EE - 599 Systems for ML

Project Milestone 1

Team Members:

Likhitha Dayashanker: 1847171486

Harini Thirunavukkarasu: 6114026517

1. **What are the main four major aspects of LLMs covered in [1]?**
    - Pre-training
    - Adaptation tuning
    - Utilization
    - Capacity evaluation

2. **What is the difference between PLM and LLM?**
    - **PLM** -> Pre-trained Language Model; **LLM** -> Large Language Model
    - LLMs are a subset of PLMs. To be precise, the PLMs have a large number of parameters in the order of tens or hundreds of billions of parameters.

3. **What are the main three types of LLM architecture? What are the differences? Name a few examples for each type.**
    - Encoder-Decoder eg. Vanilla Transformer model
    - Casual Decoder eg. GPT-3 model
    - Prefix Decoder eg. U-PaLM and GLM-130B

Differences between them include the following:

The Encoder-Decoder architecture adapts 2 concepts called self-attention to encode the input sequence for generating its latent representations and the other one is cross-attention to autoregressively generate the target sequence.

The causal decoder architecture incorporates the unidirectional attention mask to guarantee that each input token can only attend to the past tokens and itself.

The prefix decoder revises the masking mechanism of the casual decoders to enable performing bidirectional attention over the prefix tokens and unidirectional attention only on generated tokens to bidirectionally decode the data.

4. **What is language modeling? What is the difference between causal language modeling and masked language modeling?**

    - Language modeling is basically a tool for language understanding and generation. It involves predicting the probability distribution of a sequence of words in a given language. Language models learn to assign higher probabilities to more likely word sequences based on the context provided by the preceding words.
    - Casual language modeling is prediction of the next word through a probability distribution curve based on the previous words mentioned.
    - In Masked language modeling, certain words in the sentence are masked and the model is trained to predict these masked words based on the context of unmasked words.

5. **What is a text classification task? What kind of model do we use for this task? Do you need to retrain a classification head for a new downstream task?**

- The goal of a text classification task is to assign labels to a given document or a tweet or a sentence based on its content.
- The models used for this task includes - Logistic regression, Naive Bayes, Support Vector Machines, CNNz, RNNz, Transformers etc.
- For simpler models like SVM, Logistic regression, Naive Bayes, etc we may need to retrain the entire model or a specific parameter for a new task whereas for a transformer based deep learning model we can take an already existing pre-trained model and fine tune only the classification head on your specific task.

6. **What is a summarization task? What kind of model do we use for this task?**

- The goal of a summarization task is to generate a concise and coherent summary of a longer text while retaining its key information and essential meaning.
- Models used: BERT, GPT, Text-to-Text Transfer Transformer, etc.

7. **Why are Adam and AdamW the most widely used optimizers to train LLMs? If training a model with N parameters, we know that the optimizer needs to store N gradients. Does Adam or AdamW introduce extra overhead?**

