Milestone 1 Report

Section 2: Background

Answers:

1. The four main aspects are: pre-training (how to pretrain a capable LLM), adaptation (how to effectively adapt pre-trained LLMs for better use), utilization (how to use LLMs for solving various downstream tasks) and capability evaluation (how to evaluate the abilities of LLMs and existing empirical findings).
2. When the parameter scale exceeds a certain level, these enlarged language models not only achieve a significant performance improvement but also show some special abilities that are not present in small-scale language models. To discriminate the difference in parameter scale, the term large language models (LLM) is for the pre-trained language model (PLM) of significant size.
3.

- Encoder-decoder Architecture: it consists of two stacks of Transformer blocks as the encoder and decoder, respectively. The encoder adopts stacked multi-head self-attention layers to encode the input sequence for generating its latent representations, while the decoder performs cross-attention. Examples include the vanilla transformer.
- Causal Decoder Architecture: This architecture incorporates the unidirectional attention mask, to guarantee that each input token can only attend to the past tokens and itself. Examples include GPT-series models.
- Prefix Decoder Architecture: This architecture revises the masking mechanism of causal decoders, to enable performing bidirectional attention over the prefix tokens and unidirectional attention only on generated tokens. Examples include GLM-130B and U-PaLM.

4. Language modeling is a probabilistic models to predict the next word or token in a sequence, given the preceding words or tokens.
   Causal language modeling predicts the next word based on previous words. Mainly used for text generation.
   Masked language modeling predicts a masked word in a sentence using its context. Mainly uses this technique for pre-training.
5. Text classification, is a task in Natural Language Processing, where a given piece of text is assigned to one or more predefined categories or labels. The goal is to automatically classify the text into one or several categories based on its content. Various models could be used depending on the specific tasks, including traditional ones(SVM, etc.) based on TF-IDF, RNN, LSTM, and transformers.
   When using models like BERT or other transformer-based models for a new downstream text classification task, it's common to retrain or "fine-tune" the classification head. For a specific classification task, classification head (a few dense layers) would typically be added to the pre-trained model, and get fine-tuned to adjust the weights.
6. Summarization task involves condensing a longer piece of text into a shorter version, retaining only the most crucial information. The goal is to capture the essence of the original content in fewer words. T5, BERT and GPT can be fine-tuned for summarization tasks.
7. Adam and AdamW are optimizers based on adaptive estimates of lower-order moments for first-order gradient-based optimization. They are widely used to adjust the learning rates of each parameter based on the historical information of its gradients. This adaptive learning rate often leads to faster convergence and less sensitivity to the initial learning rate setting, which could save time, memory and energy, especially in large-scale traing processes. Due to its need to apply first-order and second-order momentums to further adjust parameters dynamically, it would introduce overhead. They need to maintain moving averages of past gradients and past squared gradients. This means for each parameter, Adam and AdamW store two additional values, leading to a total memory requirement of 3N.
8. Learning rate scheduler is used to adjust the learning rate during training based on the number of epochs or the number of batch iterations. Sometimes starting with a higher learning rate and reducing it later can help the model converge faster and achieve a better final performance. The torch.optim.lr_scheduler.CosineAnnealingLR scheduler in PyTorch modifies the learning rate using a cosine annealing schedule. It starts with an initial learning rate, decreases it following a cosine curve until it gets near a minimum value (eta_min), and repeats this for T_max epochs. This approach can help navigate the loss landscape and potentially avoid local minima.
9. Tokenization is a preprocessing method widely used in NLP. It aims to segment raw text into sequences of individual tokens, which are subsequently used as the inputs of LLMs. In traditional NLP research, word-based tokenization is the predominant approach, which is more aligned with human's language cognition. However, wordbased tokenization can yield different segmentation results for the same input in some languages (e.g., Chinese word segmentation), generate a huge word vocabulary containing many low-frequency words, and also suffer from the "outof-vocabulary" issue. Thus, several neural network models employ character as the minimum unit to derive the word representation (e.g., a CNN word encoder in ELMo). Recently, subword tokenizers have been widely used in Transformer based language models, typically including BytePair Encoding tokenization, WordPiece tokenization and
    Unigram tokenization.
10. LLaMA was trained on a diverse dataset including English CommonCrawl(67%), C4(15%), Github(4.5%), Wikipedia(4.5%), Books(4.5%), ArXiv(2.5%), and StackExchange(2.0%). Training set for LLaMA 65B and LLaMA 33B is 1.4 trillion tokens. For smallest model, LLaMA 7B, is trained on one trillion tokens.

