1. Text generation with an RNN

This tutorial demonstrates how to generate text using a character-based RNN. You will work with a dataset of Shakespeare's writing from Andrej Karpathy's [The Unreasonable Effectiveness of Recurrent Neural Networks](http://karpathy.github.io/2015/05/21/rnn-effectiveness/). Given a sequence of characters from this data ("Shakespear"), train a model to predict the next character in the sequence ("e"). Longer sequences of text can be generated by calling the model repeatedly.

**BERT**

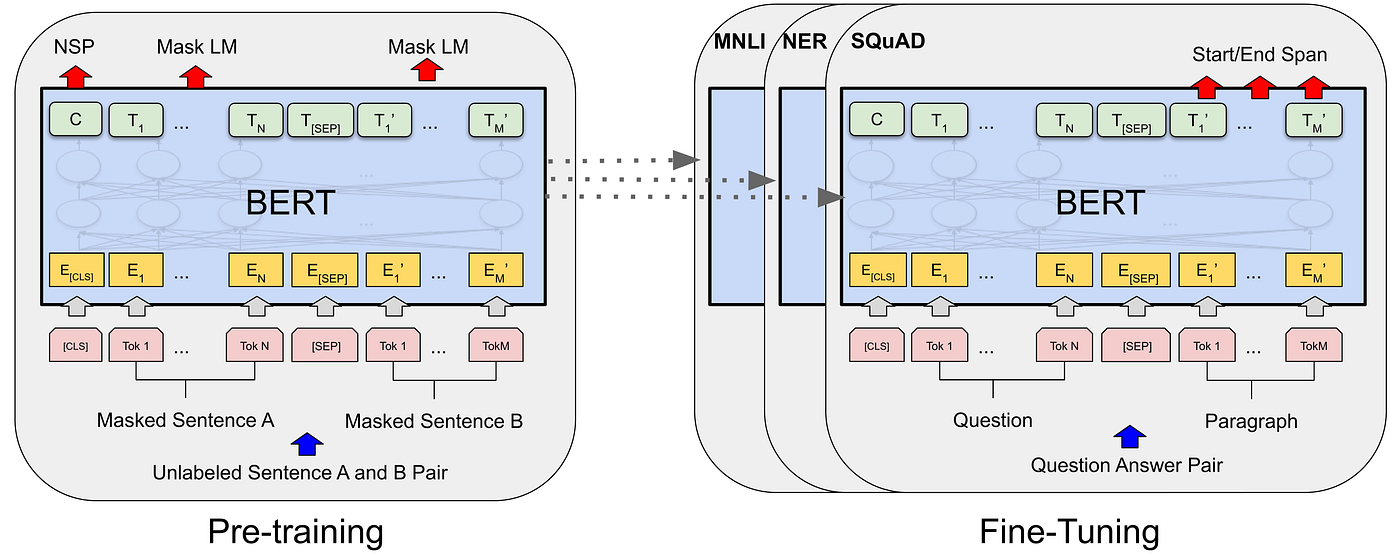
Developed by Google, BERT has revolutionized the way machines understand and process human language. In this article, we’ll delve into the key concepts behind BERT, its impact on NLP, and its implications for various applications.

## **Concept**

BERT, introduced in a research paper by Google AI researchers in 2018, stands out for its bidirectional approach to language understanding. Unlike its predecessors, BERT considers the entire context of a word by looking at both the left and right context in a sentence. This bidirectional understanding allows BERT to capture the nuances of language, including context-dependent meanings and relationships between words.

# Encoder Only Model

## BERT (Bi-directional Encoder Representation Transformers)



**BERT: Bi-directional Encoder Decoder Transformers**

## Abbreviation

* Bi-directional: Reads text in both left and right directions, better context.
* Encoder: Encoder Only Architecture.
* Representation: Creates meaningful word representation.
* Transformer: Based on Transformer Architecture.

## Overview

* Encoder Only model.
* Stacked Encoders.
* Bi-directional contexts.

## Pre-training and Fine-tuning

One of the strengths of BERT lies in its two-step process: pre-training and fine-tuning. During pre-training, BERT learns contextualized representations of words by predicting missing words in sentences. This phase allows BERT to develop a deep understanding of language patterns and relationships.

