Solal Danan
solal.danan@polytechnique.edu

Lab session # 2
ALTEGRAD 2024

10/22/24

1 Question 1

The **square mask** maintains the autoregressive nature of the model, which prevents the model from "looking ahead" at future tokens during training. This ensures that predictions made at any given position are based only on the current and past tokens. In the implementation, the mask is created using `torch.triu`, generating an upper triangular matrix where future positions are masked with $-\infty$, and current or past positions remain unmasked (set to 0). The mask is applied in the attention mechanism, specifically in the computation of Z, which represents the output of the attention layer. The self-attention mechanism calculates Z as follows:

$$Z = \text{softmax}(E + \text{mask}).V \quad \text{with} \quad \text{mask}(i,j) = \begin{cases} 0 & \text{if } i \geq j \\ -\infty & \text{if } i < j \end{cases}$$

where:

- E is the attention score matrix, which represents the relevance of each token to others in the sequence.
- mask ensures that future tokens are blocked by setting their attention scores to $-\infty$.
- V is the value matrix, containing the information the model will use for the current token.

Without the mask, the model would attend to all positions, including future tokens, which would break the autoregressive property. It ensures no lookahead in autoregressive tasks like language modeling, maintaining the sequential prediction structure.

Positional encoding is essential because transformers do not have an inherent sense to capture the order of tokens in a sequence. The model uses sinusoidal encoding (introduced in the "Attention is All You Need" paper [2]) to add position-related information to each token. The encoding is calculated as:

$$\text{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{\text{nhid}}}}\right), \quad \text{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{\text{nhid}}}}\right)$$

where pos is the token position and i is the dimension index. This encoding is added to the token embeddings, modifying the input such that the new input to the transformer becomes the sum of the original embeddings and the positional encodings. This allows the model to capture relative and absolute positions of tokens in the sequence, ensuring it understands the order in which tokens appear.

2 Question 2

In the context of transfer learning, we begin by pretraining a model on a language modeling task and then fine-tune it for a more specific task like classification. The reason we need to replace the classification head lies in the different goals and output requirements of these two tasks. The only difference needed in the architecture is the last linear layer. For both tasks this layer inputs nhidden (the output shape of the transformer) and outputs nclasses which is the parameter that differs.

- For language modeling, the model's objective is to predict the next word in a sequence based on the previous words. This is a self-supervised task, meaning the model does not require labeled data because the target is part of the input itself. The classification head in language modeling is designed to output a probability distribution over the entire vocabulary at each step, predicting which word will come next. The output dimension must cover all possible words in the vocabulary ($n_{classes} = v$ the vocabulary size in the case of token prediction). Pretraining on language modeling creates a strong base for understanding language, which can later be transferred to other tasks.
- In contrast, the classification task requires assigning a single label to the entire input sequence (for instance categorizing a review as positive or negative). This is a supervised task, meaning it requires that each input is associated with a known label. In a classification task, we don't need to predict words from the vocabulary at every position. Instead, the goal is to predict from a fixed number of classes. The classification head, therefore, needs to output a probability distribution over these classes rather than the full vocabulary. Here $n_{classes} = 2$, although we could have chosen 1 and used a sigmoid activation function.

3 Question 3

We will calculate the number of trainable parameters in our model for both the language modeling and classification tasks.

Let the following variables be defined:

- v: the vocabulary size (50,001 in this case).
- n_h: the hidden size, which is also used as the embedding size (200 in this lab).
- n_l: the number of transformer layers (4 in this lab).
- n_c: the number of output classes for the final linear layer (either v for language modeling or 2 for classification).

Next, we can decompose the forward pass through the model as follows:

1. The input tokens are first passed through an embedding layer, which converts them into vectors of size n_h. The embedding layer thus has $v \times n_h$ parameters.
2. The resulting embeddings are then passed through n_l transformer layers (decoder blocks). In each layer, we account for the following parameters:
   - Three weight matrices for the multi-head attention mechanism (queries, keys, and values), each of size $(n_h + 1) \times n_h$.
   - A feedforward network consisting of two linear layers, requiring $2 \times (n_h + 1) \times n_h$ parameters.
   - Normalization layers contributing $2 \times 2 \times n_h$ parameters.
   - An additional linear layer of size $(n_h + 1) \times n_h$.

