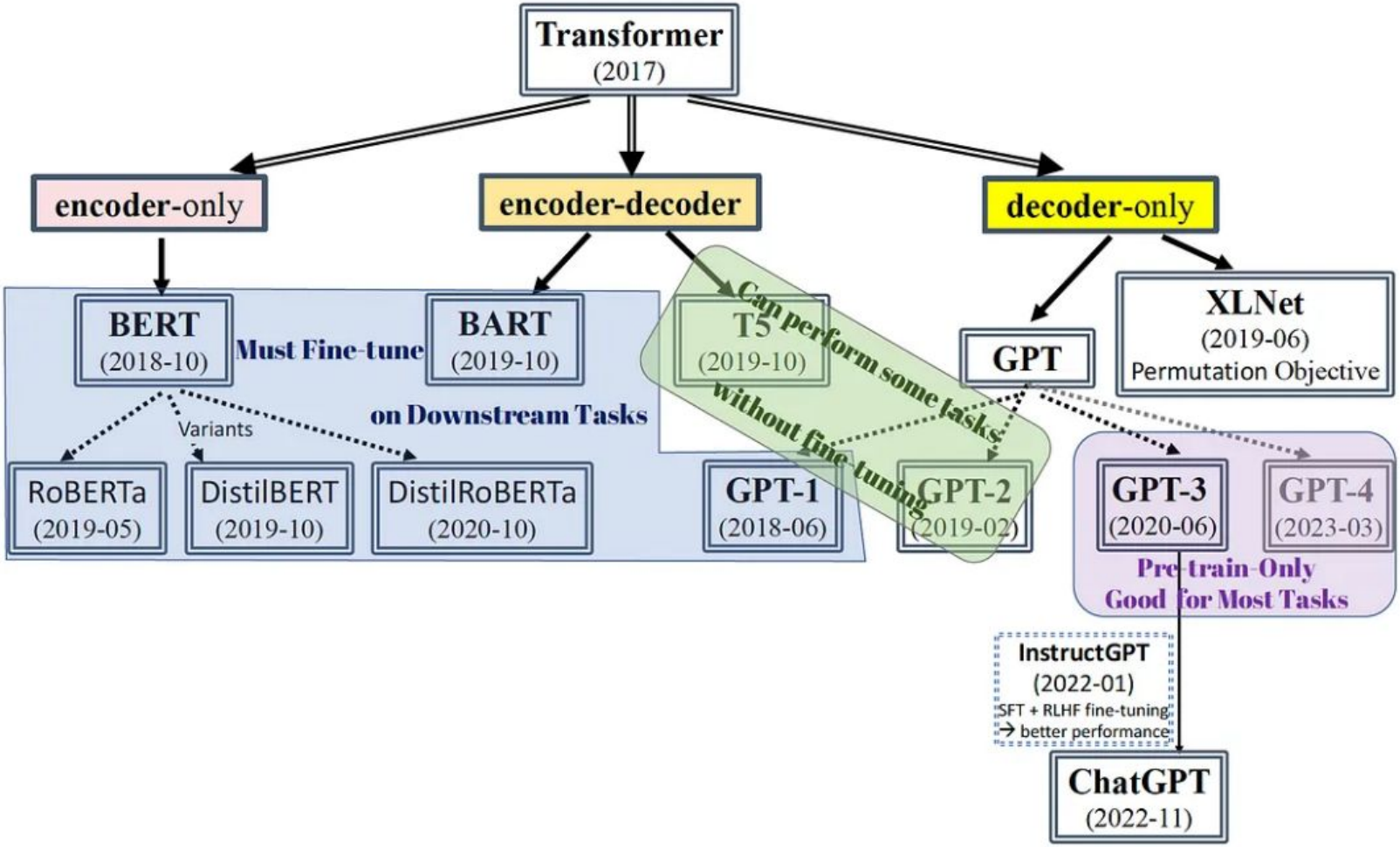
https://jalammar.github.io/illustrated-gpt2/

https://medium.com/nerd-for-tech/gpt3-and-chat-gpt-detailed-architecture-study-deep-nlp-horse-db3af9de8a5d

https://medium.com/the-modern-scientist/an-in-depth-look-at-the-transformer-based-models-22e5f5d17b6b



COME, LET US UNDERSTAND THE UNCOOL TRANSFORMERS!!! - OPTIMUS PRIME


Transformer
(2017)
encoder-only
encoder-decoder
decoder-only
BERT
(2018-10)
Must Fine-tune
on Downstream Tasks
BART
(2019-10)
T5
(2019-10)
Can perform some tasks without fine-tuning
GPT
XLNet
(2019-06)
Permutation Objective
Variants
RoBERTa
(2019-05)
DistilBERT
(2019-10)
DistilRoBERTa
(2020-10)
GPT-1
(2018-06)
GPT-2
(2019-02)
GPT-3
(2020-06)
GPT-4
(2023-03)
Pre-train-Only
Good for Most Tasks
InstructGPT
(2022-01)
SFT + RLHF fine-tuning
→ better performance
ChatGPT
(2022-11)

PRE-TRAINING OBJECTIVE

(a). Pre-training: Training on huge and diverse text datasets (such as Wikipedia, question-answering websites, literature books, etc.) to establish a broad and upper-level understanding of natural language patterns in an unsupervised manner.

(b). Fine-tuning: The pre-trained model is further trained on lower-level, more specific downstream tasks separately, such as sentiment analysis, text classification, named entity recognition, machine translation, and question-answering, etc.

However, the fine-tuning on downstream tasks requires the creation of carefully prepared datasets with corresponding labels and often involves modifying the fine-structure of the model, which demands significant labor force.

Pre Training Objective

However, our ultimate goal is to unify all downstream tasks into one solitary pre-training task, thus eliminating the need for any subsequent fine-tuning on separate tasks.

The choice of pre-training objectives can significantly impact the performance of a Transformer-based model and the degree of adaptation required for fine-tuning on specific tasks.

AR VS AE

Autoregressive (AR) and autoencoding (AE) are currently two successful types of objectives.

AR models focus on regenerating text sequences and are particularly skilled in text generation tasks such as machine translation, abstractive summarization and question-answering.

AE models aim to reconstruct the original text from corrupted text data and excel in language understanding.

