Models used with the Unsupervised Pre-Trained Model + Supervised Fine-Tuning paradigm

Pre-training + Fine-Tuning

We present a few models using this approach.

BERT

- [paper (https://arxiv.org/pdf/1810.04805.pdf)](https://arxiv.org/pdf/1810.04805.pdf)
- [model card (https://huggingface.co/bert-base-uncased)](https://huggingface.co/bert-base-uncased)

BERT (Bidirectional Encoder Representations from Transformers) is also a *fine-tuning* (universal model) approach.

Training objective

BERT is trained to solve **two** tasks

- Masked Language Modeling
- Next sentence prediction
    - does one sentence follow from another

(For a list of auxiliary tasks used, see [here (https://arxiv.org/pdf/2107.13586.pdf#page=44)](https://arxiv.org/pdf/2107.13586.pdf#page=44))

The **Masked Language Model** task is a generalization of "predict the next" token

- Mask (obscure) 15% of the input tokens, chosen at random
- The method for masking takes one of three forms
    - 80% of the time, hide it: replace with $[\mathrm{MASK}]$ token
    - 10% of the time: replace it with a random word
    - 10% of the time: don't obscure it

The training objective is to predict the masked word

The authors explain

- Since BERT does not know which words have been masked
- Or which of the masked words were random replacements
- It must maintain a context for **all** tokens

They also state that, since random replacement only occurs 1.5% of the time (10% * 15%), this does not seem to destroy language understanding

The second task is *entailment*

- Given two sentences, does the second logically follow from the first.

Perhaps this forces BERT to encode even more global context into its representations

Training

- BooksCorpus dataset (like GPT): 800MM words
- Wikipedia (English): 2,500MM words
- Training time
    - 4 days on 64 TPU chips

See Section A.2 ("Pre-training procedure", page 13) for details of training

- Optimizer: AdaM
- Learning rate decay
- Warmup

Architecture

BERT is an *Encoder*.

The original Transformer consists of an

- An Encoder which could attend to all tokens
    - does not use *masked attention* to force causal ordering
- A Decoder which used masking to enforce causal attention (not peeking into the future)

The Encoder allows bi-directional access to all elements of the inputs

- is appropriate for tasks that require a context-sensitive representation of each input element.

An Encoder is useful for tasks that require a summary of the sequence.

The summary can be conceptualized as a "sentence embedding"

- Sentiment

BERT in action

Interactive model for MLM (https://huggingface.co/bert-base-uncased?text=Washington+is+the+%5BMASK%5D+of+the+US)

GPT: Generalized Pre-Training

paper (https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf)

Summary article (https://openai.com/blog/language-unsupervised/)

GPT is a sequence of increasingly powerful (and big) models of similar architecture.

It is based on the paradigm of Unsupervised Pre-Training and Supervised Fine-Tuning.

Architecture

GPT models are stacks of Transformer Decoders.

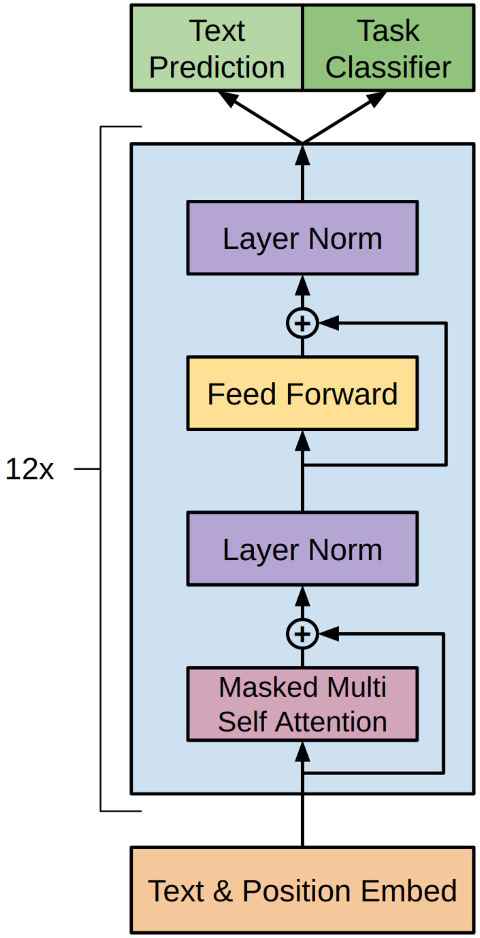
Recall the specifics of a Transformer Decoder

- Recurrent: output of time step t appended to input available at time step $(t+1)$
- Causal ordering of inputs
    - Left to Right, unidirectional
    - Implemented via Masked Self-attention

A Decoder is appropriate for *generative* tasks.

