ELL8299/ELL881/AIL861

Advances in Large Language Models

Semester I, 2025-26

Answer the questions in the spaces provided. No extra pages will be given.
Answers should be brief and to-the-point.

Course Code: _____ Name: _____ Entry Number: _____

Total Marks: 20 **Time: 40 minutes**

1. What is a language model? Write the expression for perplexity of a sentence $\mathcal{S}$ (consisting of a sequence of tokens $\{t_1, t_2, ..., t_n\}$) for a trigram language model. **(1+1 = 2 marks)**

   Answer:

   - A probabilistic model which assigns a probability to a sequence of tokens, i.e. computes $\mathrm{P}(t_1, t_2, ..., t_n)$ for a sequence of tokens $\{t_1, t_2, ..., t_n\}$, is called a language model.
   - $\mathrm{PPL}(\mathcal{S}) = \mathrm{P}(t_1, t_2, ..., t_n)^{-\frac{1}{n}} = \left[\frac{1}{\mathrm{P}(t_1)\mathrm{P}(t_2|t_1)\Pi_{i=3}^{n}\mathrm{P}(t_i|t_{i-1}, t_{i-2})}\right]^{\frac{1}{n}}$

2. (a) Why do Transformers use positional encoding while LSTMs don't? Explain briefly. **(2 marks)**
   (b) Write the difference between BERT and GPT in terms of pre-training objective. **(1 mark)**

   Answer:

   (a) Transformers use positional encoding because their self-attention mechanism is permutation-invariant, hence no information regarding the order of the tokens is captured by self-attention. LSTMs, on the other hand, don't need separate positional encoding as their architecture is inherently sequential, thus naturally capturing token ordering.

   (b) BERT is pre-trained primarily using the **masked language modeling** objective, where a certain percentage of tokens are randomly masked and the model is trained to predict these masked tokens by considering the bidirectional context. In addition to it, BERT is also trained using the **next sentence prediction** objective. GPT, on the other hand, is pre-trained using the **causal language modeling**, or, next-token-prediction objective. It is trained to predict the next token in a sequence given only the preceding tokens.

3. Suppose you are given an instruction tuning dataset $\mathcal{D} = \{(\mathbf{P}_i, \mathbf{R}_i)\}_{i=1}^{N_\mathcal{T}}$ consisting of $N_\mathcal{T}$ (prompt, response) pairs, where $\mathbf{P}_i$ is the input prompt and $\mathbf{R}_i$ is the corresponding ground-truth response. Assume that the number of tokens in a sequence $\mathbf{S}$ is denoted as $|\mathbf{S}|$; thus, we can expand $\mathbf{P}_i$ and $\mathbf{R}_i$ as:

$$\mathbf{P}_i = \{p_i^{(1)}, p_i^{(2)}, \ldots, p_i^{(|\mathbf{P}_i|)}\},$$
$$\mathbf{R}_i = \{r_i^{(1)}, r_i^{(2)}, \ldots, r_i^{(|\mathbf{R}_i|)}\}$$

   Write an expression for the loss optimized during instruction tuning of an auto-regressive language model on the given dataset $\mathcal{D}$. **(2 marks)**

   Answer: During instruction tuning, the cross-entropy loss over the response tokens is considered, the expression for which can be formalized as:

$$\mathcal{L}_{IT} = \frac{-\sum_{i=1}^{N_\mathcal{T}} \sum_{j=1}^{|\mathbf{R}_i|} \log\ \mathbb{P}_\mathcal{M}\left(r_i^{(j)} \mid \mathbf{P}_i, r_i^{(1)}, \ldots, r_i^{(j-1)}\right)}{\sum_{i=1}^{N_\mathcal{T}} |\mathbf{R}_i|}$$

4. Give one advantage of subword tokenization methods (BPE/WordPiece/Unigram) over pure word-level tokenization. **(1 mark)**

   Answer: Subword tokenization methods are more robust and effcetive in handling out-of-vocabulary (OOV) words, compared to pure word-level tokenization techniques.