Autoencoding (AE) models: BERT, BART, and T5;

Autoregressive (AR) models: GPT;

Encoder Only

Google

BERT

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

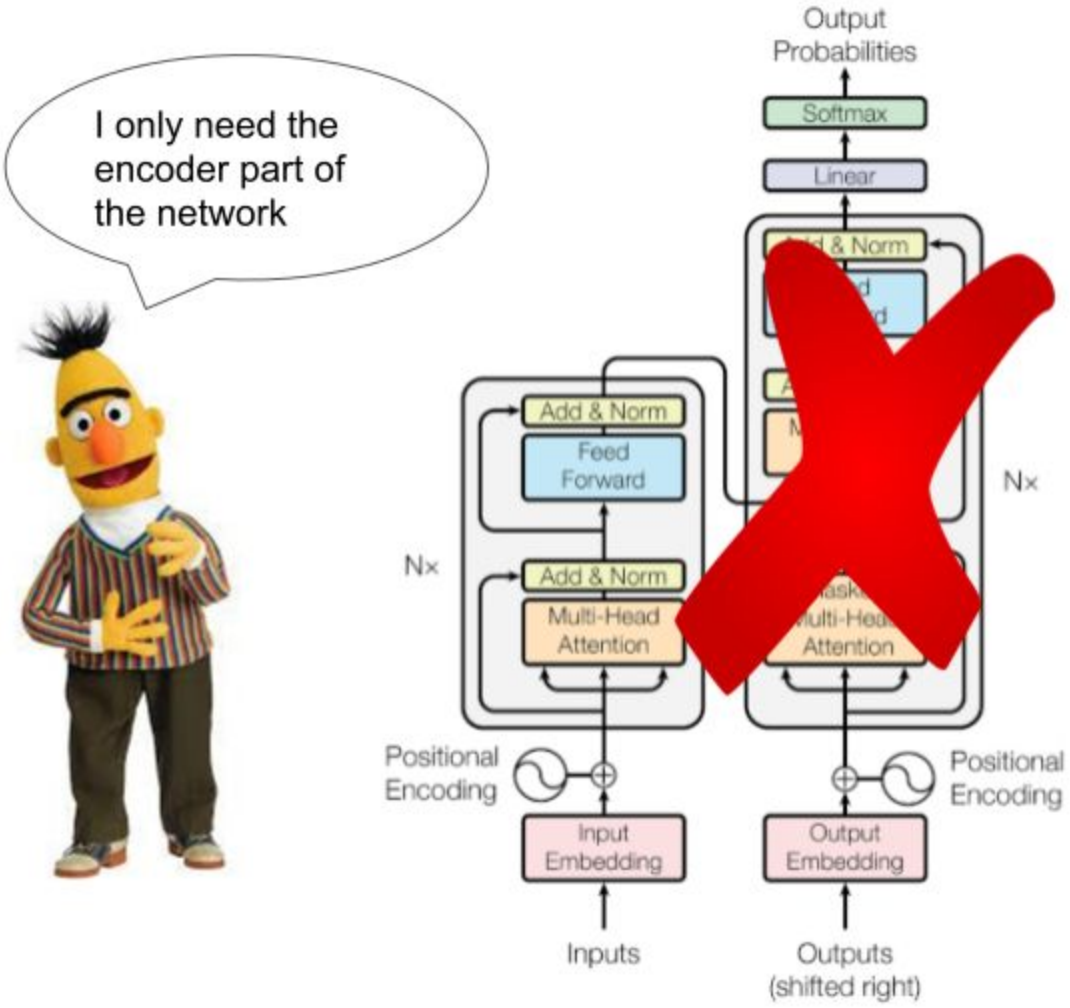
We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn

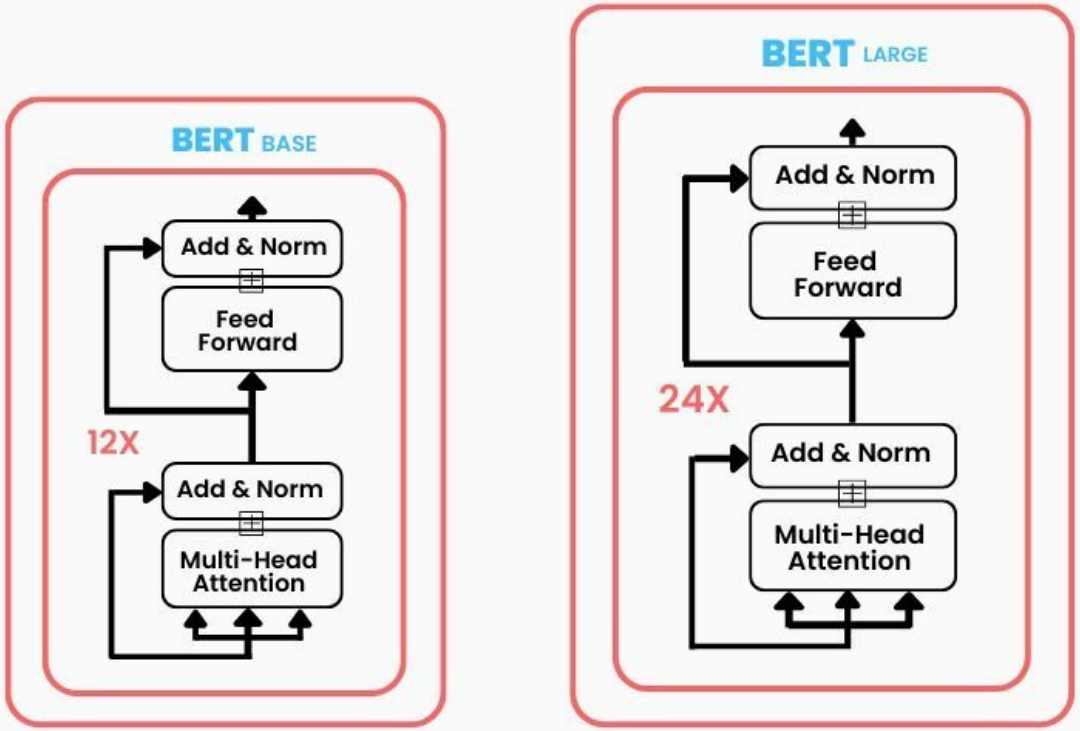
BERT (Bidirectional Encoder Representations from Transformers)

Google introduced BERT in 2018 as a bidirectional model that exclusively utilizes encoder stacks.

BERT outperformed GPT-1 (earlier 2018), which processes text data from left-to-right, in various tasks with the same number of parameters, largely due to its bidirectional processing capability.


I only need the encoder part of the network
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

BERT Size & Architecture



110M Parameters

340M Parameters

Turing

1. Masked LM (MLM)

One of the pre-training objectives of the BERT model is the denoising objective, which involves predicting 15% of randomly masked tokens.

The idea here is “simple”: Randomly mask out 15% of the words in the input — replacing them with a [MASK] token — run the entire sequence through the BERT attention based encoder and then predict only the masked words, based on the context provided by the other non-masked words in the sequence.