The Unsupervised Pre-Training task is generative.

- They are all trained on a Language Model objective: predict the next word



Picture from: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

Size

Each generation of the GPT family

- Increases the number of stacked Transformer blocks
- Increases the size of the training data

The first generation model (called "GPT") architecture

- $N = 12$ Transformer blocks (stacked)
- $d = 768$ (referred to as d_{model} in the paper)
    - Recall that d is the size of each position of the Encoder output
    - Is also the size of the output of all internal layers
- $n_{\text{heads}} = 12$
    - Recall that Multi-head Attention uses several Attention heads
    - On a reduced length transformation of the length d input
    - $d_{\text{head}} = \frac{d_{\text{model}}}{n_{\text{heads}}} = 64$
- Feed Forward Network
    - Output of Attention layer (size d_{model}) connected to
    - $4 * d_{\text{model}} = 3072$ internal nodes
- $\bar{T} \leq 512$
    - maximum sequence length.

GPT uses a total of 117 million weights.

It is trained on

- 5GB of text (BooksCorpus dataset consisting of 7,000 books: 800MM words)
- Training time
    - 30 days on 8 GPUs
    - 26 petaflop-days

Unsupervised Pre-Training

The Pre-Training task is to predict the next word in the sequence.

The Unsupervised Training objective is to

- maximize the likelihood for the "target" word (next word in sequence)
- maximize log likelihood on $\mathcal{U}$ (a corpus of tokens)

$$\mathcal{L}_1(\mathcal{U}) = \sum_i \log p(u)_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

The stacked Decoder blocks are described mathematically in the paper as

$$
\begin{aligned}
h_0 &= UW_e + W_p && \text{concatenate Input Embedding and Positi} \\
h_i &= \text{transformer\_block}(h_{i-1}) && \text{connect output of layer } (i-1) \text{ to input of} \\
& && \text{for } 1 \le i \le n \\
p(U) &= \text{softmax}(h_n W_e^T) && \text{Final output is probability distribution o} \\
& && h_n \text{ is output of top transformer block} \\
& && h_n W_e^T \text{reverses the embedding to obtain t}
\end{aligned}
$$

where

$$
\begin{array}{ll}
U & \text{context of size } k : [u_{-k}, \ldots, u_{-1}] \\
W_e & \text{token embedding matrix} \\
W_p & \text{position encoding matrix} \\
h_i & \text{Output of transformer block } i \\
n & \text{number of transformer blocks/layers}
\end{array}
$$

See [Section 4.1 ("Model specifications") of the paper (https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf#page=4)](https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf#page=4) for details of training

- Optimizer: AdaM
- Learning rate decay
- Warmup

We briefly introduced these concepts in earlier modules.

Hopefully it is somewhat interesting to see them used in practice.

Supervised Fine Tuning

The end-user uses the pre-trained model (architecture and weights)

- Trains on a small set $\mathcal{C}$ of domain-specific examples for a **Classifiation task** on a sequence of words

$$\begin{aligned} \mathcal{C} &= [\mathbf{x^{(i)}}, \mathbf{y^{(i)}} | 1 \le i \le ||\mathcal{C}||] \\ &= \mathbf{x}^{\mathbf{(i)}}_{(1)}, \ldots, \mathbf{x}^{\mathbf{(i)}}_{(m)}, \mathbf{y^{(i)}} \end{aligned}$$

- To fine-tune the weights

The process is described mathematical short-hand in the paper by defining the Fine Tuning Objective:

