Some Large Language Models used with 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

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

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

It is a *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

- Text generation
- Predict the next word in a sentence

Each generation of the GPT family

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

All models use

- Byte Pair Encoding
- Initial encode words with word embeddings

They are all trained on a Language Model objective.

GPT: architecture

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

The models can be described as

$$
\begin{aligned}
h_0 &= UW_e + W_p \\
h_i &= \text{transformer\_block}(h_{i-1}) \quad \text{for } 1 \le i \le n \\
p(U) &= \text{softmax}(h_n W_e^T)
\end{aligned}
$$

where

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

Let's understand this

- h_0, the output of the input layer
  - Uses word embeddings W_e on the input U
  - Adds *positional* encoding W_p to the tokens
- There are layers h_i of Transformer blocks $1 \leq i \leq n$
- The output $p(U)$
  - Takes the final layer output h_n
  - Reverses the embedding W_e^T to get back to original tokens
  - Uses a $\mathrm{softmax}$ to get a probability distribution over the tokens U
    - Distribution over the predicted next token

The training objective is to maximize log likelihood on $\mathcal{U}$ (a corpus of tokens)

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

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

- 12 Transformer blocks (37 layers)
    - $n_{\text{heads}} = 12, d_{\text{head}} = 64$
        - $d_{\text{model}} = n_{\text{heads}} * d_{\text{head}} = 768$
        - d_{model} is size
            - representation (bottle-neck layer)
                - fed into Dense Feed Forward layer
- 117 million weights
- Trained on
    - 5GB of text (BooksCorpus dataset consisting of 7,000 books: 800MM words)
    - Sequence of 512 tokens
    - Training time
        - 30 days on 8 GPUs
        - 26 petaflop-days

See Section 4.1 ("Model specifications") 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.

Unsupervised Training is used to create the Language Model.

This is followed by Fine Tuning on a smaller task-specific training set $\mathcal{C}$

This can be described as:

- Add linear output layer W_y to the model used for Language Modeling:
- h_l^m is output of transformer block l on input of length m
- Using Θ from unsupervised pre-training
- Fine Tuning Objective:
    - maximize log likelihood on $\mathcal{C}$
      $\mathcal{L}_2(\mathcal{C}) = \sum_{(\mathbf{x},\mathbf{y})} p(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_m) = \mathrm{softmax}(h_l^m W_y)$

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

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

Results of Supervised Pre-Training + Fine-Tuning

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

In [1]:

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

```
Done
```