After pre-training, BERT is fine-tuned for specific NLP tasks, such as sentiment analysis, question answering, or named entity recognition. This adaptability makes BERT a versatile tool for a wide range of applications.

## Contextualized Embeddings:

BERT’s bidirectional approach results in contextualized word embeddings, which capture the meaning of a word based on its surrounding context. Traditional embeddings assign a fixed representation to each word, ignoring variations in meaning based on context. BERT’s contextual embeddings enable more accurate and nuanced language understanding.

## Tasks

1. **Masked Language Modelling (MLM)**

* Randomly masks words in a sentence.
* BERT’s task is to predict masked words using context.
* The outcome is to learn word relationships and context understanding.

2. **Next Sentence Prediction (NSP)**

* Focus on learning sentence relationships.
* BERT’s task is to determine if the second sentence follows the first naturally.

**Example:**

* Here Sentence A follows Sentence B
* Sentence A: I bought a mobile phone
* Sentence B: There was a sale of mobile phones

## Use Cases

1. Text Classification
2. Named Entity Recognition
3. Extractive Question Answering
4. Semantic Similarity
5. Search Engines
6. Chatbots and Virtual Assistants

## Code

from transformers import pipeline  
task = "sentiment-analysis"  
model\_name = "bert-base-uncased"  
input\_text = "The food in that hotel was not so good"  
sentiment\_analysis\_pipeline = pipeline(  
 task,  
 model = model\_name)  
sentiment\_analysis\_pipeline(input\_text)

[{'label': 'LABEL\_0', 'score': 0.7003146409988403}]

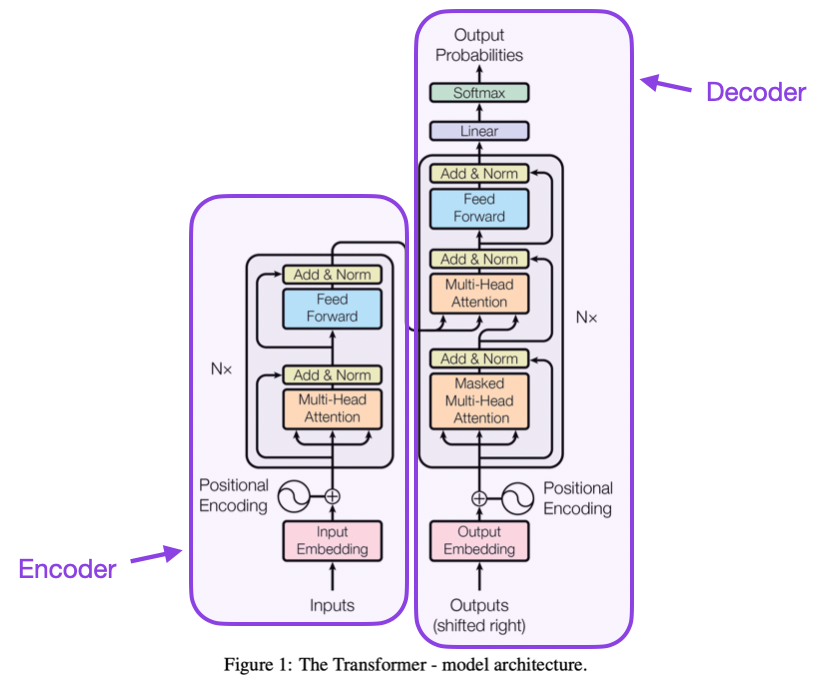
## References

**Sentiment Analysis:** <https://github.com/SharathHebbar/Transformers/blob/main/Encoder/sentiment-analysis.ipynb>

**Fill Mask:** <https://github.com/SharathHebbar/Transformers/blob/main/Encoder/fill-mask.ipynb>

**Extractive Question Answering:** <https://github.com/SharathHebbar/Transformers/blob/main/Basics/5_Extractive_QnA_using_BERT.ipynb>