- maximize log likelihood on $\mathcal{C}$
  $\mathcal{L}_2(\mathcal{C}) = \sum_{(\mathbf{x},\mathbf{y})} \log p(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_m) \quad \text{where } \mathbf{y} = \text{softmax}(h_l^m W_y)$

Let's understand this

- Take output of layer l of the model: h_l^m
    - the m is referring to the length of the input
- Add a Classification head specific to the narrow domain
    - $\text{softmax}(h_l^m W_y)$ is the mathematical formula for Logistic Regression
- Using weights from unsupervised pre-training

The authors also experimented with a Fine Tuning Objective that included the Language Model Pbjective

$$\mathcal{L}_3(\mathcal{C}) = \mathcal{L}_2(\mathcal{C}) + \lambda \mathcal{L}_1(\mathcal{C})$$

Results of Unsupervised Pre-Training + Supervised Fine-Tuning

- Tested on 12 tasks
- Improved state-of-the-art results on 9 out of the 12

GPT 2

GPT-2

paper (https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf)

Model card (https://github.com/openai/gpt-2/blob/master/model_card.md)

Summary (https://openai.com/blog/better-language-models/)

Second Generation model.

Size

- $N = 48$ Transformer blocks (4 times first generation)
- $d = 1536$ (2 times first generation)
- $n_{\text{heads}} = 16$ (1.5 times first generation)
    - $d_{\text{head}} = \frac{d_{\text{model}}}{n_{\text{heads}}} = 96$
- $\bar{T} = 1024$ (2 times first generation)

GPT-2 uses 1.5 billion weights.

It is trained on

- 40GB of data (10 times the first generation)

Results on Zero-shot tasks

Tested on 8 tasks

- State of the art on 7 out of the 8

GPT-3

Third Generation model.

paper (https://arxiv.org/abs/2005.14165)

Model card (https://github.com/openai/gpt-3/blob/master/model-card.md)

Summary ()

Size

- $N = 96$ Transformer blocks (8 times first generation)
- $d = 12,288$ (16 times first generation)
- $n_{\text{heads}} = 96$ (8 times first generation)
    - $d_{\text{head}} = \frac{d_{\text{model}}}{n_{\text{heads}}} = 128$
- $\bar{T} = 2048$ (4 times first generation)

GPT-3 uses 175 billion weights.

It is trained on

- 570 GB of data (100 times first generation)
- Training cost
    - $ 42K
    - 190K KWh of electricity @ $ 0.22 per KW hour

The training set comes from several sources

- Common Crawl (https://commoncrawl.org/the-data/get-started/)
    - web crawler over multiple years
    - 570 GB (100 times GPT)
    - 410 billion tokens
- Additional training sets, for experiments
    - Webtext2 (https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf)
        - Web pages originating from highly ranked Reddit links
        - 19 billion tokens
    - Books
        - 67 billion tokens -Wikipedia
        - 3 billion tokens

GPT-4 (non-official)

OpenAI has not released details about the GPT-4 architecture.

However, an industry-intelligence firm has compiled a detailed report (https://www.semianalysis.com/p/gpt-4-architecture-infrastructure) using various sources.

This report is **not free**, but information from it has leaked (https://archive.is/2RQ8X) and we use this speculation as the basis for this section.

Size

- $N = 120$ Transformer blocks (10 times first generation)
- $\bar{T} = 32K$ (64 times first generation)

GPT-4 uses about 1.8 trillion parameters.

One **cannot** make an apples-to-apples comparison of number of parameters as GPT-4 uses a *Mixture of Experts (MOE)* model

- 16 experts
    - each of size 111 billion parameters
    - each specializes in certain tasks (e.g., areas of knowledge)

There are also 55 billion parameters for attention (the $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ matrices ?) that are shared across experts.

