ADVANCED LEARNING FOR TEXT AND GRAPH DATA

Lab session 3: Transfer Learning in NLP

Lecture: Prof. Michalis Vazirgiannis
Lab: Moussa Kamal Eddine and Hadi Abdine

November 23, 2021

This handout includes theoretical introductions, coding tasks and questions. Before the deadline, you should submit here a **.zip** file (max 10MB in size) named `Lab<x>_lastname_firstname.pdf` containing a `/code/` folder (itself containing your scripts with the gaps filled) and an answer sheet, following the template available here, and containing your answers to the questions. Your answers should be well constructed and well justified. They should not repeat the question or generalities in the handout. When relevant, you are welcome to include figures, equations and tables derived from your own computations, theoretical proofs or qualitative explanations. **One submission is required for each student. The deadline for this lab is November 30, 2021 11:59 PM**. No extension will be granted. Late policy is as follows: $]0, 24]$ hours late $\rightarrow$ -5 pts; $]24, 48]$ hours late $\rightarrow$ -10 pts; > 48 hours late $\rightarrow$ not graded (zero).

---

1 Introduction

Transfer learning that is, solving tasks with models that have been pretrained on very large amounts of data, was a game changer in many deep learning tasks. In NLP, while annotated data are scarce, raw text is virtually unlimited and readily available. Thus, the ability to learn good representations from plain text could greatly improve general natural language understanding. Learning without labels is enabled via self-supervised learning, a setting in which a system learns to predict part of its input from other parts of its input.

One way to realise this pre-training is to use *generative pre-training* [4] of a language model. In this pre-training phase, the model will learn to predict the next tokens in a sequence given the previous ones. Thus, this phase does not require any type of annotations apart from the input text itself. Once the language model is sufficiently pre-trained, it can be fine-tuned on supervised tasks while requiring minimal changes to its architecture (replacing the classification head).

In this lab we will:

- Implement and pretrain a language model with transformer architecture.
- Use the pretrained model (transfer learning) to perform a sentiment analysis task which consists of classifying some books reviews into positive and negative ones.
- Compare the performance of the pretrained model to a model trained from scratch.

2 The Model

Our model is based on Transformers [5]. While the Transformer is a model that follows the encoder-decoder structure, it is possible to use only the encoder part (as in BERT) or the decoder part (as in gpt) to perform some specific tasks. In this notebook, we use a multi-layer Transformer decoder for the language model. Fortunatly, PyTorch recent releases include a standard transformer blocks that can be easily used and adapted.

Let's start by implementing the model. The different layers used in this model are the following:

- The embedding layer
- Positional Embedding
- The transformer layers
- Linear layer for decoding and classification

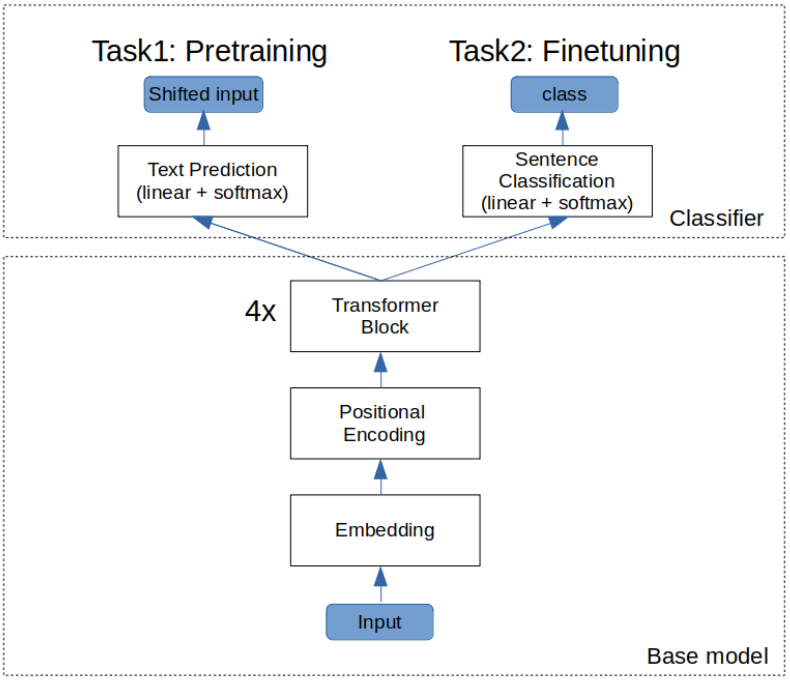

Figure 1: Visualization of the model

The same model can be used for language modeling and classification by just replacing the last layer in the model. This is why we split our model into two modules: - base module: which consists of the 3 first layers. - classifier module: which consists of the last layer.

