NLPHW4

ssb3vk

December 2023

Question 1 Based on the description in Vaswani et al., 2017, please describe the major difference between the Transformer encoder and the Transformer decoder. As we explained in class, both encoder and decoder consists of the major components, such as feed-forward module and multi-head attention. Your answer should cover the following aspects

- (1 point) What is the major module difference between the Transformer encoder and the Transformer decoder?
  The major module difference between the transformer encoder and decoder is that the decoder has an additional sub-layer (in addition to the two sub-layers in the encoder layer) which performs multi-head attention over the output of the encoder. Additionally, the self-attention sub-layer in the decoder stack is modified so that positions cannot "attend" (I believe this means take-as-input) subsequent positions. This "Masked Multi-Head Attention" ensures that the predictions for position i can depend only on the outputs at positions less than i. This way, the decoder module can't "cheat" by looking ahead and generating text based off the correct output.
- (2 points) How that module is implemented?
  This module is implemented in the traditional multi-head attention way. As opposed to performing single attention with d_{model}-dimensional keys, values, and queries, the authors linearly project the queries, keys, and values h times with different, learned linear projections to d_q, d_k, and d_v dimensions. Then they perform the attention function in parallel, yielding d_v-dimensional outputs. All of these outputs are concatenated and re-projected yielding final values. We covered this in lecture and I distinctly remember your explanation:

  $$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

  where

  $$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

  This enables the model to attend to information from different representational sub spaces at different positions, exposing more information about the input.

The decoder makes a slight modification. To only allow the decoder to attend to positions in the decoder up to and including that position, the authors prevent leftward information flow by using scaled dot-product attention to mask out all values in the input of the softmax that correspond to illegal connections. They set all values that correspond to illegal connections to negative infinity.

- (1 point) Why the difference is critical, e.g., for machine translation?
  From a practical standpoint, the difference is critical as it decreases the per-layer computational complexity as well as offers greater opportunities for parallelization. But from a theoretical standpoint, the masked multi-head attention decreases the path-length between long-range dependencies in the network. Learning these long-range dependencies is key to sequence transduction (mapping symbol representations to another sequence of equal length). A key factor that affects the ability to learn these long range dependencies is the length of the forward and backward paths. By decreasing the overall path lengths, the forward and backward signals have a shorter overall path to traverse, enabling more easy learning of long-range dependencies. The self-attention layer connects all of these positions with a constant number of sequential operations whereas recurrent layers require O(n) sequential operations. A single convolutional layer does not connect all pairs of input and output positions.
  This is particularly relevant to machine translation because machine translation is fundamentally a sequence transduction task. And this theoretical justification is backed up by empirical results. Even their base model is able to surpass all previously published models and ensembles at a fraction of the training cost.

Question 2 Comparison between the BERT model and the GPT model (GPT-1, to be specific). Although both of them are based on the Transformer architecture. Their difference can be categorized at least in the following three aspects

- (1 point) Model architecture: following your answer to the previous question, please identify which specific Transformer (e.g., encoder, decoder, or both) is used in each of the two models.
  BERT uses only the Transformer's encoder structure. It's design enables it to process data bidirectionally so it can take context from both the left and right side of a token in the input sequence. GPT-1 is based only on the Transformer's decoder architecture. Specifically, it employs the unique masked multi-head attention used in the decoder architecture. This attention layer is capped by a position-wise feed forward layer to produce an output distribution over the token space.
- (2 points) Pre-training strategy: please summarize the training strategies used in each of these two models. Note that, your answer should not include fine-tuning strategies.

Bert's pretraining strategies do not use the traditional left-to-right or right-to-left language models. Instead, BERT is trained via two unsupervised tasks. The authors dub Task 1 "Masked LM". The authors raise a point here that a deep bidirectional model would be more powerful than either a left-to-right model or the concatenation of a left-to-right and a right-to-left model, normal conditional models can only be trained in one direction. Bidirectional conditioning would allow each word to "see itself" and allow the model to use the next training word as its prediction. So, to train a deep bidirectional representation, the authors randomly mask some percentage of the input tokens and force the model to predict those masked tokens. Note that this mask is not always a [MASK] token, sometimes it is simply a random token.

The second task is called "Next Sentence Prediction (NSP)". Both QA and NLI are based on understanding relationships between sentences which isn't captured well by natural language modeling, what Masked LM was training. To understand sentence relationships, the authors pre-train the model for binarized "next sentence prediction". Where 50% of the time the model is shown succeeding sentences and the other half of the time, the second sentence is a random sentence. I believe that here, the model is trained to respond positively to the pair of sequential sentences.

GPT-1's pretraining strategies are far simpler. As opposed to masking words or predicting sentence relationships, GPT is only trained on a sequence of tokens and is trained to tune the output distribution over the target tokens. A traditional log-likelihood (insert formula from section 3.1 of the GPT paper here).

- (1 point) Applications: what are the difference of these two models regarding NLP tasks. Please list the NLP applications that can highlight the difference between these two models. As a counter-example, "text classification" is not a good answer, as both of them can be used for classification. There are many NLP tasks, and your answer can only be the tasks that we described (at least, briefly) in class, including sentiment classification, topic classification, word embedding, language modeling, question answering, natural language inference, semantic role labeling, machine translation, text generation, conversation modeling.

  BERT Tasks:

  - General Language Understanding Evaluation

  - Question/Answer Pairs: given a question and a passage from wiki containing answer, task is to predict answer text span in passage

  - Squad 2.0. Extension above where no short answer exists.

  - SWAG: sentence-pair completion examples. Grounded commonsense inference. Task is to choose most plausible continuations among four choices.