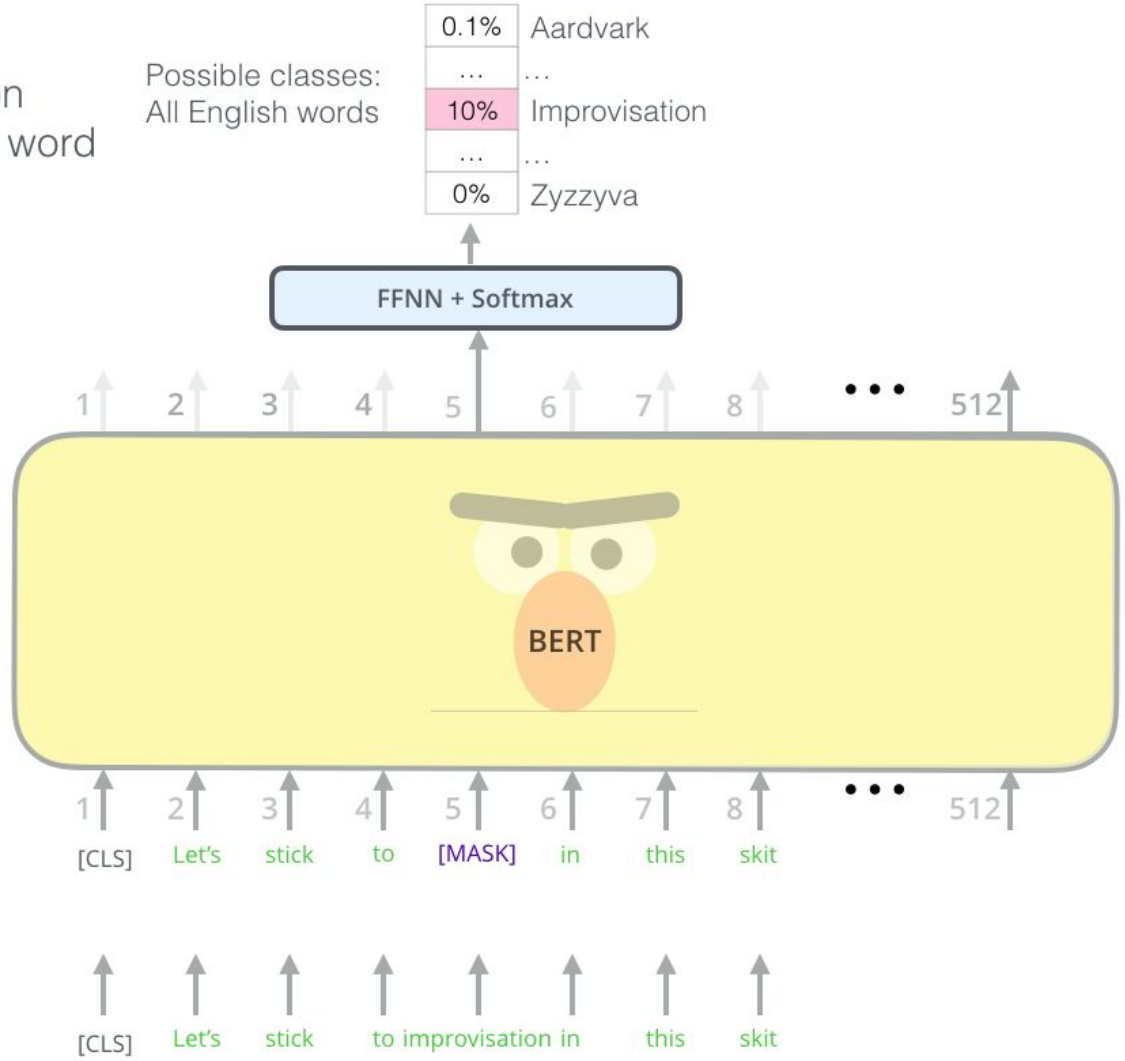
1. Masked LM (MLM)

The model only tries to predict when the [MASK] token is present in the input, while we want the model to try to predict the correct tokens regardless of what token is present in the input. To deal with this issue, out of the 15% of the tokens selected for masking:

- 80% of the tokens are actually replaced with the token [MASK].
- 10% of the time tokens are replaced with a random token.
- 10% of the time tokens are left unchanged.

While training the BERT loss function considers only the prediction of the masked tokens and ignores the prediction of the non-masked ones. This results in a model that converges much more slowly than left-to-right or right-to-left models.

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



Transformer VS BERT

Transformer: I am a ___

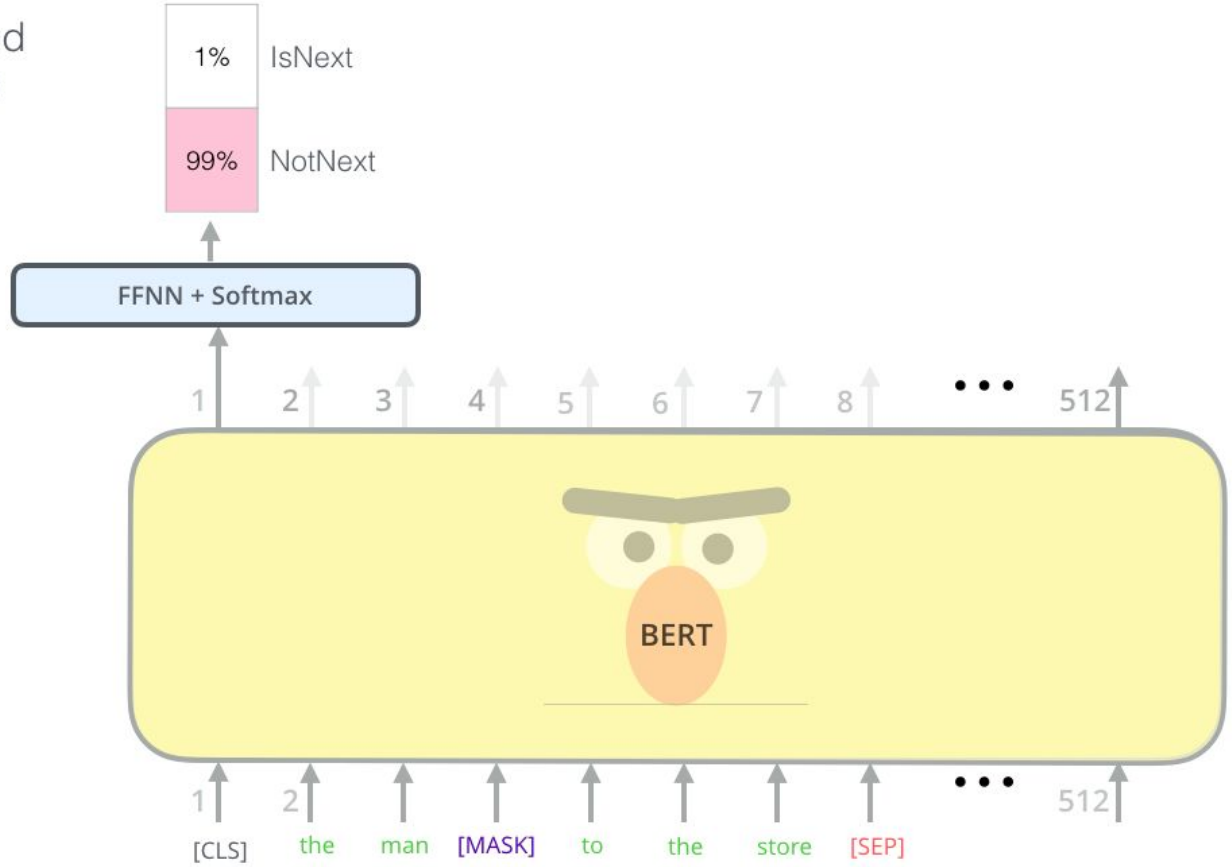
BERT: I am a ___. I like playing Cricket

2. Next Sentence Prediction (NSP)

In order to understand relationship between two sentences, BERT training process also uses next sentence prediction. During training the model gets as input pairs of sentences and it learns to predict if the second sentence is the next sentence in the original text as well.

- 50% of the time the second sentence comes after the first one.
- 50% of the time it is a random sentence from the full corpus.