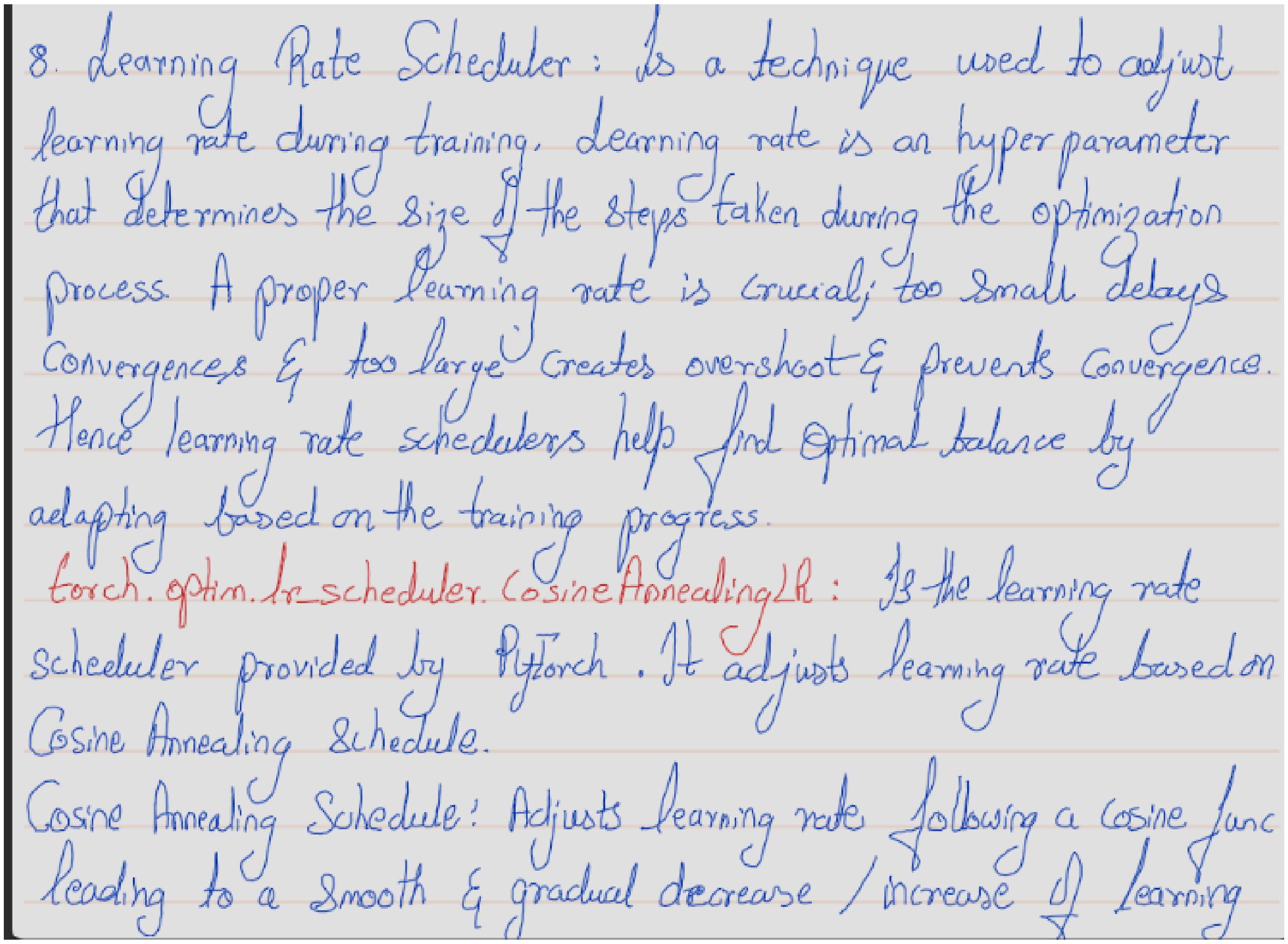
7. Adam (Adaptive Moment Estimation) & AdamW are most popular optimizer for training LLMs due to their adaptive learning rate, momentum & regularization properties.

- Adaptive learning rate: They adjust learning rate for each parameter during training. It computes different learning rates based on historical gradients. to enhance converging rate
- Momentum: They use momentum to accelerate training. & Speed up Convergence

- Bias Correction (In AdamW): AdamW introduces weight decay. Is corrects bias during moving average for stable & reliable training.
- Efficiency: Though Adam & AdamW use additional parameters like momentum & moving average with each parameter leading to Computational overhead yet it is manageable due to Speedups that are achievable.

