Homework 4
CS 5787 Deep Learning
Naman Makkar (nbm49)
Spring 2023

Instructor: Prof. Alejandro (Alex) Jaimes

Discussion Section Leader: Jack Morris

TAs: Andrew Bennett, Yair Schiff

Graders: Eylul Ertay, Anastasia Sorokina, Zhengchun Shang, Jamie Cao

Due: 5/5/23, midnight

Your homework submission must cite any references used (including articles, books, code, websites, services (e.g., ChatGPT, GitHub copilot, etc), and personal communications). All solutions must be written in your own words, and you must program the algorithms yourself. **If you do work with others, you must list the people you worked with.** Submit your solutions as a PDF to Canvas.

Your homework solution must be typed, you may use LaTeXif you wish. Homework must be output to PDF format. I suggest using `http://overleaf.com` if you use LaTeXto create your document. Overleaf is free and can be accessed online.

Note that this assignment adds up to 15% of the total grade: 150 points. In addition, there are 50 points of extra credit (e.g., if you get 200 points that means you'd get the 15% for the class, plus the equivalent of another 5% that can be applied to other HWs or to your total grade).

If told to implement an algorithm, do not post your code to a public web repository (e.g., GitHub).

Part 1 - Concepts (150 points total, 10 points each)

Answers should be short (1-2 paragraphs; single sentence answers are not valid), concise, and contain sufficient technical details. No coding for this section.

1. What is a self-attention mechanism and how is it used in Transformers?

   Solution: Self-attention is a mechanism that allows the transformer model to weigh the importance of different tokens in a sequence relative to each other, and each token is compared with every other token to come up with an attention score. The attention score is calculated with the help of a similarity measure like the dot product between the query, key and value vectors which are obtained from the input tokens by multiplying them with weight matrices.

   For an input sequence of tokens $\mathbf{X} = x_1, x_2, \ldots, x_n$, where n is the length of the sequence, compute the query ($\mathbf{Q}$), key ($\mathbf{K}$), and value ($\mathbf{V}$) matrices:

   $$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$$
   $$\mathbf{K} = \mathbf{X}\mathbf{W}_K$$
   $$\mathbf{V} = \mathbf{X}\mathbf{W}_V$$

   where $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are learnable weight matrices. Self-Attention is given by:

   $$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

2. What is the difference between multi-head attention and single-head attention in Transformers?

   Solution: Single head attention computes a single set of attention weights while multi-head attention computes multiple sets of attention weights, each attending to a different part of the input sequence. Each head has its own set of learnable weight matrices - $\mathbf{W}_Q^i$, $\mathbf{W}_K^i$, and $\mathbf{W}_V^i$. The dot products are calculated independently for each head.

   $$\text{Attention}^i(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{(\mathbf{X}\mathbf{W}_Q^i)(\mathbf{X}\mathbf{W}_K^i)^T}{\sqrt{d_k}}\right)(\mathbf{X}\mathbf{W}_V^i)$$

   After the attention values are calculated for each head, they are concatenated and linearly transformed using another weight matrix $\mathbf{W}_O$.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{Attention}^1, \text{Attention}^2, \ldots, \text{Attention}^h)\mathbf{W}_O$$

3. Suppose that we design a deep architecture to represent a sequence by stacking self-attention layers with positional encoding. What could be issues?

   Solution: A few problems we could face -

   (a) **Complexity** - Self attention has a quadratic complexity with respect to the sequence length. Stacking self-attention layers with positional encodings would make the transformer model computationally expensive and memory intensive.

   (b) **Overfitting** - Increasing complexity could lead to overfitting on certain datasets and we would have to make use of regularization techniques to avoid overfitting.

   (c) **Limitations of Positional Encodings** - Positional Encodings are not optimal for very long input sequences due to the fact that they have a fixed size. We could make use of learned positional embeddings to address this issue.

4. How would you design a learnable positional encoding method?

   Solution:

   (a) Initialise an embeddings matrix with shape of the maximum allowed length of input sequences and initialise the weights using either Xavier or He initialisation.

   (b) Look up the positional embeddings for each input token from the embeddings matrix with the help of the position index of the token in the sequence.

