(CSE 5539) Special Topics in Large Language Models

First (and Last) Homework

For homework deadline and collaboration policy, check the course website*

Name: _____

Sources used for your homework, if any: _____

This assignment will assess your understanding of the foundational concepts related to Large Language Models (LLMs). The purpose of this assignment is to make sure you are prepared for this course. We anticipate that each of you will have different strengths and weaknesses, so don't be worried if you struggle with *some* aspects of the assignment. But if you find this assignment to be very difficult overall, that is an early warning sign that you may not be prepared to take this course at this time.

How to hand in your written work: via Canvas.

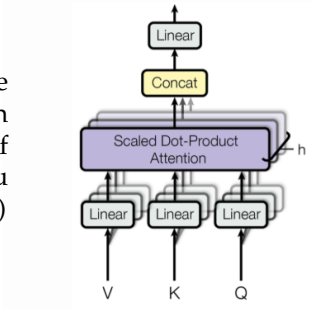
How to run experiments: You may use own personal device, Google Colab, OSC, or any other computing resources available to you. Please post on the course team if you need assistance.

Collaboration: You may discuss the homework with your classmates, if that helps you understand the problems and various ideas in the class, but you are **not allowed to share problem solutions with any other students. You must write the solutions individually**. The use of AI (e.g., Codex) is permitted but **must be documented**.

Typesetting: We strongly recommend typesetting your homework!

1 Understanding Transformer Architecture

1.1 Theory: Self-Attention



Write the equations governing Multi-Headed Self-Attention module (shown in the following figure). Show your answer in matrix form and explain the role of each parameter. Make sure to also comment on how this equation can handle batching of many inputs. Feel free to use the original paper [Vaswani et al., 2017], though you need to explain it in your own words (You may refer to the content in the 5525 class.)

Answer: *Your answer here ...*

2 Understanding Language Models

2.1 Theory: Perplexity

Recall the definition of perplexity:

$$\text{ppl}(w_1, ..., w_n) = 2^H \text{ where } H = -\frac{1}{n}\sum_{i=1}^{n} \log_2 p(w_i | w_1, \ldots, w_{i-1}),$$

for a given sequence of tokens $(w_1, \ldots, w_n)$. Assume we compute perplexity *per document* and then average these values over a large natural language corpus containing N documents and a total vocabulary size of V unique words.

- What is the minimum possible average document-level perplexity, and under what conditions is it achieved?
- Is there a finite maximum? Explain.

*https://shocheen.github.io/cse-5539-fall-2026/

Justify your answer using the formula for perplexity and explain your reasoning.
Answer: *Your answer here ...*

2.2 Theory: Extracting Text from Language Models

You're given a language model that is currently set to use greedy decoding (selecting the word with the highest probability). When decoding from this model, explain when/why you may want to use either of the following settings with sampling:

(a) a zero temperature

(b) temperature of 1

(c) temperate higher than 1

(d) nucleus (top-p) sampling

(e) top-k sampling

Answer: *Your answer here ...*

2.3 Empirical: Measuring Perplexity and Sampling Strategies

Use a pretrained language model from HuggingFace (we suggest DistilGPT2) and perform the following:

(a) **Perplexity Analysis:** Choose a short English paragraph (e.g., 3 to 5 sentences). Tokenize it and compute its perplexity using the model's log-likelihood output. Then compute the perplexity of a randomly shuffled version of the same paragraph. Comment on the difference.

(b) **Sampling Comparison:** Using the prompt "Once upon a time", generate continuations (length=150 tokens) using:

- greedy decoding,
- sampling with temperature $T \in \{0, 0.3, 0.6, 0.9, 1.2, 1.5\}$.

Report the outputs and briefly compare their diversity and quality.

Include a link to your GitHub code or Colab notebook.
Answer: *Your answer here ...*

3 Understanding Optimization

Look at the definition of [Stochastic] Gradient Descent in PyTorch: https://pytorch.org/docs/stable/generated/torch.optim.SGD.html. Let's see if we understand a few nuances here.

input : γ (lr), θ_0 (params), $f(\theta)$ (objective), λ (weight decay), μ (momentum), τ (dampening), *nesterov*, *maximize*

for $t = 1$ **to** ... **do**
  if *maximize* :
    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
  else
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  if $\lambda \neq 0$
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  if $\mu \neq 0$
    if $t > 1$
      $\mathbf{b}_t \leftarrow \mu\mathbf{b}_{t-1} + (1-\tau)g_t$
    else
      $\mathbf{b}_t \leftarrow g_t$
    if *nesterov*
      $g_t \leftarrow g_t + \mu\mathbf{b}_t$
    else
      $g_t \leftarrow \mathbf{b}_t$
  $\theta_t \leftarrow \theta_{t-1} - \gamma g_t$

return θ_t

3.1 Maximization vs Minimization

Notice the `maximize` parameter which controls whether we are running a maximization or minimization. In the algorithm, the effect of this parameter is essentially shown as a change in the sign of the gradients. How do you think this change leads to the desired outcome (maximization vs minimization)?
Answer: *Your answer here ...*

3.2 Empirical: Visualizing SGD Behavior

Use PyTorch to optimize a simple 2D function such as $f(x, y) = x^2 + y^2$ (minimization) or $f(x, y) = -x^2 - y^2$ (maximization). Run gradient descent using `torch.optim.SGD` under the following settings:

(a) Vary `momentum` between 0 and 0.9, and visualize the trajectory of (x, y) on the contour plot of the function.

(b) Repeat (a) with `weight_decay` set to a small value (e.g., 0.1). What changes do you observe?

(c) Try setting `maximize=True` on $f(x, y) = -x^2 - y^2$. What changes do you observe?

Generate 2 to 3 plots to support your discussion. Include a link to your code and briefly interpret what each knob changes about the optimization behavior.
Answer: *Your answer here ...*

4 Understanding Fine-tuning Models

In this assignment we will try different ways of fine-tuning models. Consider the SST2 dataset in Huggingface. Train a ModernBERT model for this classification task.

4.1 Empirical: Building a Head-Tuned Classifier

Build a classifier based on ModernBERT and fine-tune the classification head only (**not** the model weights) so that the accuracy is maximized for this task. Plot the accuracy on train and dev (validation) sets over the course of training. Report the results on the test set corresponding to your best model measured on the dev (validation) set in Table 1. Include the results in Table 1. Include a link to your code on Github or Google Colab. We recommend using Huggingface Transformers for this but you are welcome to use any libraries.
Answer: *Your answer here ...*

4.2 Theory and Empirical: Building a LoRA Classifier

- Repeat (4.1) with LoRA. **Make sure that the number of trainable parameters in your LoRA is similar ballpark as that of (4.1).** Include the results in Table 1. Include a link to your code on Github / Colab.

Approach	Accuracy (validation)
Head tuning	
LoRA	

Table 1: Results of training ModernBERT on SST2.

Answer: *Your answer here ...*

5 Understanding Efficiency in Transformers

5.1 Theory: Prefill vs Decode

What are **prefill** and **decode** phases in autoregressive generation? Why are they fundamentally different in how they compute attention? Which part makes autoregressive inference inefficient? Look up one idea used in practice to speed up generation despite this inefficiency.

Answer: *Your answer here ...*

5.2 Theory: KV Cache Bottlenecks

In standard multi-head attention, we store and reuse key/value vectors in a **KV cache** to avoid recomputing them. Why does this cache grow linearly with sequence length? Suggest one way to reduce memory or bandwidth cost. What are the potential trade-offs of doing so?

Answer: *Your answer here ...*

References

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* (NeurIPS), 2017. URL https://arxiv.org/abs/1706.03762.