Assignments
M3224.000300 Natural Language Processing with Neural Networks
Fall 2023

Due on Thursday Nov. 9, 2023 by 9:30am (before class)

(Assignment 4: Pre-trained models)

Graduate School of Data Science
Seoul National University
Instructor: Dr.Jay-yoon Lee (lee.jayyoon@snu.ac.kr)
TA: Yikyung Kim (k2y1513@snu.ac.kr)
TA: Seongil Park (athjk3@snu.ac.kr)

Honor Pledge for Graded Assignments

"I, YOUR NAME HERE , affirm that I have not given or received any unauthorized help on this assignment, and that this work is my own."

0 Instructions

- Total score cannot exceed 100 points. For example, if you score 95 points from non-bonus questions and 10 points are added from bonus questions, your score will be 100 points, not 105 points.
- Skeleton codes for each of the problems are at the directory `/q1` and `/q2`.
- Run the `bash collect_submission.sh` script to produce your 2000_00000_coding.zip file. Please make sure to modify collect_sumbssion.sh file before running this command. (**2000_00000** stands for your student id)
- Modify this tex file into `2000_00000_written.pdf` with your written solutions
- Upload both `2000_00000_coding.zip` and `2000_00000_written.pdf` to etl website. **(4pts)**
- **If submission instructions are not followed,** `4 points` **will be deducted.**

1 Utilizing pre-trained models (50 pts)

We learned how different pre-trained models work in class. In this assignment, we will load pre-trained models from Hugging Face, and use them for downstream tasks. Please complete the given jupyter notebook files **(q1/q1a.ipynb)** and **(q1/q1b.ipynb)**, and submit it along with your answer to this latex file.

1.1 Inference using a pre-trained Question Answering model (20 pts)

In this problem, we will utilize a pre-trained BERT(Bidirectional Encoder Representations from Transformers) model for question answering. BERT excels in this task by leveraging its contextual understanding of language, employing techniques such as tokenization, segment embeddings, positional embeddings, and multi-head self-attention to capture intricate relationships between words. During the inference phase, the model predicts answer spans in a passage based on its extensive pre-trained knowledge and fine-tuned task-specific understanding.

There are various approaches to employing deep learning models for question answering, and we will explore the intricacies of this task in upcoming lectures. For now, prepare for a hands-on journey into loading a pre-trained BERT model from Hugging Face, which encodes questions and context together and answers by selecting a span in the given context. The model's output includes scores for the start and end positions of the answer span. Follow the instructions in the provided Jupyter notebook to complete the code.

(a) Load the pre-trained model and tokenizer and complete the `__init__` function. **(2 pts)**

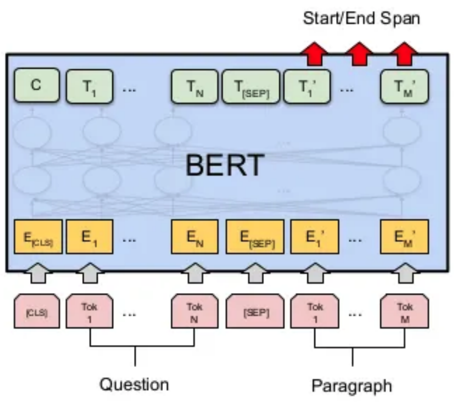

Figure 1: Question answering using BERT

(b) Implement the `tokenize` function to encode the question and context. **(6 pts)**

(c) Complete the `generateAnswer` function for inference. **(12 pts)**

(d) **(bonus)** Add a condition that ensures the end position comes after the start position in your inference code. **(4 pts)**

1.2 Fine-tune a pre-trained Image-to-Text model for Image Captioning (30 pts)

In this problem, you will load a pre-trained model, *pix2struct*, and fine-tune it to perform an image captioning task.
Pix2Struct is a pretrained image-to-text model for visual language understanding, and it has been fine-tuned on a variety of tasks and datasets, including image captioning and visual question answering. It is pretrained by learning to parse masked screenshots of web pages into simplified HTML. For more details about this model, please refer to the paper *Pix2Struct: Screenshot Parsing as Pretraining for Visual Language Understanding*: https://arxiv.org/abs/2210.03347
We will use a image captioning dataset from Hugging Face to fine-tune the model. The dataset contains Pokémon images with captions as below. Please refer to the jupyter notebook for detailed description.