   (c) Gradients for the positional embeddings matrix will be computed through back-propagation and the weights for the embeddings matrix will be updated and the optimal positional encodings for the task will be achieved.

5. Is it a good idea to replace scaled dot-product attention with additive attention in the Transformer? Why?

   Solution: Additive attention mechanism makes use of a feed-forward neural network to compute the attention scores instead of a dot product mechanism. This has a few benefits and drawbacks -

   Benefits -

   (a) Better for lower dimensional spaces and smaller models. It can compute more complex relationships between keys and queries than the simple dot product can.

(b) Bypasses the risk of high attention scores for high-dimensionality vectors.

Drawbacks -

(a) It is more complex than the dot-product attention and has many more parameters. This could lead to longer training times, increased memory requirement.

(b) It would be difficult to scale Transformer models with additive attention due to the increased complexity of the attention mechanism and increased parameters.

(c) Transformers are designed and heavily optimised for the more computationally efficient dot-product attention, and it might not be easy to integrate additive attention into a Transformer model.

6. For language modeling, should we use the Transformer encoder, decoder, or both? How would you design this method?

Solution: Language modelling is an autoregressive task that involves predicting the next token in the sequence, given the previous tokens. For this task using the Transformer encoder should be sufficient.

Here is how we could design this method -

(a) Tokenize the input into a sequence of tokens, convert the tokens into embeddings and add positional encodings to the embeddings. Pass these embeddings through a Transformer encoder.

(b) Apply causal (lookahead) masking to the self attention mechanism in the encoder layers to ensure that the self attention calculations are carried out on previous tokens and not on future tokens.

(c) Train the model using the cross-entropy loss to minimize the difference between the target tokens and the predicted tokens.

(d) Carry out inference with the trained model, starting with an initial token, iteratively predict the next tokens in the sequence.

7. What can be challenges to Transformers if input sequences are very long? Why?

Solution: The following challenges would be faced by the Transformer models if input sequences are very long -

(a) Self attention mechanism has a quadratic complexity with respect to the input sequence length. A hug input sequence would make the transformer model both compute and memory intensive. Architectures like Performer and Linformer aim to address the complexity of the self attention mechanism.

(b) Transformers are optimised to capture long range dependencies for a fixed maximum sequence length, increasing the length of the input sequence beyond that would mean that the Transformer would struggle to capture dependencies between the tokens.

8. Say that you are asked to fine tune a language model to perform text classification by adding additional layers. Where will you add them? Why?

   Solution: The best approach would be to add fully connected layers after the hidden layers of the model followed by either a softmax or sigmoid activation function. This is done since the hidden layers of the model have already been pretrained on a large dataset and have learned useful features and representations from the pretraining task. The fully connected layers added to the hidden pretrained layers can be utilised for finetuning for downstream tasks. The representations of the pretrained layers are leveraged for downsteam tasks while the additional fully connected layers act as task specific classifiers and learn to map the pretrained models representations to the task specific target classes.

9. All other things being equal, will a masked language model require more or fewer pretraining steps to converge than a left-to-right language model? Why?

   Solution: A **left-to-right language model** is trained to predict the next word in the sequence given the previous word, as a result it learns the context and the relationships between the words in this sequential process and learns the context and stucture of the language quickly.

   A **masked language model** on the other hand is trained on masked input sequences where it has to predict the masked words given the context provided to it by the unmasked words. Masking of the input sequence is random, meaning that the masked language model requires a larger number of input examples to learn the context and the structure of the language.

10. Machine translation has long been evaluated based on superficial matching between an output translation and a ground-truth translation. How would you design a measure for evaluating machine translation results by using natural language inference?

    Solution:

    (a) Generate the translated text using the Machine Translation system. Generate premise-hypothesis pairs for the translated sentences where the premise serves as the source sentence and the translated sentence serves as the hypothesis.

(b) Make use of a pretrained natural language inference model to predict the relationship between the premise and hypothesis pairs and calculate the percentage of pairs where the natural language inference model predicts entailment as the relationship between the premise and the hypothesis. The higher this percentage the better the machine translation, since it means that the translated sentences are semantically similar to the source sentences.

