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 $[\text{MASK}]$ 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)](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)](https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf)

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

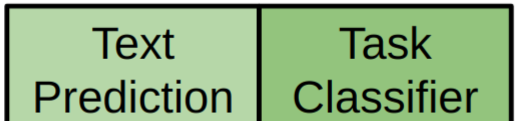
- Recurrent: output of time step t appended to input available at time step $(t+1)$
- Causal ordering of inputs
    - Left to Right, unidirectional
    - Implemented via Masked Self-attention

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

GPT: architecture


Text Prediction
Task Classifier

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

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

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

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

- 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)](https://arxiv.org/abs/2005.14165)

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

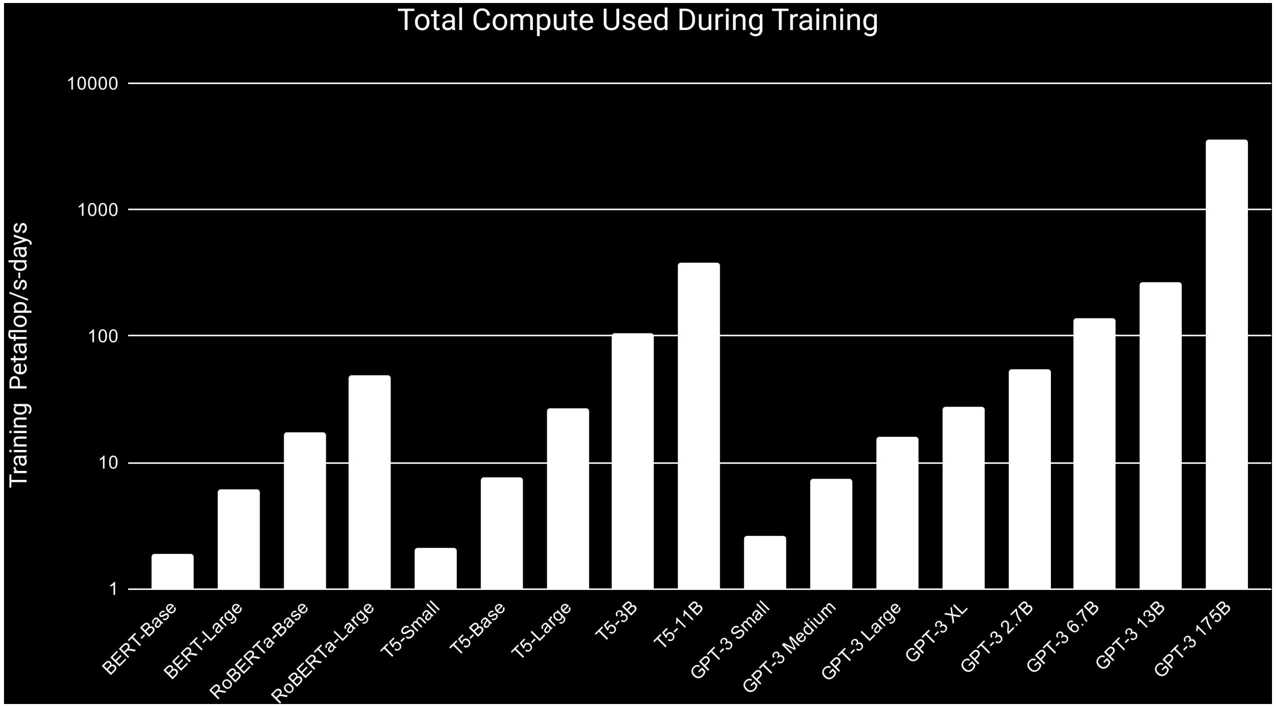
- [Common Crawl (https://commoncrawl.org/the-data/get-started/)](https://commoncrawl.org/the-data/get-started/)
    - web crawler over multiple years
    - 570 GB (100 times GPT)
    - 410 billion tokens
- Additional training sets, for experiments
    - [Webtext2 (https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf)](https://d4mucfpksywv.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

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.

- 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 [1]:

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

```
Done
```