After the pretraining, the parameters of the base model can be "transferred" to the model used in the classfication task, while the classifier module should be replaced by a new head initialized randomly.

Some of the code is adapted from this nice tutorial: `https://pytorch.org/tutorials/beginner/transformer_tutorial.html`

Task 1
Fill in the gaps in the `TransformerModel()` class to implement the described model.

Question 1 (4 points)
What is the role of the square mask in our implementation? What about the positional encoding?

Question 2 (2 points)
Why do we have to replace the classification head? What is the main difference between the *language modeling* and the *classification* tasks?

Question 3 (6 points)
How many trainable parameters does the model have in the case of

- *language modeling* task.
- *classification* task.

Please detail your answer.

3 Vocabulary and Tokenization

To train the language model, the text in our corpus should be first tokenized. We use sentencepiece `https://github.com/google/sentencepiece` that implements byte-pair-encoding (BPE), a subword tokenization algorithm. The vocabulay is given in `dict.txt` file. Let's load it, and map each token to a unique index.

In our experiments we use datasets that are already tokenized.

Task 2
Fill in the gaps to create a `token2ind` and `ind2token` mapping dictionaries

4 Data Loader

We use the `DataLoader` class, to load our dataset and generate the mini-batches used in the training. `get_loader()` returns a `DataLoader` object, which is an iterable over data samples. The data loader can return mini-batches for language modeling or sequence classification based on the `task` argument that we pass to `get_loader()` function. Currently the function supports *language_modeling* and *classification* tasks.

For the *language_modeling* task, both the input and the target are batches of sequences. In fact, the target is basically a shifted version of the input, in such a way that each token is predicted given all previous tokens. For example, for a sequence A B C D:
Input: <sos> A B C D
Output: A B C D <eos>

For the *classification* task, the input is a batch of sequences and the target is a batch of scalar labels. For more information about data loaders check: `https://pytorch.org/docs/stable/data.html`

Task 3
Fill in the gap inside `Dataset` class in order to create the input sequence

5 The Training

In this section we will implement a `train()` function that trains our model for one epoch. As we said, in this lab we will use a language modeling objective in the pretraining phase. Given the previous

tokens in a sequence, the model will try to predict the next one. The same function can be used for both pretraining and fine-tuning phase.

The training procedure is as follows: 1. Iterate over the data-loader. 2. In each iteration perform one forward pass. 3. Compute the loss through back-propagation. 4. update the parameters of your model using sgd. 5. repeat for n epochs

N.B: While in the *language_modeling* task all the vectors at the output of the base model are used, in the *classification* task we only use the vector representing the last token to perform the prediction.

Task 4
Fill in the gaps in `train()` function.

6 Text Generation

Being trained on a language modeling objective, our model can be used in the inference mode to generate/complete sentences. However, the pretraining phase takes a lot of time compared to the fine-tuning. For example, CamemBERT$_{base}$ [3] was pretrained for 24 hours on 256 Nvidia V100 GPUs and BARThez [2] was pretrained for 60h on 128 Nvidia V100 GPUs!

Of course, we don't have enough time and resources to efficiently pretrain our model, so instead we will load the weights from a checkpoint that has been pre-trained for 12 hours on 1 GPU.

Take a look here: `https://pytorch.org/tutorials/beginner/saving_loading_models.html#saving-loading-a-general-checkpoint-for-inference-and-or-resuming-training`

Task 5
Implement the function `infer_next_tokens()` that takes as input a string `sent` and an integer `max_len` and returns a completion of the input sentence. The generation should stop when the model generates `<eos>` or the length of the generated sentence reaches `max_len`

7 Supervised Task

It's time to train the model on the supervised task, which is in our case sentiment analysis. It consists of predicting whether a book review is a positive or a negative review. The model will be trained in 2 settings.

- Training from scratch: All model parameters are randomly initialized.
- Transfer learning: Only the classification head is trained from scratch, all other parameters are copied from the pre-trained model.

The training function is already implemented. However, to evaluate the model at each epoch we have to implement a function that computes the accuracy of the model on the validation set.

Task 6
Implement the `accuracy()` function. This function takes as input a `data_loader` and returns an float.

Task 7
Visualize the evolution of the accuracy of the model in function of the epoch in both settings.

Question 4 (4 points)
Interpret the results.

Question 5 (4 points)
What is one of the limitations of the language modeling objective used in this notebook, compared to the masked language model objective introduced in [1].

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] Moussa Kamal Eddine, Antoine J-P Tixier, and Michalis Vazirgiannis. Barthez: a skilled pretrained french sequence-to-sequence model. *arXiv preprint arXiv:2010.12321*, 2020.

[3] Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah, and Benoît Sagot. Camembert: a tasty french language model. *arXiv preprint arXiv:1911.03894*, 2019.

[4] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[5] 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.