11. How can we leverage BERT in training language models?

Solution:

(a) **Finetuning** - We can make use of pretrained BERT models with added fully connected linear layers for fine-tuning on task-specific dataset.

(b) **Data Augmentation** - BERT can be utilised for generating additional training data for language models by generating paraphrases of the input texts, this aids in training language models when we have limited training data available.

(c) **Pretraining** - We can make use of BERT's masked language modelling capabilities to pretrain BERT like models on datasets in other languages.

(d) **Feature Extraction** - BERT can act as a feature extractor for down steam tasks, we can feed an input sequence to BERT and obtain contextualised embeddings for each token in the text which can be used as input features for traditional Machine Learning models like SVMs, Logistic Regression models.

12. Suppose that we have a trained model based on multi-head attention and we want to prune the least important attention heads to increase the prediction speed. How can we design experiments to measure the importance of an attention head?

Solution: We can design the following experiments to measure the importance of the attention heads -

(a) We can compute the average attention weight for each head, with the higher values indicating the importance of the attention head in capturing the representation of the input. Use the weights as an importance score.

(b) Prune the attention heads based on a threshold of importance score computed using the attention head's weights.

(c) Carry out post pruning training to finetune the pruned model after pruning the least important attention head and check the model performance on the validation set.

(d) Different pruning techniques like structured and unstructured pruning can also be tested and their performances compared.

(e) The most primitive way of approaching this problem would be to carry out an ablation study by pruning one attention head at a time and test the performance of the model on the validation set.

13. Consider sequence to sequence problems (e.g., machine translation) where the input sequence is always available throughout the target sequence prediction. What could be limitations of modeling with decoder-only Transformers? Why?

Solution: Limitations of a decoder-only Transformer -

(a) For a seq2seq problem the encoder is responsible for creating a meaningful representation of the inputs for the decoder which then generates a target sequence. Decoder only models do not have an encoder and therefore cannot utilise the encoded representation of the input sequence.

(b) In an encoder-decoder Transformer model the input and output sequences are explicitly connected by cross-attention mechanisms which help in capturing long range dependencies whereas the decoder-only Transformer model only makes use of self-attention mechanisms which is not as efficient in capturing long range dependencies between input and output sequences.

(c) Decoder only Transformer models would have tremendous difficulties in seq2seq problems involving different input and output modalities since for these problems the encoder and decoder are optimised for a specific modality, for example text-to-image or text-to-speech.

14. What is the difference between a pre-trained Transformer model and a fine-tuned Transformer model?

Solution: Both the pretrained and fine-tuned Transformer models have been trained on a large-scale dataset, however the fine-tuned transformer model is trained on a downstream task with the help of transfer learning with task-specific data. The fine-tuned model contains task specific head in the form of fully connected linear layers in addition to the layers of the pre-trained Transformer.

The pretrained Transformer captures general language understanding but is not specialised for specific tasks. Fine-tuning the pretrained model on a task-specific dataset with a task-specific head results in better performance on that task.

15. You are going to use a LLM for an NLP task. What are the different ways you can use it? if you fine-tune it, what are the options?

Solution:

(a) **Fine-Tuning** - The pretrained LLM can be trained on a downstream task on a task specific dataset. This involves adding a task specific head to the pretrained model and training the model on task-specific data. While finetuning -

i. We can either only train the task specific head.

ii. We could unfreeze the model and train the entire architecture on the task-specific data.

iii. We could gradually unfreeze the layers of the pretrained LLM as we progress in the training regime

(b) **Feature Extraction** - We could make use of the layers of the pretrained LLM and extract embeddings from it and feed them to a task specific model.

(c) **Few Shot Learning** - We can leverage the LLM's ability to generalise from a few examples and provide it with a few examples of the input-output format for training the pretrained model for few shot learning.