In an MoE model: **only some of the experts** are active in generating each output token

- GPT-4 "routes" the work to 2 of the experts

Such models are called *sparse*

- only a fraction of the parameters are involved in each output
- as opposed to a *dense* model that utilizes all the parameters

The advantage of this is *reduced cost of inference*

- with 2 experts

$$
\begin{aligned}
\text{number of parameters used} &= \\
&= 2 \quad && \text{number of experts} \\
& \quad * \quad 111 && \text{billions of parameters per} \\
& \quad + \quad 55 && \text{billion shared attention pa} \\
&= 280 && \text{billions of parameters}
\end{aligned}
$$

  - inference cost 560 TFlops
- with 16 experts (Dense model)

$$
\text{number of parameters used} \quad = \quad 16 * 111 + 55 \approx 1.8 \qquad \text{trillic}
$$

  - inference cost 3700 TFlops

GPT-4 is trained on what is called 13 Trillion tokens

- but that is *not* the size of the training set
- these are the total number of tokens trained over *multiple* epochs
    - 2 epochs for text
    - 4 epochs for code

The $\bar{T}$ (context length) also needs explanation

- starts at $8K$ for pre-training
- expanded to $32K$ during fine-tuning

Cost

GPT-4 was trained on 25,000 NVidia A100 GPUs.

- cost (per Amazon): $10 K/GPU
    - $250 MM hardware cost
        - but is run on Microsoft cloud
    - $ 1/hour
    - 90 to 100 days

      $$\$60\text{MM} = 24 \text{ hours/day} * 100 \text{ days} * 25000 \text{ GPU} * \$1\text{hour}$$

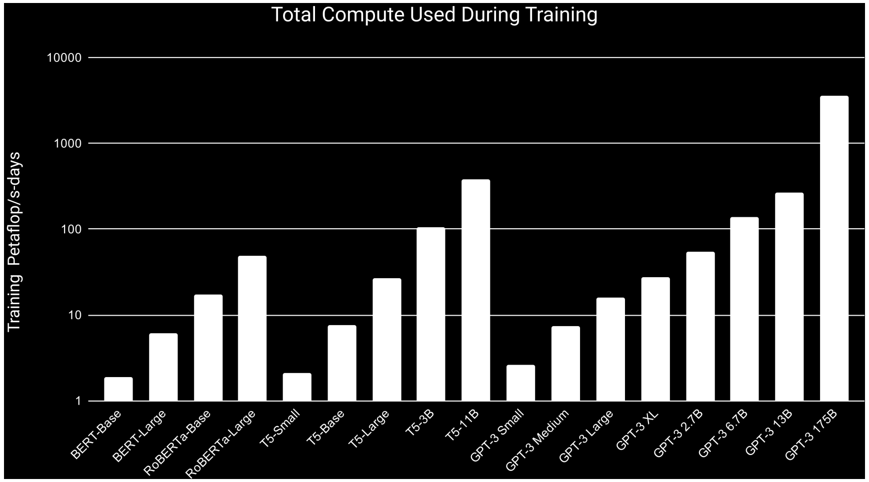
        - Model Flops Utilization (MFU) only $32\% - 36\%$
            - GPU *math operations* only active a fraction (max. 60%) of each second
                - lots of time waiting for data to be moved into memory (computation is *memory-bound*)
            - training: hardware faults, loss plateaus: re-start from checkpoints

Evolution of the GPT generations

You can see from the following graph how the computation times increase by orders of magnitude over the generations of GPT

- GPT-3 small $\approx$ GPT
- GPT-3 XL $\approx$ GPT-2

Compute time



Picture from: https://arxiv.org/pdf/2005.14165.pdf

Can you compete with GPT ? Why Transfer Learning matters

Intellectually: you know (approximately) how to replicate GPT-3.

Practically: can you do it ?

Scaling up the size of the training set: WebText

We argued early in the course that the "dirty secret" of Machine Learning was the effort expended in sourcing, cleaning, and pre-processing training data.

The GPT project illustrates this.

One key to the success of GPT-2 (and later generations) was a newly created training set that was scraped from the Web.

The most common web-scraped dataset is [Common Crawl (https://commoncrawl.org/)](https://commoncrawl.org/)

- large, diversified
- quality problems ?
    - Large set of pages pointed to are "gibberish"

The GPT team tried to create a high-quality crawl by using a curated approach to links

- Based on Reddit
- Only follow links originating from highly-ranked (high "karma") Reddit pages

The result is called WebText

- 40GB; 8MM documents
- removed any Wikipedia
    - since it is included in many of the benchmark tasks whose performance we want to measure out of sample

From a practical standpoint:

- this is a highly labor-intensive step
- that **precedes** training

Creating a large, quality dataset such as this is a significant impediment to your attempting to create our own model.