Predict likelihood that sentence B belongs after sentence A



Tokenized Input

Input

[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Sentence A

Sentence B

2. Next Sentence Prediction (NSP)

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

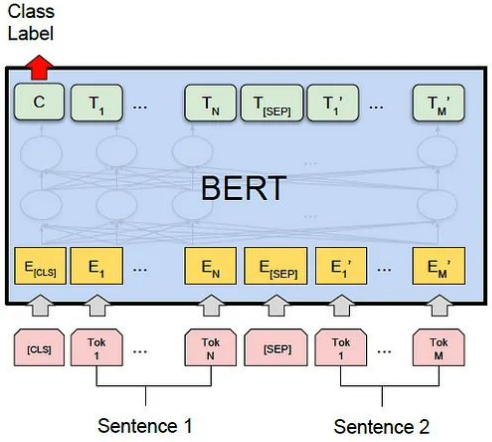
Label = NotNext

Fine Tuning

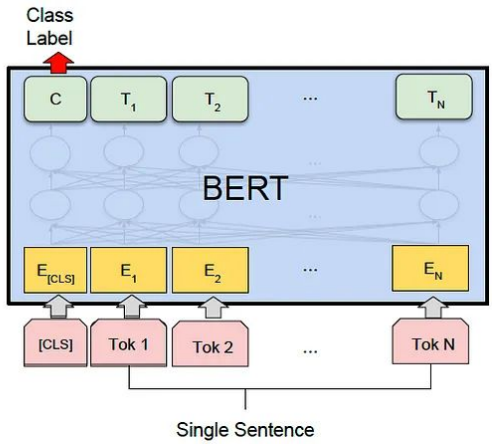
Real-life language tasks are not denoising tasks, leading to a discrepancy between pre-training and fine-tuning for BERT models. Therefore, fine-tuning is necessary for individual downstream tasks.

BERT categorizes downstream tasks into four types:

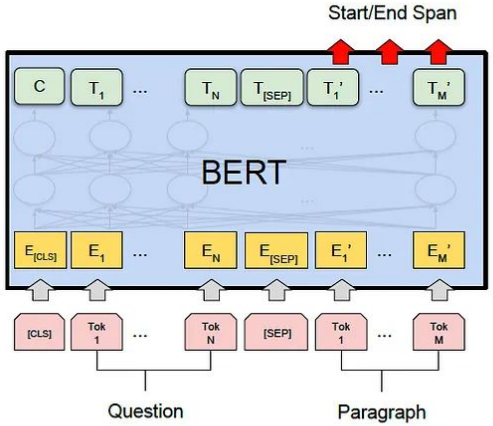
- Sentence Pair Classification Tasks (e.g., semantic similarity between two sentences)
- Single Sentence Classification Tasks (e.g., sentiment analysis)
- SQuAD (Question-Answering)
- Named Entity Tagging.



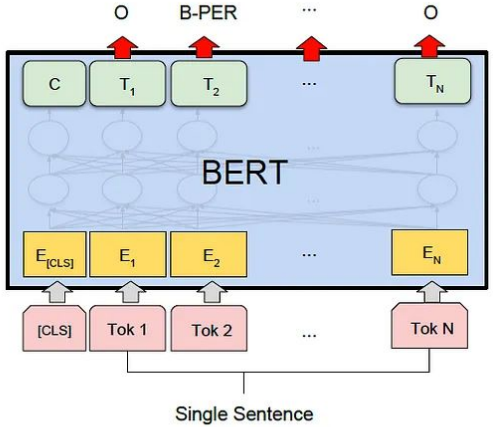
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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



(c) Question Answering Tasks: SQuAD v1.1



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

Architecture details

BERT-Base: 12-layer, 768-hidden-nodes, 12-attention-heads, 110M parameters

BERT-Large: 24-layer, 1024-hidden-nodes, 16-attention-heads, 340M parameters

Fun fact: BERT-Base was trained on 4 cloud TPUs for 4 days

BERT-Large was trained on 16 TPUs for 4 days!


I'm a token!
Vision Transformers
I'm a token!
A dog smiling

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*], **Xiaohua Zhai**[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby**[*,†]

[*]equal technical contribution, [†]equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

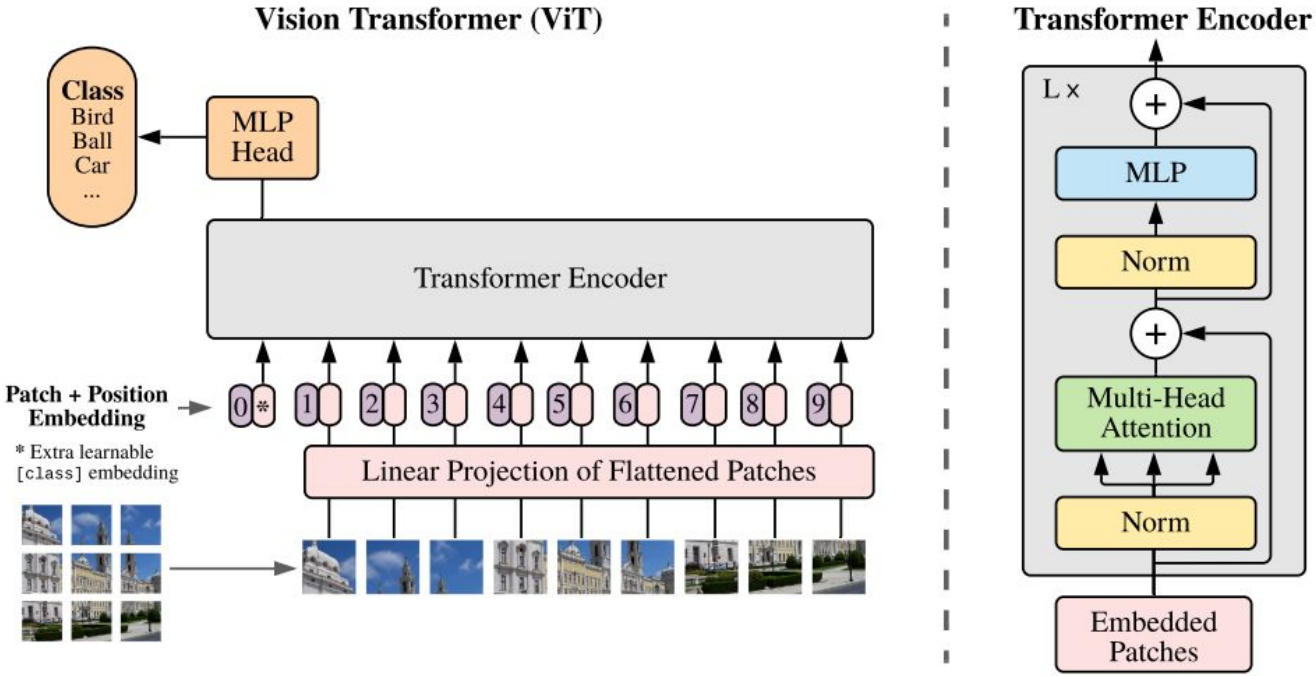
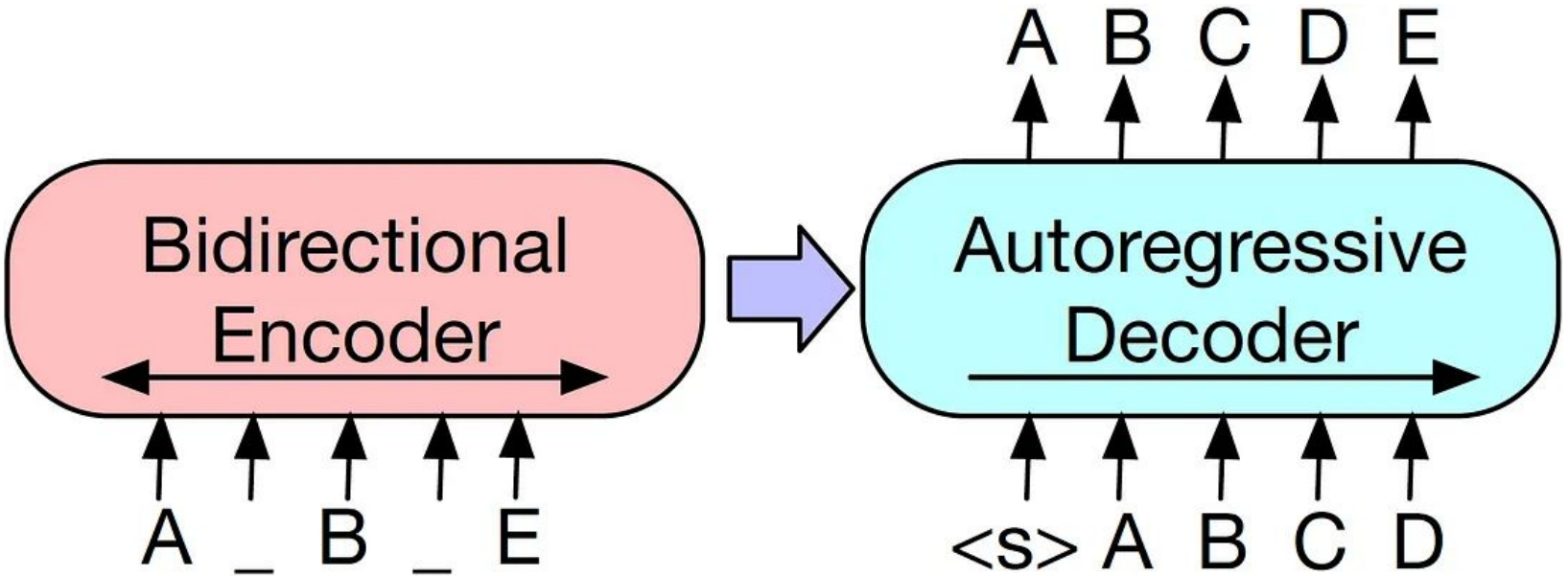