# GPT: Overview

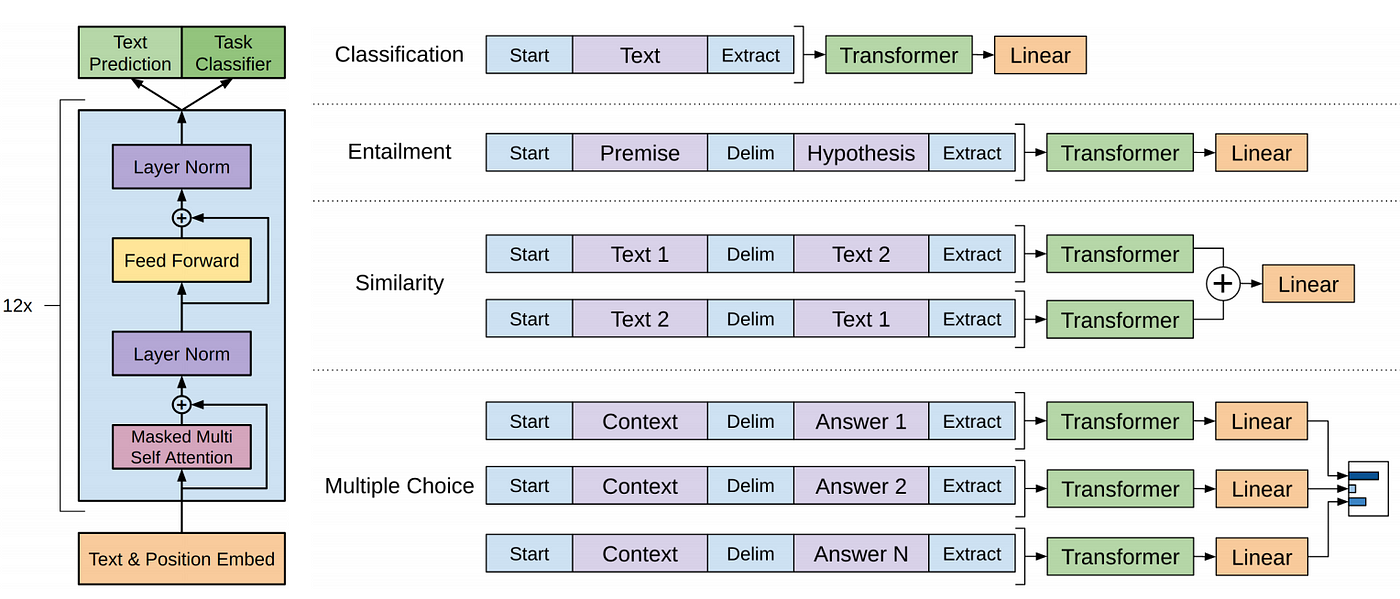


**Transformers: Attention is all you need**

Developed by OpenAI, GPT has emerged as a groundbreaking model, showcasing the vast potential of pre-trained transformer architectures. In this article, we’ll explore the fundamentals of GPT, its transformative impact on language processing, and its implications for diverse applications.

# Decoder Only Models

## GPT (Generative Pre-trained Transformers)



**GPT: Generative Pre-trained Transformers**

## Abbreviation

* Generative: It generates text.
* Pre-trained: As it is pre-trained.
* Transformer: Based on Transformer Architecture.

## Overview

* Coherent and contextually relevant text.
* Decoder Only.
* Stacked Decoder.
* Unidirectional Provisioning.

## Pre-training

What sets GPT apart is its pre-training paradigm. Before fine-tuning for specific tasks, GPT undergoes a pre-training phase on vast amounts of diverse text data. During this phase, the model learns to predict the next word in a sentence, gaining an intrinsic understanding of syntax, semantics, and contextual relationships within language.

## Versatility Through Fine-tuning

GPT’s real strength lies in its versatility. Once pre-trained, the model can be fine-tuned for a myriad of applications, including text completion, language translation, summarization, and even creative writing. This adaptability makes GPT a powerful tool for a wide range of industries and use cases.

## Contextual Understanding

GPT excels in contextual understanding, thanks to its ability to consider the entire context of a given input. This contextual awareness allows the model to generate more coherent and contextually relevant outputs, making it particularly effective for tasks that require a nuanced understanding of language.

## Tasks

1. Causal Language Modelling (CLM)

* Predicts the next word in a sentence.
* GPT’s task is to predict the next word in a sentence.