11. Perplexity is defined as the exponentiated average negative log-likelihood of a sequence.
    Given a sequence $X = (x_0, x_1, ..., x_t)$,

$$PPL(X) = \exp\{-\frac{1}{t}\sum_{i}^{t}\log p_\theta(x_i|x_{<i})\}$$

    In the formula, each of the item $\log p_\theta(x_i|x_{<i})$ is the log-likelihood of the i-th token conditioned on the preceding tokens according to our model. Intuitively, it can be thought of as an evaluation of the model's ability to predict uniformly among the set of specified tokens in a corpus. Importantly, this means that the tokenization procedure has a direct impact on a model's perplexity which should always be taken into consideration when comparing different models.
12. Generating text from LLM can be done using various decoding methods.

- Greedy Search: Selects the highest probability word at each timestep, but may miss high probability words hidden behind a low probability word.
- Beam Search: Keeps the most likely `num_beams` hypotheses at each timestep, selecting the hypothesis with the highest overall probability, offering a more robust alternative to greedy search.
- Sampling: Picks the next word randomly based on its conditional probability distribution, potentially leading to incoherent text. Adjusting the `temperature` parameter can control the randomness.
- Top-K Sampling: Filters the K most likely next words and redistributes the probability mass among them. However, a fixed K may lead to sub-optimal choices of next words.
- Top-p Sampling: Dynamically chooses a set of words whose cumulative probability exceeds a threshold p, adapting to the next word's probability distribution.

13. Prompt learning refers to a method of training machine learning models, especially large language models like those in the GPT series. Instead of fine-tuning a model on a specific task with labeled data, prompt learning involves providing the model with a series of prompts or questions and having the model generate responses based on its pre-trained knowledge.
14. In-context learning refers to a process where a system learns from the specific environment or situation it's in, rather than from explicit instructions or pre-defined datasets. It allows for adaptive behavior based on immediate surroundings or experiences. In AI, it's exemplified by models that adjust their responses based on immediate user input or context.
15. These terms refer to the number of examples provided to the model to guide its response in the process of prompt learning.

- Few-shot: The model is given several examples to infer the desired task.
- One-shot: Only one example is provided.
- Zero-shot: No examples are given; the model is expected to understand the task solely from the prompt.

16. Instruction tuning is a method employed to refine the performance of LLM to better align with human preferences or specific tasks.
    LLaMA is fine-tuning on instructions data rapidly leads to improvements on MMLU(Massive Multitask Language Understanding). They observe that a very small amount of fine-tuning improves the performance on MMLU, and further improves the ability of the model to follow instructions.
    Instruction tuning is a form of supervised training because it involves providing the model with specific instructions and then adjusting the model based on the feedback or the desired outcomes.
17. As is mentioned in the descriptions, Alpaca is a dataset of 52,000 instructions and demonstrations generated by OpenAI's text-davinci-003 engine. This instruction data can be used to conduct instruction-tuning for language models and make the language model follow instruction better.
    Examples:
    `Instruction: Explain why the following fraction is equivalent to 1/4.`
    An instruction describes the task the model should perform. Each of the 52K instructions is unique.
    `Input: 4/16`
    An input is an optional context or input for the task. For example, when the instruction is "Summarize the following article", the input is the article. Around 40% of the examples have an input.
    `Output: The fraction 4/16 is equivalent to 1/4 because both numerators and denominators are divisible by 4. Dividing both the top a`
    This is simply the answer to the instruction as generated by text-davinci-003.
    An extra section is text, which is the combination of the instruction, input and output formatted with the prompt template used by the authors for fine-tuning their models.
18. Human alignment in LLM is the process of adjusting the models to align with human values, minimizing biases, and ensuring that they are useful and not harmful to users. It's important as it improves usability, mitigates misunderstandings and biases, and allows for better control over the models in accordance with human expectations.