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Encoder Decoder

BART (Bidirectional and Auto-Regressive Transformers)



BART

BART = Bidirectional and Auto-Regressive Transformers

It’s a sequence-to-sequence model by Facebook (Meta), combining:

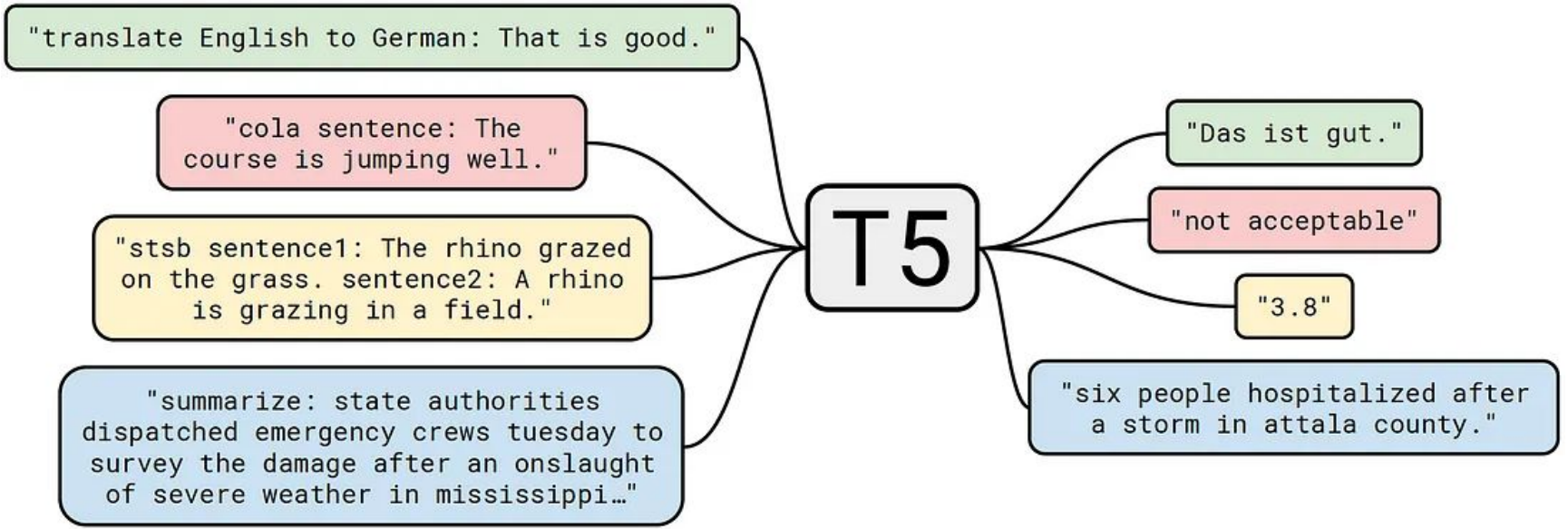
The encoder from BERT (fully bidirectional)

The decoder from GPT (autoregressive).

BART

BART (2019) proposed by Facebook greatly resembles T5 in terms of the denoising pre-training objective and the encoder-decoder architecture, with the only difference being that 30% of the tokens are masked and sentence permutation is used in the pre-training text.

T5 (Text-To-Text Transfer Transformer)



T5 (Text-To-Text Transfer Transformer)

In 2020, Google proposed T5 as a unified model capable of transforming all downstream tasks into text generative tasks, even classification problems.

T5 uses an encoder-decoder architecture and a denoising objective

During pre-training, 15% of the tokens fed into the encoder are randomly masked.

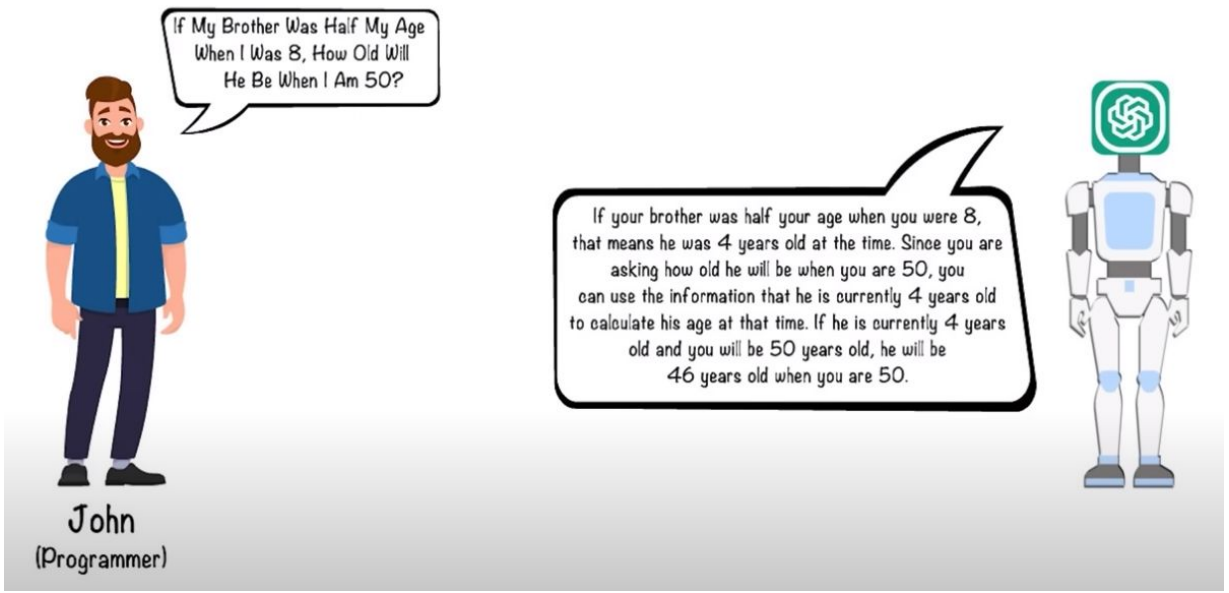
T5 (Text-To-Text Transfer Transformer)