## Use Cases

1. Text Generation
2. Machine Translation
3. Summarization
4. Content Generation
5. Conversational AI

## Code

from transformers import pipeline  
task = "text-generation"  
model\_name = "gpt2"  
max\_output\_length = 30  
num\_of\_return\_sequences = 2  
input\_text = "Hello, I am"  
text\_generator = pipeline(  
 task,  
 model = model\_name)  
text\_generator(  
 input\_text,  
 max\_length=max\_output\_length,  
 num\_return\_sequences=num\_of\_return\_sequences)

[{'generated\_text': 'Hello, I am not the one who wrote this, but I know how to use it because if you take away my title right from what it says'},  
 {'generated\_text': "Hello, I am so sorry for what you've done. I had to make up things right, and the fact is, you and I are now"}]

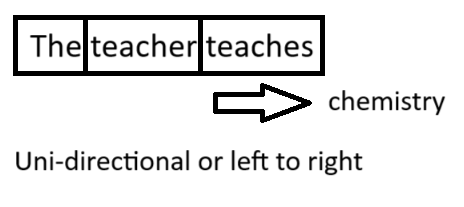
## References

**Text Generation:** <https://github.com/SharathHebbar/Transformers/blob/main/Decoder/text-generation.ipynb>

**Instruction Following Using GPT:** <https://github.com/SharathHebbar/Transformers/blob/main/Basics/6_Instruction_following_using_GPT.ipynb>

**Decoder Only Models**

* Decoder Only Models also known as Auto Regressive Models uses only a decoder for performing tasks such as Text Generation.
* Its task is to predict the next token taking into consideration the text to its left words in a sentence.
* In this technique, the next word in a sequence will be generated using information from their preceding words.
* It has a Uni-directional context in the sense it takes in the previous word in the sentence and predicts the next word in the sentence.

[](https://github.com/SharathHebbar/Transformers/blob/main/Decoder/assets/context.png)

* It is also called Causal Language Modelling.
* Examples of Causal Language Modelling/ Decoder Only models are GPT, GPT-2, GPT-3, BLOOM, PaLM

**Summarization, translation, Q&A, text generation and more at blazing speed using a T5 version implemented in ONNX.**

This package is still in alpha stage, therefore some functionalities such as beam searches are still in development.

## Installation

ONNX-T5 is available on PyPi.

pip install onnxt5

For the dev version you can run the following.

git clone https://github.com/abelriboulot/onnxt5

cd onnxt5

pip install -e .

## Usage

The simplest way to get started for generation is to use the default pre-trained version of T5 on ONNX included in the package.

**NOTE:** Please note that the first time you call get\_encoder\_decoder\_tokenizer, the models are being downloaded which might take a minute or two.

from onnxt5 import GenerativeT5

from onnxt5.api import get\_encoder\_decoder\_tokenizer

decoder\_sess, encoder\_sess, tokenizer = get\_encoder\_decoder\_tokenizer()

generative\_t5 = GenerativeT5(encoder\_sess, decoder\_sess, tokenizer, onnx=True)

prompt = 'translate English to French: I was a victim of a series of accidents.'

output\_text, output\_logits = generative\_t5(prompt, max\_length=100, temperature=0.)

# output\_text: "J'ai été victime d'une série d'accidents."

Other tasks just require to change the prefix in your prompt, for instance for summarization:

prompt = 'summarize: <PARAGRAPH>'

output\_text, output\_logits = generative\_t5(prompt, max\_length=100, temperature=0.)

If you want to get the embeddings of text, you can run the following

from onnxt5.api import get\_encoder\_decoder\_tokenizer, run\_embeddings\_text

decoder\_sess, encoder\_sess, tokenizer = get\_encoder\_decoder\_tokenizer()

prompt = 'Listen, Billy Pilgrim has come unstuck in time.'

encoder\_embeddings, decoder\_embeddings = run\_embeddings\_text(encoder\_sess, decoder\_sess, tokenizer, prompt)

ONNXT5 also lets you export and use your own models. See the examples\ folder for more detailed examples.

T5 works with tokens such as summarize:, translate English to German:, or question: ... context:. You can see a list of the pretrained tasks and token in the appendix D of the [original paper](https://arxiv.org/pdf/1910.10683.pdf).