Yes they do introduce extra overhead

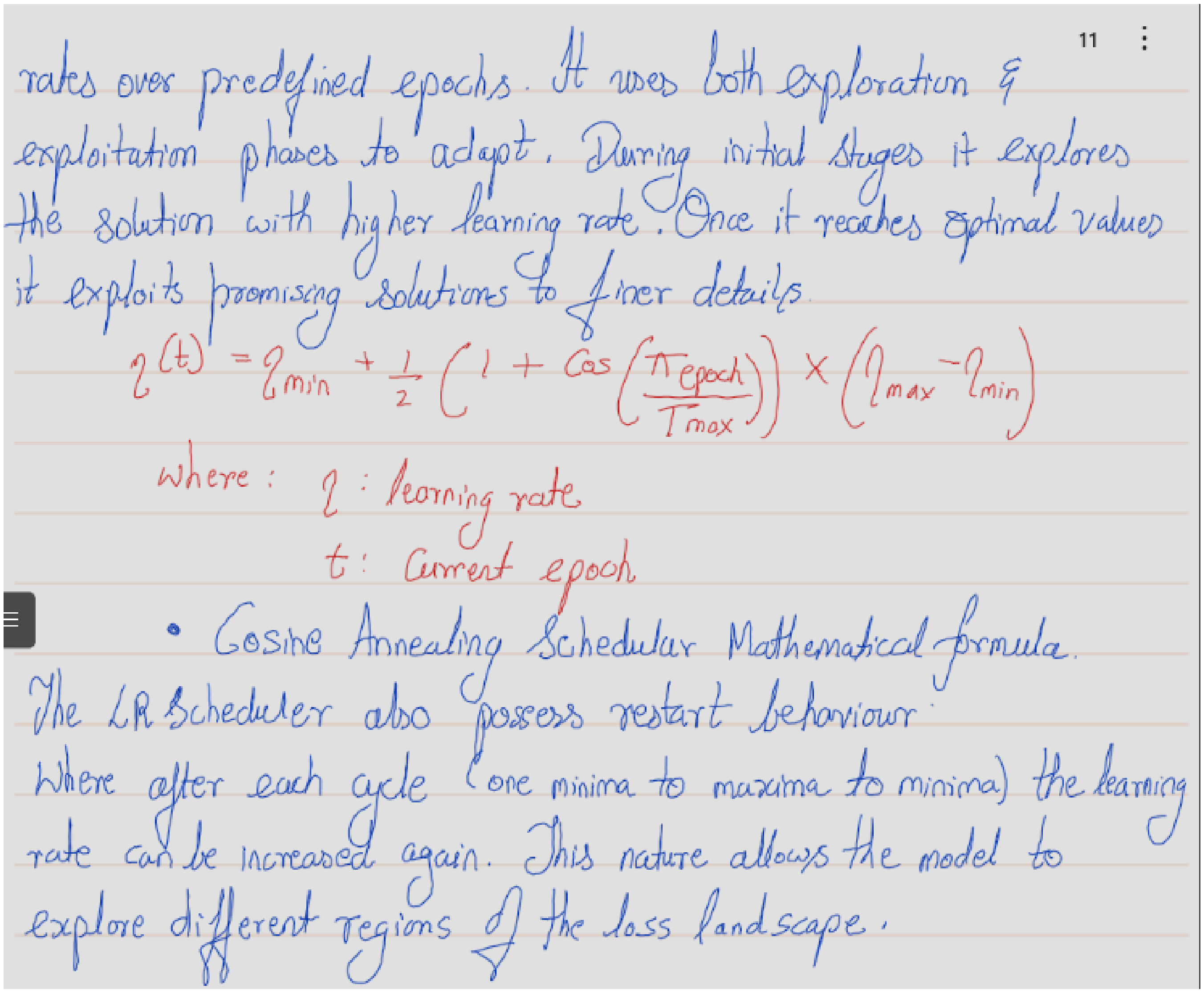
8. **What is a learning rate scheduler? Explain the behavior of torch.optim.lr scheduler.CosineAnnealingLR.**



8. Learning Rate Scheduler : Is a technique used to adjust learning rate during training. Learning rate is an hyper parameter that determines the size of the steps taken during the optimization process. A proper learning rate is crucial; too small delays convergences & too large creates overshoot & prevents convergence. Hence learning rate schedulers help find optimal balance by adapting based on the training progress.

torch.optim.lr_scheduler.CosineAnnealingLR : Is the learning rate scheduler provided by PyTorch. It adjusts learning rate based on Cosine Annealing Schedule.

Cosine Annealing Schedule: Adjusts learning rate following a cosine func leading to a smooth & gradual decrease / increase of learning



rates over predefined epochs. It uses both exploration & exploitation phases to adapt. During initial stages it explores the solution with higher learning rate. Once it reaches optimal values it exploits promising solutions to finer details.

$$\eta(t) = \eta_{min} + \frac{1}{2}\left(1 + \cos\left(\frac{\pi\, epoch}{T_{max}}\right)\right) \times \left(\eta_{max} - \eta_{min}\right)$$