It is a modified version of BERT's masking and denoising algorithm: consecutive masked tokens are replaced by, a sentinel and are treated as a new single token added to the original vocabulary.

This helps the model learn to predict how many words are missing in the blank.

Later, generative capability is learned by randomly splitting the input text in the dataset, with the first part fed into the encoder and the second part treated as the output to be auto-regressively regenerated.

Decoder Only



Improving Language Understanding by Generative Pre-Training

Improving Language Understanding by Generative Pre-Training

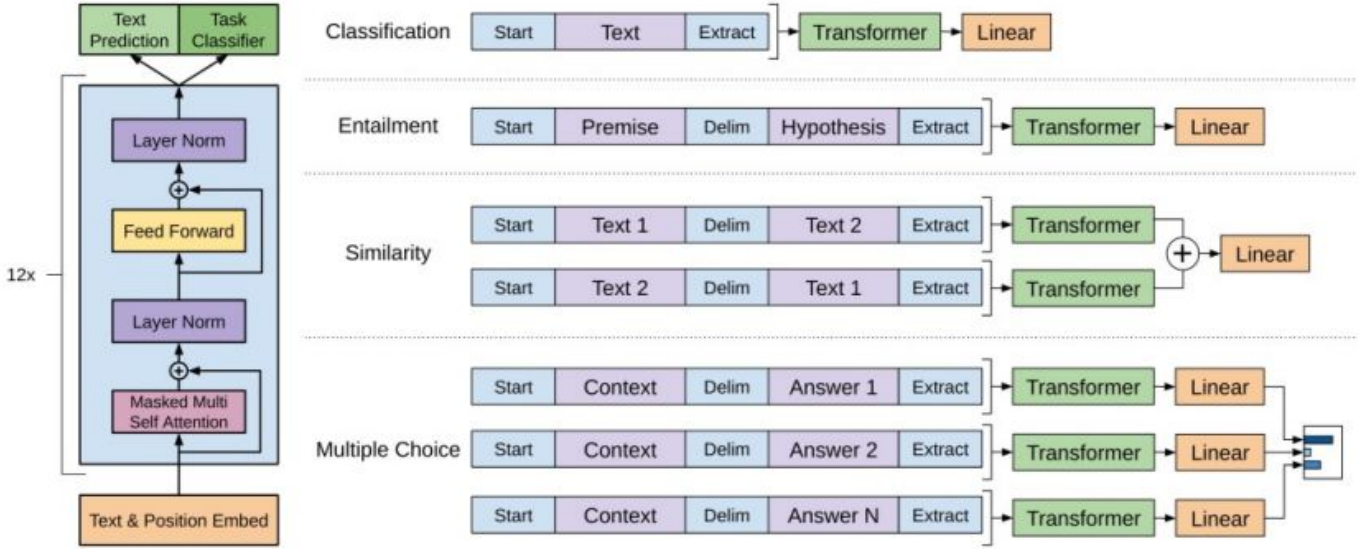
Alec Radford OpenAI alec@openai.com
Karthik Narasimhan OpenAI karthikn@openai.com
Tim Salimans OpenAI tim@openai.com
Ilya Sutskever OpenAI ilyasu@openai.com

Abstract

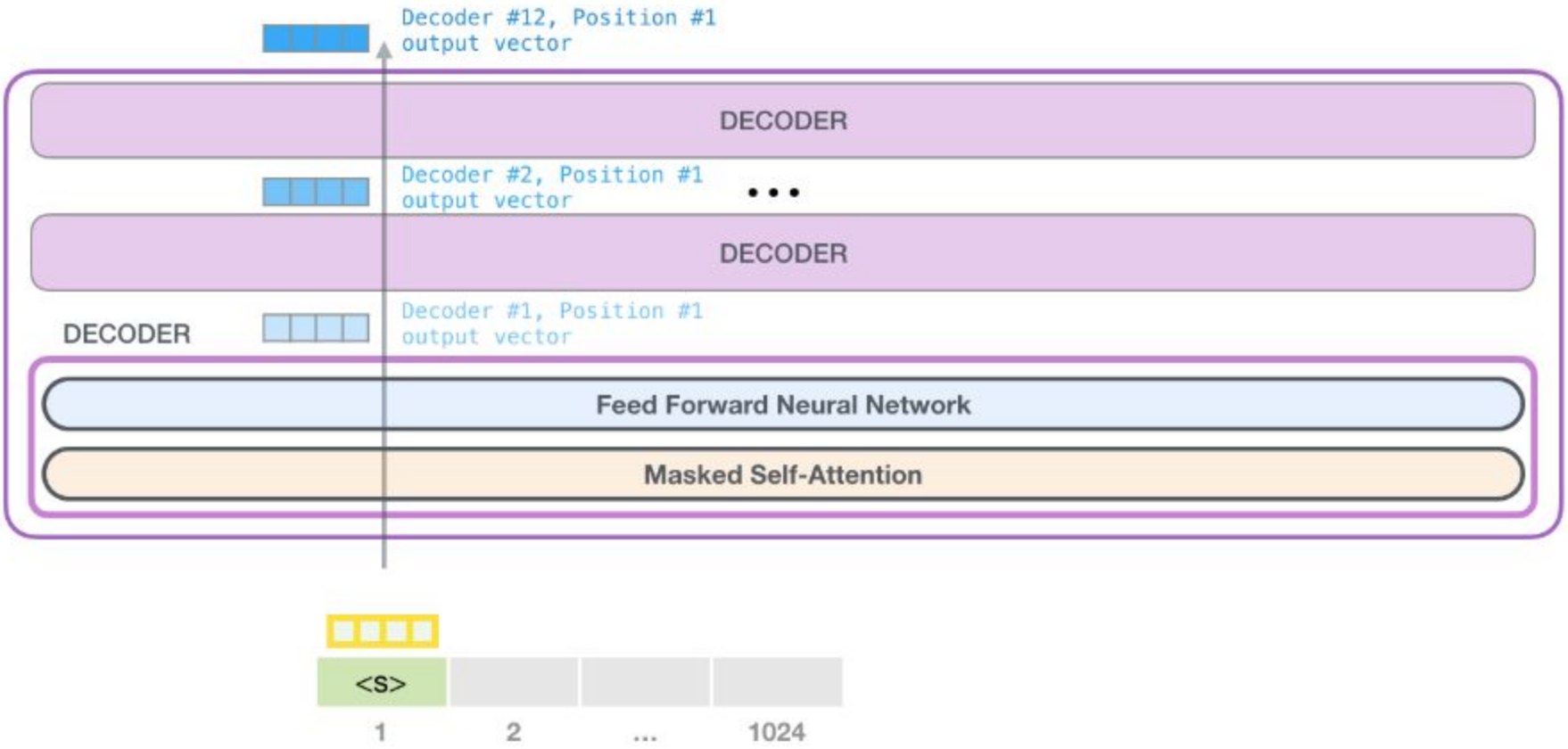
Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification. Although large unlabeled text corpora are abundant, labeled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. We demonstrate that large gains on these tasks can be realized by *generative pre-training* of a language model on a diverse corpus of unlabeled text, followed by *discriminative fine-tuning* on each specific task. In contrast to previous approaches, we make use of task-aware input transformations during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. We demonstrate the effectiveness of