Section 3: Preliminary

3.1 Gradient Accumulation

1. For each mini batch, the loss function is

$$Loss_k = \frac{1}{8 \cdot 4} \sum_{i=8k+1}^{8k+8} (y_i - \hat{y_i})^2, k = 0, 1, 2, 3$$

   If we add the losses of all the batches together, it would be:

$$TotalLoss = \frac{1}{32} \sum_{i=1}^{32} (y_i - \hat{y_i})^2$$

   And that would be the same with the loss of the whole batch.
2. In essence, it is natural that the results may differ, since the mean and standard deviation might be different in each mini batch.
   Consider $\hat{y_i} = Wx_i + b_i$. If $x_i' = \frac{x_i - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}}$, then $\hat{y_i}' = W\frac{x_i - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}} + b_i$.
   In the loss function mentioned above,

$$Loss_k = \frac{1}{32} \sum_{i=8k+1}^{8k+8} (y_i - \hat{y_i}')^2, k = 0, 1, 2, 3$$

   We cannot simply sum them up together to get the total loss.

3.2 Gradient Checkpointing

1. During model inference, we do not have to store the gradient used for calculations in the back propagation process, which might also involve gradient accumulation mentioned above.
2. The poor strategy: $O(1)$ memory, $O(n^2)$ computation steps.
   The general strategy: $O(\sqrt{n})$ memory, $O(n)$ steps.

3.3.1 SVD and truncated SVD

1. Matrix rank is the maximum number of linearly independent rows or columns in a matrix.
2. The Singular Value Decomposition (SVD) decomposes a matrix into three other matrices, A is represented as $A = U\Sigma V^T$:

- U: An orthogonal matrix containing the left singular vectors.
- Σ: A diagonal matrix containing the singular values.
- V^T: An orthogonal matrix containing the right singular vectors.

3. When an orthogonal matrix is multiplied by its transpose, the result is the identity matrix. This is a fundamental property of orthogonal matrices.
4. Rank is n
5. Keeping only the top-k singular values, it is essentially performing a form of lossy compression or dimensionality reduction on the image data.
   As k increases, more singular values are retained, leading to a less compressed representation of the image. The reconstructed image will be a closer approximation to the original image, retaining more of the original detail.
6. If the rank is r which r < n, The singular value matrix will have non-zero singular values only for the rank of r, and the rest will be zeros.
7. $A = U_k S_k, B = V_k^T$
   where U_k is the first k columns of U, S_k is the first k rows and columns of S, and V_k^T is the first k rows of V^T.
   The reconstruction $W \approx AB$ is a good approximation especially when singular values beyond the top-k are near zero, as they contribute less to the structure of W.
8. Broad Spectrum of Singular Values: Slow decay of singular values can lead to significant information loss upon truncation.
   Near-Singular Matrices: Near-singular matrices have singular values that are very close to zero. When applying truncated SVD to such matrices, the method may fail to distinguish between relevant and irrelevant singular values, potentially leading to poor approximations.
   Noise Sensitivity: Truncated SVD can be sensitive to noise in the data. If a matrix has noisy data, the singular values corresponding to the noise may be mistaken for significant singular values.

3.3.2 LoRA

1. $r < n$
2. In the back-propagation, since $r << n$, the gradient computation for $n * r$ matrix A and $r * n$ matrix B has a much lower computational complexity compared to on the full $n * n$ matrix W.
3. Before inference, we can explicitly compute and replace W0 with $W = W_0 + AB$, after that it compute the same as the original model.

3.4 Mixed Precision Training

1. Directly using FP16 in every situation would cause a loss of precision, especially when the value of the gradients are very small. By maintaining an FP32 master copy of the weights, the updates are applied to this copy with higher precision. After the update, the FP32 weights can be again turned to FP16 for the following forward and backward passes.
   Although FP16 weight storages are additional, we could save up more space if we store activations in FP16, which always take up a significant portion of the memory. Changing from FP32 to FP16 for activations would directly cut down the memory requirements by half, and this would normally lead to a net decrease in memory usage.