Cost of Training GPT-3 on your own

The computational requirements for training a Large Language Model is immense !

In the following table observe the "Total train compute" cost for models of varying size

- in flops (floating point operations)
- in Peta Flop (PF) days
    - number of days, assuming 10^{15} floating point operations per second available, running all day
    - can reduce number of days by more hardware (more floating point operations per second)

D Total Compute Used to Train Language Models

This appendix contains the calculations that were used to derive the approximate compute used to train the language models in Figure 2.2. As a simplifying assumption, we ignore the attention operation, as it typically uses less than 10% of the total compute for the models we are analyzing.

Calculations can be seen in Table D.1 and are explained within the table caption.

Model	Total train compute (PF-days)	Total train compute (flops)	Params (M)	Training tokens (billions)	Flops per param per token	Mult for bwd pass	Fwd-pass flops per active param per token	Frac of params active for each token
T5-Small	2.08E+00	1.80E+20	60	1,000	3	3	1	0.5
T5-Base	7.64E+00	6.60E+20	220	1,000	3	3	1	0.5
T5-Large	2.67E+01	2.31E+21	770	1,000	3	3	1	0.5
T5-3B	1.04E+02	9.00E+21	3,000	1,000	3	3	1	0.5
T5-11B	3.82E+02	3.30E+22	11,000	1,000	3	3	1	0.5
BERT-Base	1.89E+00	1.64E+20	109	250	6	3	2	1.0
BERT-Large	6.16E+00	5.33E+20	355	250	6	3	2	1.0
RoBERTa-Base	1.74E+01	1.50E+21	125	2,000	6	3	2	1.0
RoBERTa-Large	4.93E+01	4.26E+21	355	2,000	6	3	2	1.0
GPT-3 Small	2.60E+00	2.25E+20	125	300	6	3	2	1.0
GPT-3 Medium	7.42E+00	6.41E+20	356	300	6	3	2	1.0
GPT-3 Large	1.58E+01	1.37E+21	760	300	6	3	2	1.0
GPT-3 XL	2.75E+01	2.38E+21	1,320	300	6	3	2	1.0
GPT-3 2.7B	5.52E+01	4.77E+21	2,650	300	6	3	2	1.0
GPT-3 6.7B	1.39E+02	1.20E+22	6,660	300	6	3	2	1.0
GPT-3 13B	2.68E+02	2.31E+22	12,850	300	6	3	2	1.0
GPT-3 175B	3.64E+03	3.14E+23	174,600	300	6	3	2	1.0

- Amazon Cloud
    - G5 instance
        - NVidia A10G Tensor Core GPUs @ 250 Tflops/GPU
        - 8 GPU instance (2 Pflops) @$10/hour (with yearly contract; \$16\hour on-demand)
            - $240 per 2Pflops-day

- GPT-3 $\approx$ 3000 Pflop-days
    - $3000/2 = 1500$ days G5 instances to get 3000 Pflops-days
    - Cost = 1500 * $240/day = \$360K

Training: tricks of the trade

Training, in practice, involves more than a model and a training set

- Using multiple machines/GPU's: expect something to fail in the middle
    - necessity to checkpoint and be able to re-start
- Loss does not always decrease with increasing epoch
    - can speed up computation by using *half-precision* arithmetic (16 versus 32 bits). Half-size means
        - more examples per batch
        - fewer bytes transfered
        - but limits the size of the smallest number that can be represented
            - so the half-precision representation of a non-zero gradient can become zero
    - how to recover ?
        - Learning rate schedule "mid-flight corrections"

Some practical lessons are found [here (Training_a_LLM_practical.ipynb)].

Can you compete

Intellectually: yes.

Practically: requires much effort and expense

Fortunately, *someone else* often has performed the Unsupervised Pre-Training of a Large Language Model.

You may have little choice other than to leverage this effort and only perform the Supervised Fine-Tuning of the Pre-trained model on your specific task.

In [2]:

```
print("Done")
```

```
Done
```