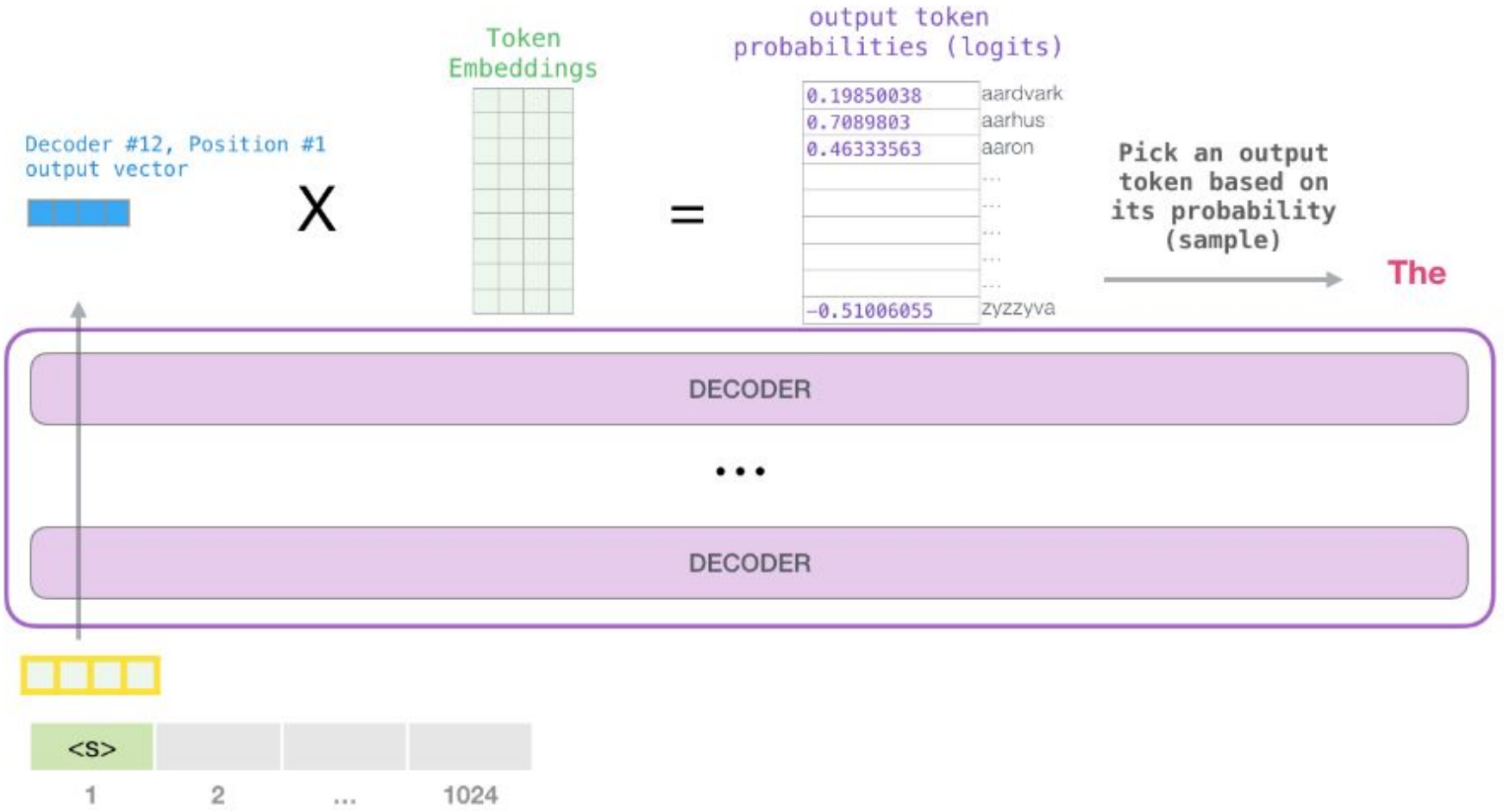
Generative Pretrained Transformer (GPT) [Radford et al., 2018]

How do we format inputs to our decoder for **finetuning tasks?**



The linear classifier is applied to the representation of the [EXTRACT] token.


Decoder #12, Position #1
output vector
DECODER
Decoder #2, Position #1
output vector
DECODER
DECODER
Decoder #1, Position #1
output vector
Feed Forward Neural Network
Masked Self-Attention
<s>
1
2
...
1024


Token Embeddings
output token probabilities (logits)
Decoder #12, Position #1 output vector
X
=
0.19850038 aardvark
0.7089803 aarhus
0.46333563 aaron
-0.51006055 zyzzyva
Pick an output token based on its probability (sample)
The
DECODER
DECODER
<s>
1
2
...
1024

GPT - 1

OpenAI proposed the GPT-1-4 models that only used decoder stacks, making them left-to-right autoregressive models.

GPT-1 (2018, 117 million parameters) did not exhibit emergent capabilities and heavily relied on fine-tuning for individual downstream tasks.

GPT - 2

GPT-2 (2019, 1.5 billion parameters) introduced the phenomenon of in-context learning for a few tasks, and improved its tokenizer by using Byte-level Encoding (BLE.

The GPT-2 was trained on a massive 40GB dataset called WebText that the OpenAI researchers crawled from the internet as part of the research effort.

The smallest variant of the trained GPT-2, takes up 500MBs of storage to store all of its parameters.

The largest GPT-2 variant is 13 times the size so it could take up more than 6.5 GBs of storage space.

GPT - 3

GPT-3 (2020, 175 billion parameters) has surprisingly demonstrated strong in-context learning capabilities, including zero-shot and few-shot learning abilities.

In few-shot or zero-shot settings without further fine-tuning, GPT-3 can achieve comparable performance with other fine-tuned state-of-the-art (SOTA) models.

Zero-Shot, One-Shot, and Few-Shot Learning

These describe how much help (examples or guidance) we give to a model before it has to perform a task

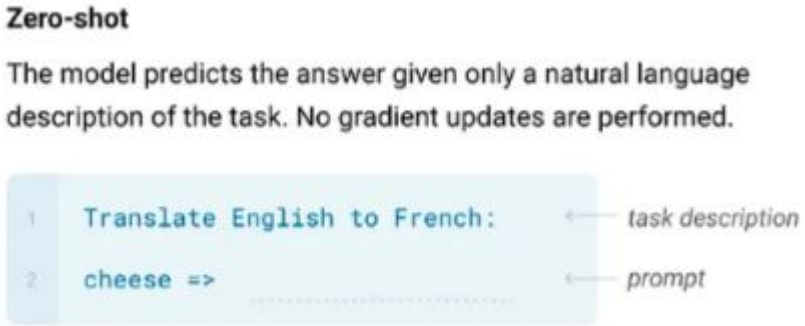
Zero-Shot

"Hey model, just do it—no examples."

