GRAVIER Thomas
thomas.gravier@gadz.org

Lab session # 2

ALTEGRAD 2024

10/22/24

1 Question 1

The role of the square mask is to ensure that, for each part of the sentence, a token can only attend to the previous tokens and not the ones that come after it.

Without positional encoding, the attention heads have no sense of the token's position in a sentence. With positional encoding, we introduce a way to encode the position of each token in the sentence.

2 Question 2

We need to replace the **classification head** because the requirements for classification differ from language modeling.

- **Why replace the classification head?** The classification head is task-specific. In language modeling, the head predicts the *next token*, requiring an output size equal to the vocabulary. For classification, however, the output must match the number of target *classes* (e.g., "positive" or "negative" in sentiment analysis). Therefore, a different head is needed for classification tasks.
- **Main difference between language modeling and classification:**
  - *Language modeling* predicts the next word/token in a sequence, so the output size matches the vocabulary.
  - *Classification* predicts a class for an entire sequence (or each token), with the output matching the number of classes, usually much smaller than the vocabulary.

In summary, we replace the head to match the output to the classification task's specific number of classes, which differs from the language modeling objective.

Question 3

Let's calculate the number of trainable parameters for each layer of the model.

- *Language modeling task*:

  Embedding block: The number of trainable parameters is given by $n_{token} \times n_{hid}$. In our example, $n_{token} = 50,001$ and $n_{hid} = 200$, which gives a total of **10,000,200 trainable parameters**.

  Positional encoding: This part does not contain any trainable parameters.

  Transformer layers: To calculate the number of parameters in a single Transformer layer, let's break it down step by step.

  For one `Transformer layer`:

  - **Self-attention**:
    * The W_Q, W_K, W_V matrices each have dimensions (n_{hid}, n_{hid}), which results in $200 \times 200 \times 3 = 120,000$ trainable parameters.
    * The associated biases add up to $200 \times 3 = 600$ parameters.
    * The projection back to the n_{hid} dimension adds another $200 \times 200 = 40,000$ parameters, with a bias of 200.

    Therefore, the `Self-attention` mechanism contains $120,000 + 600 + 40,000 + 200 =$ **160,000** trainable parameters.

- **Feedforward**:
  - First normalization: 200 weights and 200 biases, for a total of 400 parameters.
  - Second normalization: similarly, 400 parameters.
  - First linear layer: $n_{hid} \times n_{hid} + n_{hid} = 200 \times 200 + 200 = 40,200$ parameters.
  - Second linear layer: again, $40,200$ parameters.

  In total, the `Feedforward` part consists of $400 \times 2 + 40,200 \times 2 =$ **81,200** trainable parameters.

For a single `Transformer layer`, this results in a total of $160,000 + 81,200 =$ **261,200** trainable parameters.

Since our model contains 4 Transformer layers, the total number of parameters for these layers is $4 \times 261,200 =$ **968,000** trainable parameters.

Classification Head: This final linear layer has dimensions $W \in \mathbb{R}^{n_{classes} \times n_{hid}}$ and $b \in \mathbb{R}^{n_{classes}}$. The total number of parameters is $50,001 \times 200 + 50,001 =$ **10,050,020** trainable parameters.

Therefore, for the *language modeling task*, the model has

$$10,000,200 + 968,000 + 10,050,020 = \mathbf{21,018,401}$$

trainable parameters.

- *Classification task*:

  The main difference compared to the previous task is in the `Classification Head`, where $n_{classes} = 2$ instead of $50,001$. The total number of parameters is then $2 \times 200 + 2 = 402$ trainable parameters.

  Therefore, for the *classification task*, the model has

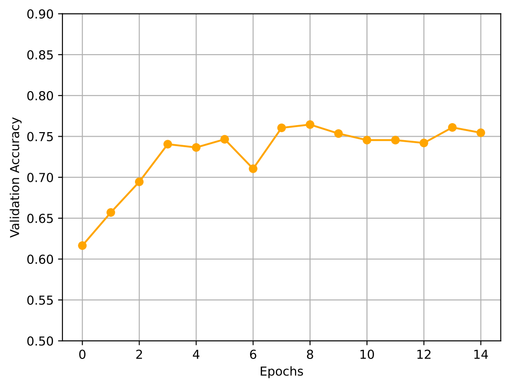
$$10,000,200 + 968,000 + 402 = \mathbf{10,969,602}$$

  trainable parameters.

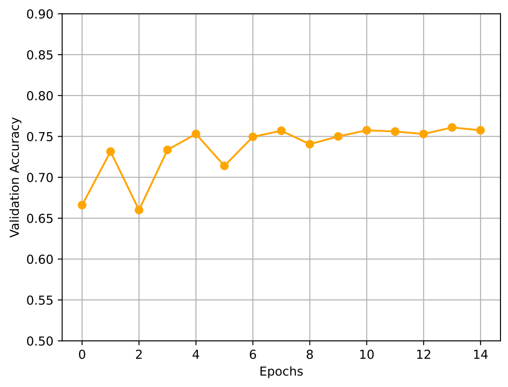
The main difference in the number of trainable parameters between the two tasks is in the final layer. In the language modeling task, the output is projected to the vocabulary size, whereas in the classification task, the output is projected to the number of classes.

Question 4

Results of Task 7:



(a) Validation accuracy progression of the model trained from scratch over 15 epochs.



(b) Validation accuracy progression of the pretrained model over 15 epochs.

Figure 1: Comparison of validation accuracies for the two models tested.

Interpretation of the results:

The pretrained model consistently performs better, achieving a higher peak accuracy (0.81 vs 0.76) and a better overall average. This suggests that the pretrained weights provide a more effective starting point, enabling faster learning.
Given the small learning rate, these results are expected, with only marginal improvements in validation accuracy after 5-6 epochs.
Furthermore, the scratch model's stabilization around 0.75 indicates difficulties in achieving better classification performance. This may be due to limited training data (around 1,600 normal sentences) or suboptimal hyperparameter tuning, which was initially configured for fine-tuning and transfer learning. Meanwhile, the pretrained model performs better at distinguishing between the two classes but still shows variability in accuracy, which suggests that it may also benefit from additional training data.
For both models, the slow improvement over epochs might indicate overfitting to the training data.
In summary, the pretrained model shows promise with its higher peak accuracy, highlighting the advantages of transfer learning for this task. Further improvements could be achieved through data augmentation, hyperparameter tuning, and additional fine-tuning for both models.

Question 5

Transformers used for language model pre-training are usually implemented with as a left-to-right architecture, which means that every tokens can only have informations from previous ones. In this notebook, we used the `TransformerEncoderLayer` from Pytorch, based on a traditional left-to-right approach. For language modeling tasks, it's well efficient. One problem is that is less adaptable when using those models for other specfic tasks. For instance, for what we do in this notebook, sentiment analysis, it could be more efficient to have access to all tokens of the sentences, as we want a general understanding of the text review and not just predict the futur word.

One solution proposed in [1] is to use a bidirectional approach, such as Bidirectional RNNs/LSTMs (we can refer to the previous Lab session). Introduced in the BERT paper ([1]), they present masked language models, by masking tokens and predicting them based on the entire context (left and right) during the unsupervised task. This could lead to a better understanding of the global review. This kind of limitation task can be known as representation learning.
With a unidirectional Language modelling, representation learning may be less robust for tasks that require understanding of the context (from both direction). In comparaison, Masked Language models can learn easily more contextually representations, as it must understand the entire sequence to predict the masked tokens.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.