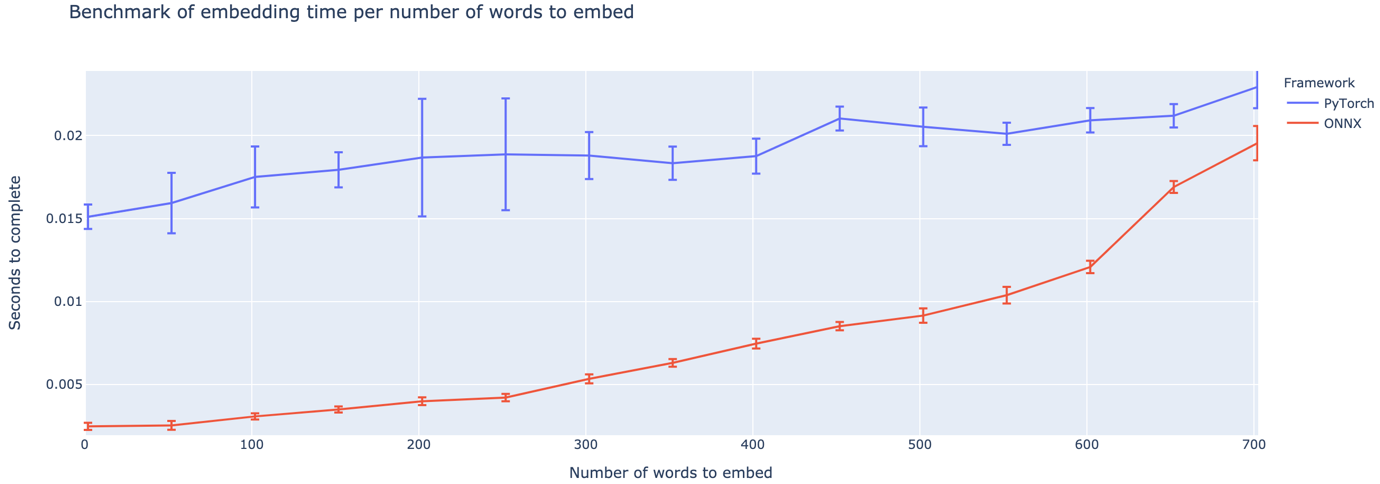
## Functionalities

* Run any of the T5 trained tasks in a line (translation, summarization, sentiment analysis, completion, generation)
* Export your own T5 models to ONNX easily
* Utility functions to generate what you need quickly
* Up to 4X speedup compared to PyTorch execution for smaller contexts