5. Assume that a decoder-only Transformer-based language model (LM) consists of 12 Transformer layers, and each multi-head self-attention block in every layer has 12 attention heads (the overall embedding dimension is divided equally among the 12 heads). The MLP/Feed-forward block in each layer has an intermediate hidden state dimension of $d_{hidden} = 3072$. The LM has an embedding dimension of $d = 768$ and a vocabulary size of $V = 50257$. It uses trainable positional encoding and is designed to handle only a maximum sequence length of $L = 1024$. At the end of 12 Transformer layers, there is another LayerNorm block. The embedding and unembedding matrices are tied (i.e., they are transpose of one another). Assuming no separate bias terms during attention computations or for the MLPs, what is the total number of trainable parameters in this LM? **[Hint: Each transformer layer consists of 2 LayerNorm blocks, and each LayerNorm block has 2 trainable parameters for each embedding dimension. Also, don't forget to include the projection matrix applied to the concatenated output of the 12 heads.] (3 marks)**

   Answer:

   - **Embedding layer:** $d \times V$ (Since the embedding and unembedding matrices are tied, we don't need to include unembedding matrix parameters separately.)
   - **Positional encoding layer:** $L \times d$
   - **Each Transformer layer:** $4d^2 + 4d + 2d_{hidden}d$
     - **Attention block**: For every attention head, the embedding dimension is $\frac{d}{12}$. Hence, each of W_Q, W_K and W_V matrices are of dimension $d \times \frac{d}{12}$. For 12 attention heads, number of parameters $= 12 \times \left(3 \times d \times \frac{d}{12}\right) = 3d^2$. The projection matrix is of dimension $d \times d$. Hence total number of parameters in the attention layer $= 3d^2 + d^2 = 4d^2$.
     - **LayerNorm blocks**: There are 2 LayerNorm blocks in each Transformer layer - one after the attention block and one after MLP. Hence total number of parameters $= 2 \times (2 \times d) = 4d$.
     - **MLP**: The up-projection matrix of the MLP is of shape $d \times d_{hidden}$ and the down-projection matrix is of shape $d_{hidden} \times d$. Hence, total number of parameters $= 2 \times d_{hidden} \times d$.
   - **Last LayerNorm:** $2 \times d$

   Total number of parameters $= dV + Ld + 12(d^2 + 4d + 2d_{hidden}d) + 2d$
   $= (768 \times 50257) + (1024 \times 768) + 12 \times (4 \times 768^2 + 4 \times 768 + 2 \times 3072 \times 768) + 2 \times 768$
   $= 124,356,864 = 124.35$ Million.

6. (a) What is the primary difference between aligning LLMs with human preferences using *proximal policy optimization* (PPO) and *direct preference optimization* (DPO)? **(1 mark)**
   (b) Write the objective function corresponding to the alignment process using PPO. Assume π_{ref} to be the instruction-tuned model, and π_θ to be the model we are aligning. Also, assume that we already have a trained reward model which assigns reward $r(x, y)$ for an output y corresponding to the input x. **(2 marks)**
   (c) What is the primary advantage of *group relative policy optimization* (GRPO) over PPO for LLM alignment? Write the objective function for GRPO and contradict it with the PPO-objective you wrote in (b). **(1+2 = 3 marks)**

   Answer:

   (a) The primary difference is that **Proximal Policy Optimization (PPO)** is a multi-stage reinforcement learning (RL) process requiring a separate explicitly trained reward model, whereas **Direct Preference Optimization (DPO)** bypasses the explicit reward modeling and RL steps by deriving a direct loss function from the preference data and optimizing the language model's policy directly to satisfy those preferences, effectively treating the language model as an implicit reward model.