Part 2 - Hands-on with LLMs (50 points)

For this problem, you may work in groups of two, and you will use an open source LLMs to run some experiments (think of this as a mini-hackathon). There are many options, but for this assignment you may use the following repository:

https://github.com/ggerganov/llama.cpp

This implementation allows you to run LLMs locally on your machine (it supports multiple models). Depending on your hardware, it can be slow, so if you prefer to use something else that uses an API are you are free to do so (note that if you use an API to call an LLM you may have to pay- you should not spend more than a few dollars, if you decide to go for that option).

You may also use LangChain:

https://python.langchain.com/en/latest/index.html

See https://python.langchain.com/en/latest/modules/models/llms/integrations/llamacpp.html

IF you work with a partner, your submissions for this section should be identical. You must clearly indicate at the start of the section who your partners is. You should also both set up and run your experiments (the idea is for both of you to do the work!).

The work for this mini-hackathon was implemented by Naman Makkar (nbm49) and Nick Pacia (snp43)

This min-hackathon should consist of the following steps:

1. Get the model running for a basic task (e.g., to answer a question). 5 Points

   Solution: We made use of the OpenAI API and queried the GPT-3.5-Turbo model with the question - **"Who was Chandragupta Maurya ?"**, receiving the response - **Chandragupta Maurya was an Indian emperor who founded the Maurya Empire in ancient India. He was born in 340 BCE in the Magadha region of present-day Bihar, India. He was the first emperor to unify most of the Indian subcontinent under one rule.**

2. Run some basic experiments. Explain the experiments you ran and your observations (you may, for example, compare the results vs what you'd get on a public LLM like ChatGPT; e.g., if you're running a smaller model, what do you find?). 10 points.

   Solution: The following models of the GPT-3 family - **'text-davinci-003', 'text-davinci-002', 'text-babbage-001', 'text-curie-001'** were utilized for the experiments and were provided 4 queries each. The 'temperature' parameter was initialised to 0.7 in order to make the response of the models less deterministic. The following were the questions asked -

   (a) **"What are you capable of ?"**

   (b) **"What is 77 divided by 26?"**

   (c) **"Write a 'Hello World' statement in Python, Javascript, and C++"**

   (d) **"Reverse a linked list in Python"**

   It was observed that text-davinci-003 and text-davinci-002 were the best performing models whereas text-babbage-001 and text-curie-001 were the worst performing models based on the quality of their responses. This is consistent with the fact that Davinci is the largest of the models while Babbage and Curie are smaller and less capable GPT-3 models.

   The Davinci models were the only 2 models that gave correct responses for the arithmetic question and the coding prompt. The Babbage and Curie model treated the arithmetic question as a text completion prompt giving answers like '0.6667' and '3.14' in addition to which they were incapable of writing the correct code for

reversing a linked list in python. All of the queries and responses can be observed in the jupyter notebook submitted.

3. Additional implementation: use your imagination here to get the model to do something interesting.. it could be as simple as prompt-tuning (e.g., if you're using a smaller model running locally, what would you do to improve performance). 20 points

   Solution: A small debate was simulated where each of the above mentioned models was queried to present 3 arguments both for and against a specific topic, and we judge the model on the quality of their arguments. A temperature of 0.7 was chosen in order to discourage deterministic and repetitive responses from the models.

   The first debate topic chosen was 'canards on a stealth fighter', while the second topic chosen was 'self-driving cars' and all the 4 models came up with 3 pros and cons for both the topics.

4. Experiments and observations on what you implemented. Run some tests, explain your results and how they could be improved. 15 points.

   It was observed that all the models were more or less very repetitive with their arguments in favour of the topic. All of the models apart from text-curie-001 understood the context of the topic. For the arguments against the topic 'canards on a stealth fighter', the Curie model started providing arguments against stealth fighters while the other models correctly understood the context of the topic and provided the reasons against having canards on the stealth fighter. All of the models directly contradict themselves when providing pros and cons for 'canards on a stealth fighter', they all made the same argument in favour of canards by stating that canards reduce the radar cross section of the stealth fighter and improve the stealth characteristics and then listed a con as the worsening of stealth characteristics due to canards. This was expected since this is a problem with ChatGPT itself makes these contradictions. Additionally, there is an ongoing debate in the Aerospace industry about the viability of canards on a stealth fighter with most aerospace giants in agreement with its adverse effects on stealth.