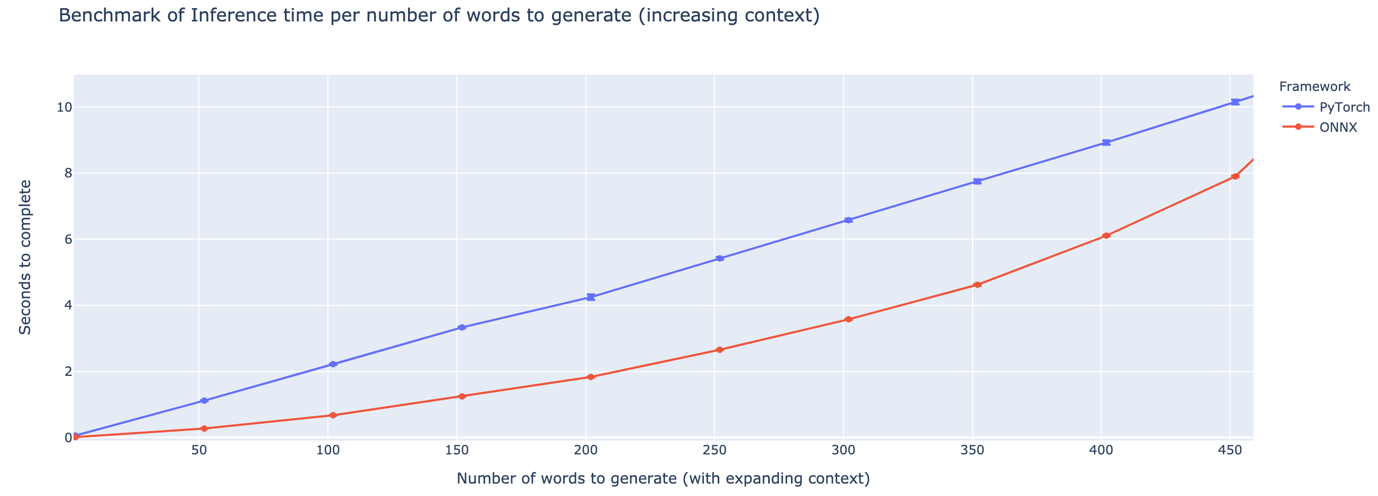
## Benchmarks

The outperformance varies heavily based on the length of the context. For contexts less than ~500 words, ONNX outperforms greatly, going up to a 4X speedup compared to PyTorch. However, the longer the context, the smaller the speedup of ONNX, with Pytorch being faster above 500 words.

#### GPU Benchmark, Embedding Task

[](https://github.com/abelriboulot/onnxt5/blob/master/data/Embedding_benchmark.png?raw=true)

#### GPU Benchmark, Generation Task

[](https://github.com/abelriboulot/onnxt5/blob/master/data/Generation_benchmark.png?raw=true)

## Contributing

The project is still in its infancy, so I would love your feedback, to know what problems you are trying to solve, hear issues you're encountering, and discuss features that would help you. Therefore feel free to shoot me an e-mail (see [my profile](https://github.com/abelriboulot) for the address!) or join our [slack community](https://join.slack.com/t/onnxt5/shared_invite/zt-gdjudd03-xutGvyQuYLMjBGnKH8fbLw).

## Acknowledgements

This repo is based on the work of Colin Raffel and Noam Shazeer and Adam Roberts and Katherine Lee and Sharan Narang and Michael Matena and Yanqi Zhou and Wei Li and Peter J. Liu from Google, as well as the implementation of T5 from the huggingface team, the work of the Microsoft ONNX and onnxruntime teams, in particular Tianlei Wu, and the work of Thomas Wolf on generation of text.

[Original T5 Paper](https://arxiv.org/pdf/1910.10683.pdf)

Explain

@article{2019t5,

author = {Colin Raffel and Noam Shazeer and Adam Roberts and Katherine Lee and Sharan Narang and Michael Matena and Yanqi Zhou and Wei Li and Peter J. Liu},

title = {Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer},

journal = {arXiv e-prints},

year = {2019},

archivePrefix = {arXiv},

eprint = {1910.10683},

}

**Using text-generation-inference and Inference Endpoints**

[Text Generation Inference](https://github.com/huggingface/text-generation-inference) is a production-ready inference container developed by Hugging Face to enable easy deployment of large language models. It has features such as continuous batching, token streaming, tensor parallelism for fast inference on multiple GPUs, and production-ready logging and tracing.

You can try out Text Generation Inference on your own infrastructure, or you can use Hugging Face's [Inference Endpoints](https://huggingface.co/inference-endpoints). To deploy a Llama 2 model, go to the [model page](https://huggingface.co/meta-llama/Llama-2-7b-hf) and click on the [Deploy -> Inference Endpoints](https://ui.endpoints.huggingface.co/new?repository=meta-llama/Llama-2-7b-hf) widget.

* For 7B models, we advise you to select "GPU [medium] - 1x Nvidia A10G".
* For 13B models, we advise you to select "GPU [xlarge] - 1x Nvidia A100".
* For 70B models, we advise you to select "GPU [2xlarge] - 2x Nvidia A100" with bitsandbytes quantization enabled or "GPU [4xlarge] - 4x Nvidia A100"

*Note: You might need to request a quota upgrade via email to*[*api-enterprise@huggingface.co*](mailto:api-enterprise@huggingface.co)*to access A100s*

You can learn more on how to [Deploy LLMs with Hugging Face Inference Endpoints in our blog](https://huggingface.co/blog/inference-endpoints-llm). The [blog](https://huggingface.co/blog/inference-endpoints-llm) includes information about supported hyperparameters and how to stream your response using Python and Javascript.