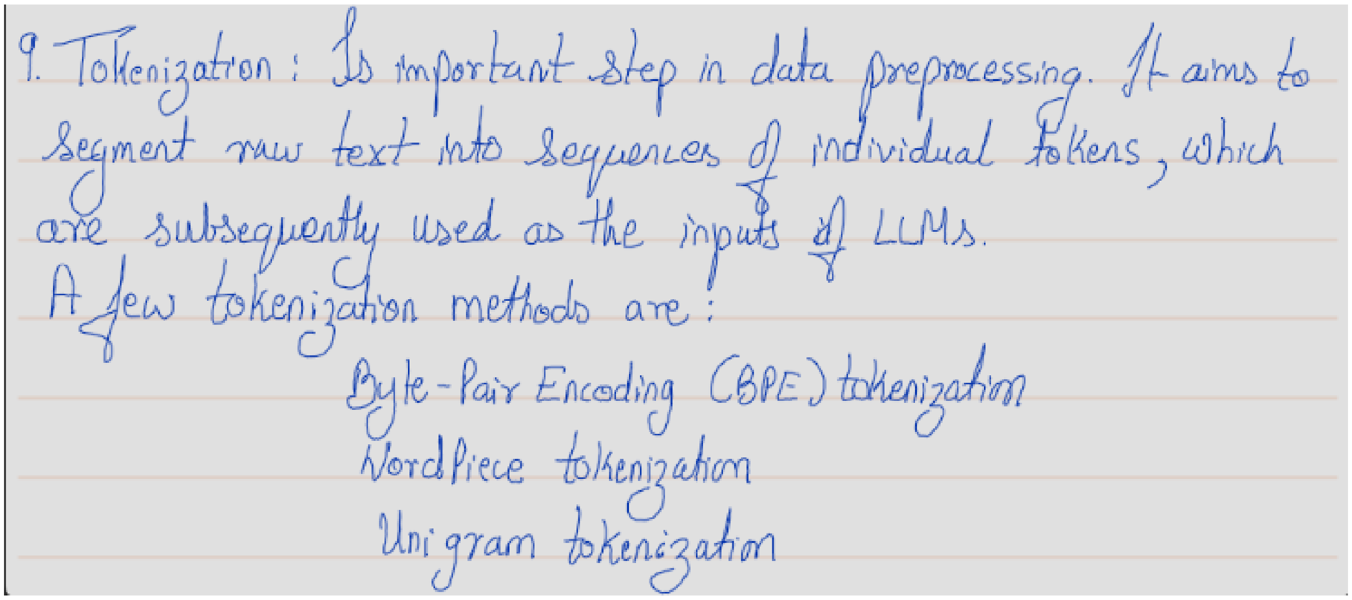
where : η : learning rate

t : Current epoch

- Cosine Annealing Schedular Mathematical formula.