(b) The practical PPO algorithm for LLM alignment optimizes a *clipped surrogate objective* to ensure stable updates. This objective uses an *advantage estimate*, $\hat{A}^{\text{PPO}}(x, y)$, which is calculated with the help of a learned value function (the critic), V_ϕ. The objective function to maximize is:

$$\mathcal{L}^{PPO}(\theta) = \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_{ref}}\left[\min\left(\frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}\hat{A}^{\text{PPO}}(x,y),\ \text{clip}\left(\frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}^{\text{PPO}}(x,y)\right)\right]$$

where, the expectation is taken over prompts x from a distribution $\mathcal{D}$ and responses y sampled from the reference policy π_{ref}. $\frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}$ is the importance sampling ratio. $\hat{A}^{\text{PPO}}(x, y)$ is the advantage, typically an estimate like the Generalised Advantage Estimate (GAE). It's calculated on a per-token basis using the reward model $r(x, y)$ and the critic V_ϕ, which estimates the expected future reward at each step of the generation. ε is a small hyperparameter that defines the clipping range.

(c) The primary advantage of GRPO is that it is critic-free. Unlike standard PPO, which requires training a separate value function (a critic) to estimate advantages, GRPO calculates advantages by normalizing the rewards of multiple responses sampled for the same prompt *relative to each other*.

The optimization objective for GRPO uses the same clipped surrogate structure as PPO, but with a different, critic-free advantage calculation. For a given prompt x, we first sample K completions $\{y_1, y_2, \ldots, y_K\}$ from π_{ref}. The advantage for a specific completion y_i is its normalized reward relative to the group:

$$\hat{A}^{\text{GRPO}}(x, y_i) = \frac{r(x, y_i) - \mu_{\mathbf{r}}}{\sigma_{\mathbf{r}} + \epsilon} \quad \text{where} \quad \mu_{\mathbf{r}} = \frac{1}{K}\sum_{j=1}^{K} r(x, y_j) \quad \text{and} \quad \sigma_{\mathbf{r}} = \sqrt{\frac{1}{K}\sum_{j=1}^{K}(r(x, y_j) - \mu_{\mathbf{r}})^2}$$

The GRPO objective to be maximized is then:

$$\mathcal{L}^{GRPO}(\theta) = \mathbb{E}\left[\min\left(\frac{\pi_\theta(y_i|x)}{\pi_{ref}(y_i|x)}\hat{A}^{\text{GRPO}}(x,y_i),\ \text{clip}\left(\frac{\pi_\theta(y_i|x)}{\pi_{ref}(y_i|x)}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}^{\text{GRPO}}(x,y_i)\right)\right]$$

Contrast with PPO: PPO calculates advantage using a learned critic $V_\phi(x)$ that provides an absolute baseline for the reward. The advantage $\hat{A}^{\text{PPO}}(x, y)$ is typically a per-token estimate that measures how much better the actual outcome following a token generation was compared to what the critic predicted for that state. GRPO, being critic-free, computes a relative advantage $\hat{A}^{\text{GRPO}}(x, y_i)$ by comparing a response's reward to the mean and standard deviation of rewards from a group of peer responses generated at the same time.

7. Explain briefly how gradient checkpointing works. Why do we use gradient checkpointing during training of large models? **(2+1 = 3 marks)**

Answer: Gradient checkpointing is a technique used to reduce the memory consumption of training large models by trading computation for memory. In a standard forward pass, all intermediate activations are stored to be used during the backward pass for gradient calculation. Gradient checkpointing modifies this by:

(i) **Selective Saving:** During the forward pass, it avoids storing the intermediate activations for designated *checkpointed* layers. It only saves the inputs to these layers, or segments of layers.

(ii) **Re-computation:** During the backward pass, whenever the gradients for a checkpointed layer are needed, it recomputes the activations for that layer by running a localized forward pass starting from the nearest saved input. Once the gradients are computed with these re-calculated activations, the activations are discarded.

The primary reason for using gradient checkpointing is to drastically reduce the memory required for training. The activations of deep neural networks can consume a huge amount of GPU memory. By not storing them and recomputing them instead, we can train much larger models or use larger batch sizes than would otherwise fit into the available hardware memory, at the cost of a moderate increase in training time.