The model performs a task without seeing any task-specific examples. It relies purely on its pretrained knowledge.

Is the following sentence +ve or -ve?

"I absolutely love this movie!"



One-Shot

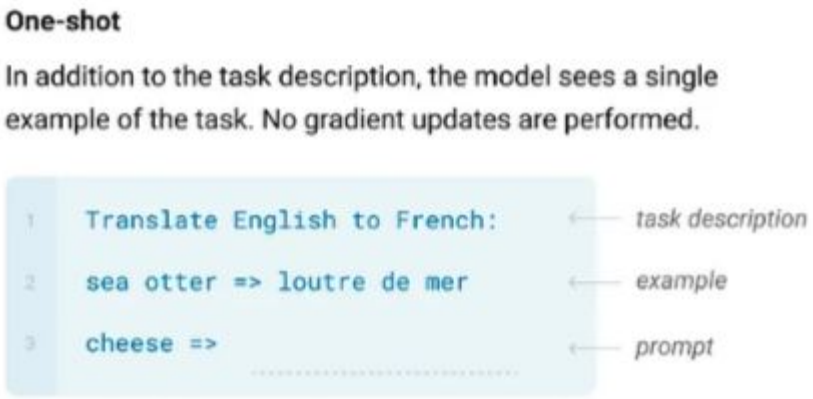
"Here's one example — now you try."

The model is shown one example of the task before doing it.

Classify the sentiment:

Example: "This is terrible." → Negative

Now classify: "This is fantastic."



Few-Shot

"Here's a few examples — learn the pattern."

The model is given a few examples (2 to ~10) before the real task.

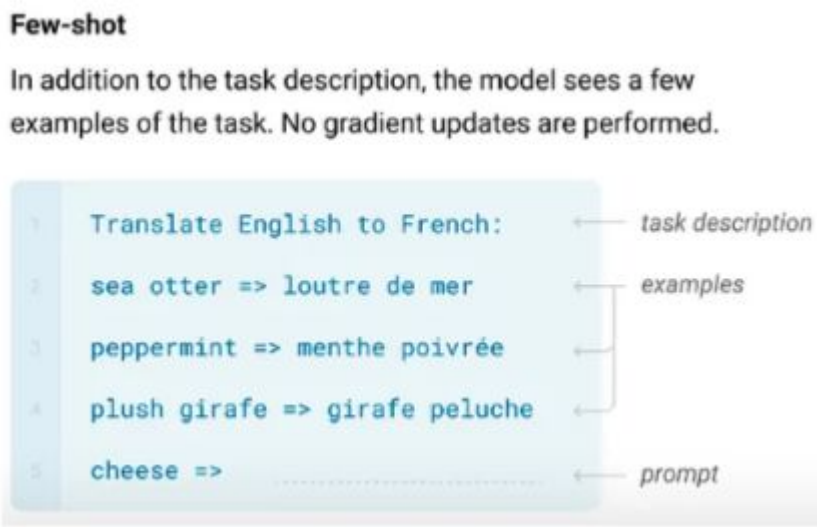
Classify the sentiment:

"Awful experience." → Negative

"I loved it!" → Positive

"Not great, not terrible." → Neutral

Now classify: "Best day ever!"



ChatGPT

It is a variant of the popular GPT-3 (Generative Pertained Transformer 3) model.

Chat GPT was modified and improved using both supervised and reinforcement learning methods, with the assistance of human trainer (RLHF).Chat GPT also has 176 billion parameters same as GPT -3 model. The learning includes 3 Steps.

- Supervised fine tuning of GPT 3.5 Model
- Reward Model
- Proximal Policy Optimization (PPO)

ChatGPT

ChatGPT is a pre-trained GPT-3 model that has been fine-tuned using the InstructGPT method.

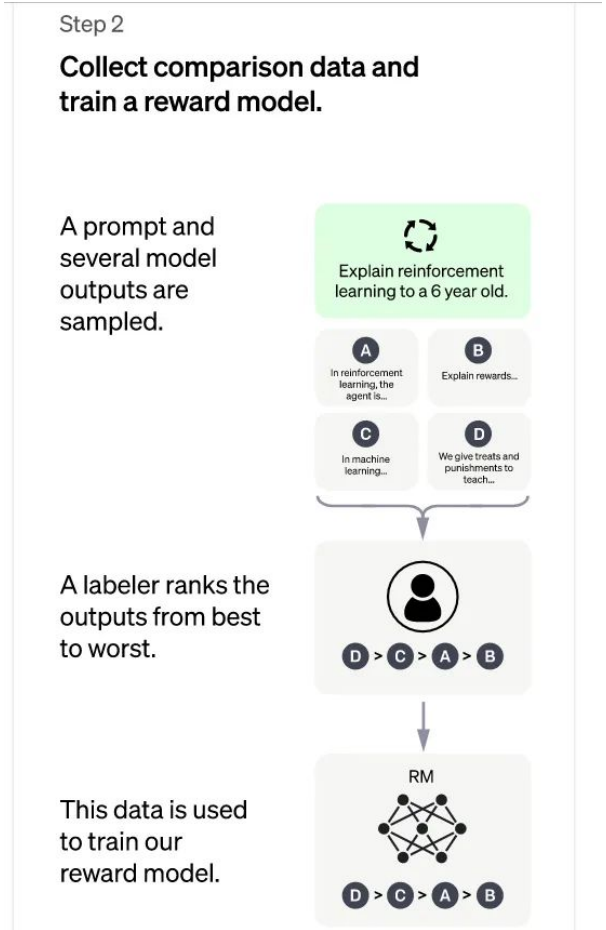
InstructGPT method was used for fine-tuning, which combines supervised learning of demonstration texts from labelers, then with reinforcement learning of generation text scoring and ranking, which are referred to as Reinforcement Learning from Human Feedback (RLHF).

Supervised fine tuning (Step1)

In first Step a pretrained GPT-3 model is used and it will be fine tuned with the help of labelers by creating a supervised dataset. Input Queries were collected from the actual user entries and model generated different responses with respect to that input prompts. The labelers then wrote an appropriate response to the input prompt's (how they want to see that prompt to be answered).The GPT-3 model was then fine-tuned using this new supervised dataset, to create GPT-3.5 model.

Reward Model (Step 2)

In this step SFT model is used and different input/prompts queries fed to the finetuned model and different responses were generated (4 to 7)for every input/prompt. Then labeler determines a reward for each of these outcomes and this reward is proportional to the quality of response with respect to initial prompt.



Proximal Policy Optimization (PPO) RL algo- Step3

In this step we pass unseen input sequences to the clone SFT model we got in step1. The model will generate response with respect to the input prompt. We pass the response to our reward model which we got in step 2 to understand, how high quality was this response for that input prompt and the output reward will be used to finetune the parameters of our SFT model .This is how our SFT model will incorporate more human like characteristics and behavior’s via Reinforcement Learning.