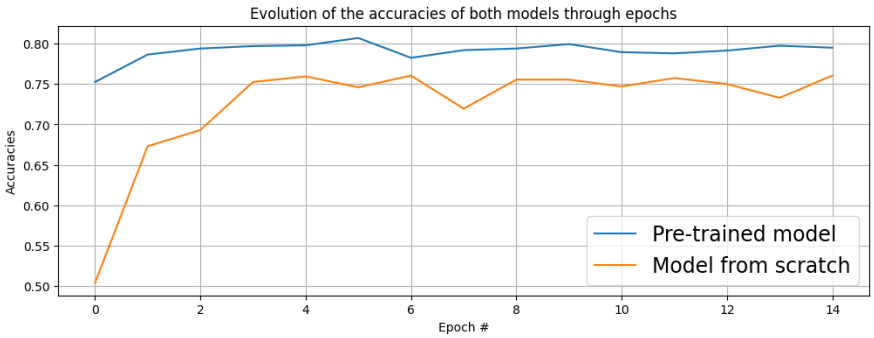
   Therefore, each layer has a total of $6n_h^2 + 10n_h$ parameters. Across n_l layers, the total number of parameters in the transformer part is $n_l(6n_h^2 + 10n_h)$.
3. For the classification task, an additional linear layer is added at the output, with $(n_h + 1) \times n_c$ parameters.

The total number of trainable parameters in the model is therefore:

$$v \times n_h + n_l(6n_h^2 + 10n_h) + (n_h + 1) \times n_c$$

For the language modeling task, where $n_c = v$, the total number of trainable parameters is approximately 21,018,401. For the classification task, where $n_c = 2$, the total is about 10,967,602.

4 Question 4



The results presented in the plot 4 highlights the benefits of transfer learning. First, we observe that the pre-trained model starts with a much higher accuracy from the very first epoch compared to the model trained from scratch. This indicates that the pre-trained model has already learned useful features during pre-training, which accelerates its convergence and performance on the classification task (sentiment analysis).
Additionally, the pre-trained model quickly stabilizes after only a few epochs (around epoch 4), reaching an accuracy that does not improve significantly with further training. This shows that fewer epochs are needed to reach a stopping condition for the pre-trained model, making it much more efficient in terms of training time.

On the other hand, even if the model trained from scratch improves a lot in the first few epochs, it never reaches the same level of accuracy as the pre-trained model, even after many epochs. This highlights the difficulty of training models from scratch, where more data and epochs are typically required to reach an acceptable performance.

5 Question 5

The limitation of the language modeling objective compared to the masked language model (MLM) objective, as presented in the research paper BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [1], lies in the way they capture context during training.

Classical strategies for applying pre-trained language representations, such as the one used in the notebook, rely on unidirectional language models. In the language modeling objective, the model predicts the next word in a sequence based only on the preceding words, which makes it unidirectional. This means the model can only consider context from one direction (left to right in this lab). This approach limits the model's ability to capture bidirectional context, which is crucial for tasks like question answering, where understanding relationships across the entire sequence is essential.

In contrast, the masked language model (MLM) objective used in BERT (Bidirectional Encoder Representations from Transformers) overcomes this limitation by allowing the model to leverage both left and right contexts. In MLM, random tokens in the input sequence are masked, and the model must predict the original tokens based on the surrounding context. This bidirectional training enables the model to generate more contextualized representations, incorporating information from both previous and subsequent words. This approach allows for the pre-training of a deep bidirectional Transformer.

Therefore, a key limitation of the language modeling objective used in the notebook is its unidirectional nature, which restricts the model's ability to fully capture dependencies across the sequence. In contrast, the MLM objective provides a more comprehensive understanding of the text, making it more effective for tasks requiring deep contextual comprehension.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.