Figure 2: (caption) a cartoon pikachu with a pink dress and a pink bow

(a) Load the image captioning dataset, and split into training and validation set. **(4 pts)**

(b) Complete the training code in `train()` function of Trainer class. **(10 pts)**

(c) Complete the validation code in `train()` function of Trainer class. **(6 pts)**

(d) After the training is complete, use the `plot()` function of Trainer class to display the figure, and then paste it to your latex file. **(6 pts)**

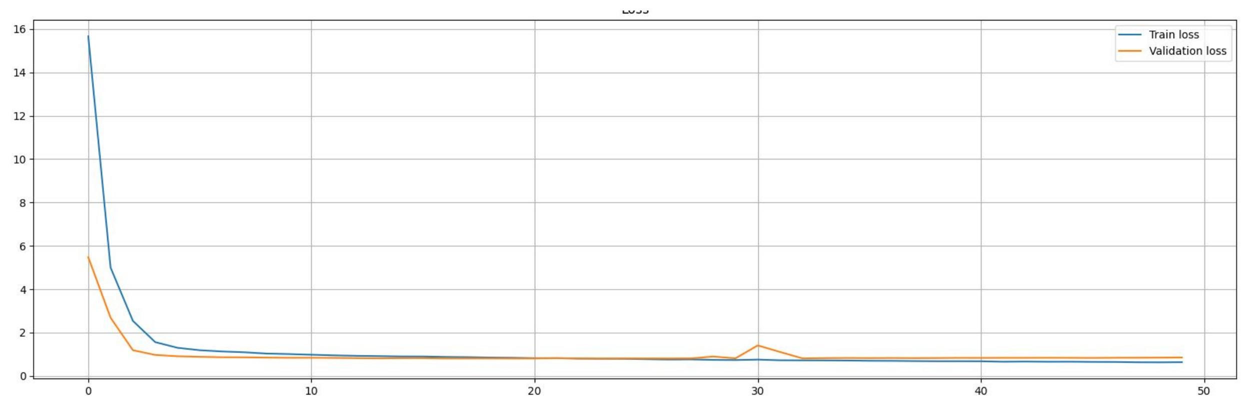

Figure 3: Train, Valid loss plot

(e) Based on your training results in (d), evaluate whether the training was successful, and write at least two ways to improve the model's performance. **(4 pts)**
Answer: loss가 일정하게 수렴 하므로 succesful
validation & train loss가 어느정도 satured 되었음으로 early stopping 통해 improve, lr을 낮추어서 한층 더 optimal에 가깝게 학습

2 Implement Minimalist Version of BERT (50 pts)

In this assignment, you will implement some important components of the BERT model to better understand its architecture. You will then perform sentence classification with the model you implemented. There are no written questions in this section. Since BERT is based on the transformer encoder architecture, for more details about transformer architecture, please refer to the paper *Attention is all you need*: https://arxiv.org/pdf/1706.03762.pdf
Please refer to `setup.md` for the environment setup before getting started. You are NOT allowed to use external libraries such as `transformers` in torch.

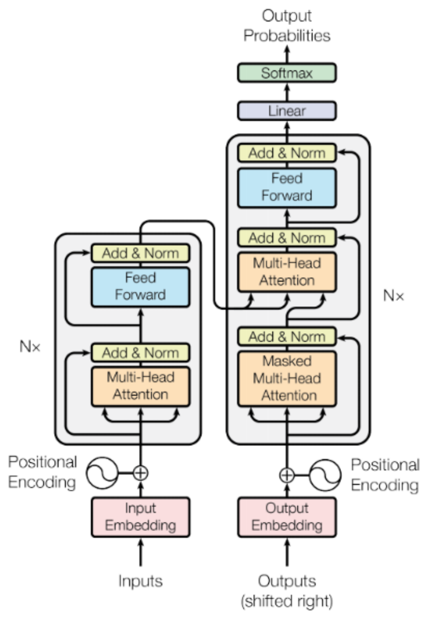

Figure 1: The Transformer - model architecture.

Figure 4: Model architecture of the transformer

(a) Implement `attention` function in `BertSelfAttention` class of `bert.py`. Scaled dot-product attention are computed as below:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

(10 pts)

(b) Implement `forward` function in `BertSelfAttention` class of `bert.py`. **(8 pts)**

(c) Implement `add_norm` function in `BertLayer` class of `bert.py`. It corresponds to Add & Norm Layer in the figure 4, which is defined as LayerNorm$(x + \text{Sublayer}(x))$ **(8 pts)**

(d) Implement `forward` function in `BertLayer` class of `bert.py`. It corresponds to encoder stacks which are described on the left of the figure 4. After completing the codes, run `python sanity_check.py` to check your implementation. **(8 pts)**

(e) Implement `train` function in `classifier.py`. This function carries out backpropagation through the model parameters. **(8 pts)**

(f) Show time! Run `python3 classifier.py --option finetune --epochs NUM_EPOCHS --lr LR --train data/sst-train.txt --dev data/sst-dev.txt --test data/sst-test.txt --use_gpu`
Each time you run the above code, a `'result.log'` file is created (be cautious as it overwrites the existing one). When submitting, make sure to attach the `'result.log'` file as well. **You don't need to submit the model checkpoint (.pt file) and *-output.txt files. (8 pts)**