GPT Tasks:

- NLI: Reading pair of sentences and judging relationship between them (entailment, contradiction, or neutral).
- Question Answering and common sense reasoning RACE.
- English passages with associated questions
- Story Cloze Test: Selecting correct ending to multi-sentence stories from two options
- Semantic Similarity: Involve predicting whether two sentences are semantically equivalent or not. rephrasing, negation, syntactic ambiguity
- Classification: CoLA, SST-2, GLUE. Senetenc is grammatically correct or not, binary classification task (sentiment),

In summary, Bert, due to its bidirectional nature is better at understanding the context of words in a sentence. This makes it really good at sentiment role labeling, natural language inference, and question answering.

GPT with its left-to-right text generation capabilities is far better at **generating** text. This includes text summarization, and machine translation.

Question 3 Prefix Tuning. In the topic of efficient fine-tuning, we talked about prefix tuning as one of the fine-tuning techniques. The basic idea of prefix-tuning is to introduce additional latent representations to each layer. During fine-tuning, these representations will be updated for task-specific applications.

- (1 point) Number of parameters introduced by prefix-tuning. Assume we are using prefix-tuning to fine-tune the GPT model. Given the dimension of latent representations is 768 and the number of prefix representations for each layer is 4, what are the total number of parameters introduced by prefix tuning? (Hint: only consider the case **without** using feed-forward NN for projection.)

  If the dimension of the latent representation is 768 and we are using 4 prefix representations for each layer, and we have 12 layers in our model (GPT-1), then we have 768*4*12 parameters introduced by prefix tuning.

- (2 points) Although the lack of theoretical justification, we can still explain the performance of prefix tuning by comparing it with discrete prompt optimization. What are the two main reasons that prefix tuning may outperform discrete prompt optimization in practice?

  Prefix tuning is meant to offer an efficient mode of fine-tuning a pretrained model without modifying the entire model. The key idea is to introduce a small number of task-specific parameters so that the model can adapt to new tasks without significantly changing its structure.

Prefix tuning outperforms discrete prompt optimization due to continuous representations. Prefix tuning allows for continuous latent representations, enabling smoother/nuanced adjustments to the model's behavior. That way, the model can adjust to the specific task it is being tuned for. Discrete prompt optimization on the other hand relies largely on fixed and discrete tokens. Handicapping the model's ability to capture variations in meaning and context, leading to subpar performance.

Additionally, prefix-tuning is far more invasive, adjusting the model's internal parameters. Discrete prompt optimization only really involves the input layer, and is therefore a weaker method of tuning the model. Further, this integration (that prefix-tuning offers) can build on the model's existing pre-trained knowledge (encoded in non prefix parameters). That way task-specific knowledge can better blend with the models' existing knowledge.

Basically, the two reasons that prefix tuning is better than discrete prompt optimization are the fact that it has a method of continuous optimization and that enables ultra-fine task-specific tuning that the discrete methods of DPO cannot match. The second reason is that the the ability to fundamentally modify the model's architecture is a far more powerful method of tuning. Yes, adjusting the prompt is cool, but changing the model to suit your needs is easily more expressive. The paper that your slides on prefix tuning are based off of.

Question 4 Multi-head Attention. Consider the procedure of computing multi-head attention (lecture 11, page 8).

- (1 point) Assume the query vector, key vector and value vector are all 512 dimensions. If we use multiple heads with $H = 8$, what are the dimensions of the three metrics $\{W_i^Q, W_i^K, W_i^V\}$ for each head i?

  The paper "attention is all you need" defines this very well. They define the parameter matrices as $W_Q^i \in R^{d_{model} \times d_k}$, $W_K^i \in R^{d_{model} \times d_k}$, $W_V^i \in R^{d_{model} \times d_v}$.

  In the paper's work they use h = 8 parallel attention layers/heads and they assume that the model's dimension keys are equal to 512. They then compute $d_k = d_v = d_{model}/h = 64$. Where $d_{model} = 512$. Therefore, we know that the dimensions of W_i^Q, W_i^K, and W_i^V are going to be 64x512, since each of these matrices map the original 512 dimensional space to the 64 dimensional space for each head.

- (1 point) Now, consider the collection $\{W_i^Q\}$, if we define W^Q as the stack of $\{W_i^Q\}$, then what is the dimensions of W^Q?

  If we define W^Q as the stack of W_i^Q's then the dimensions of W^Q is simply 512x512. This is because if we were to stack W_i^Q for all i, then

that is just a vertical concatenation of individual W_i^Q's. And because we have 8 of those, we return to the original W_Q size of 512x512.

Question 5 LoRA. The original proposal of using LoRA is only on the attention related metrics. Let us consider W^Q only in the following questions.

- (1 point) Based on the definition of LoRA, what are the dimensions of the adapter ΔW^Q for W^Q?

  The low rank adapter ΔW^Q is the same dimension as W_Q. page 19.

- (1 point) If we assume the dimensions of the query vector are 512, for low-rank estimation with $r = 4$, what are the dimensions of A and B in $\Delta W^Q = A \cdot B$?

  If we assume the dimensions of the query vector ar 512, and r = 4, the dimensions of A and B are $R^{4\times512}$ and $R^{512\times4}$ respectively.

- (2 points) What are the initialization strategies used for A and B? Any justification of these initialization strategies?

  The initialization strategies are that one of them (A or B) must be a zero matrix. That way ΔW is initially zero and therefore, we are able to start with the pre-trained model. If both of the matrices were to be non-zero, then we wouldn't start with the pre-trained model, instead we would effectively be starting from some other model. This allows a highly stable adaptation of the model to the specified task.

  Typically the non-zero matrix is seeded with some Gaussian distribution so that you have both negative and positive values.

  Additionally during training, the ΔW is scaled by alpha/r as a proxy for the learning rate.