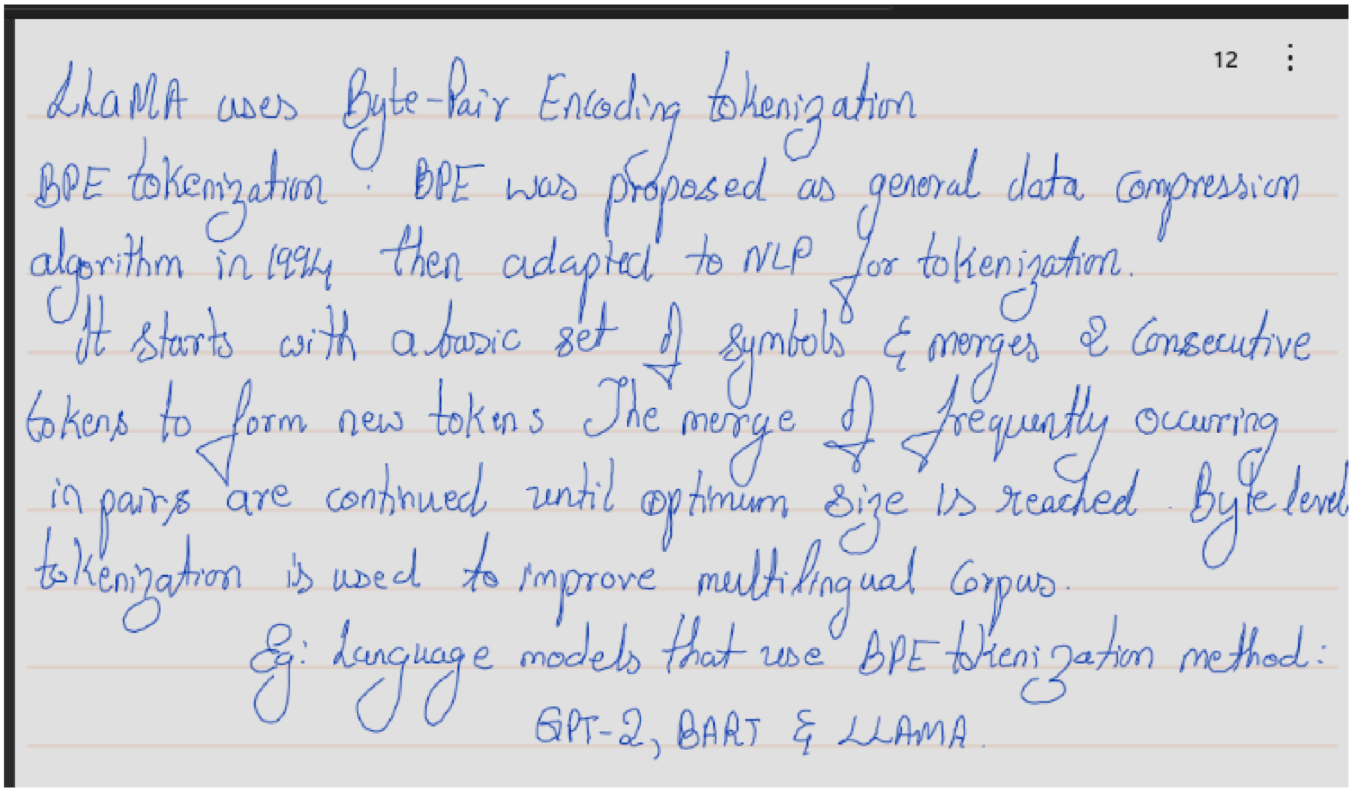
The LR Scheduler also possess restart behaviour.
Where after each cycle (one minima to maxima to minima) the learning rate can be increased again. This nature allows the model to explore different regions of the loss landscape.

9. **What is tokenization? Name a few different tokenization methods. What is the tokenization method used by LLaMA?**



9. Tokenization: Is important step in data preprocessing. It aims to segment raw text into sequences of individual tokens, which are subsequently used as the inputs of LLMs.
A few tokenization methods are:

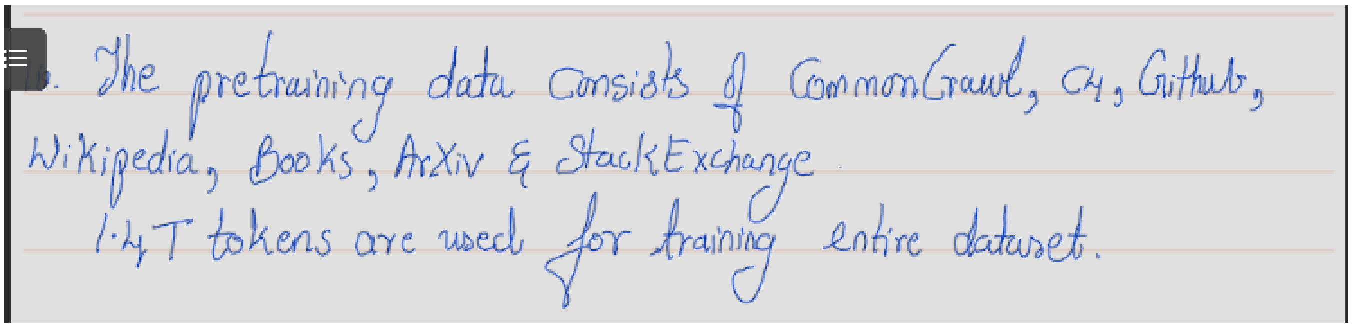
Byte-Pair Encoding (BPE) tokenization
WordPiece tokenization
Unigram tokenization



LLaMA uses Byte-Pair Encoding tokenization
BPE tokenization: BPE was proposed as general data compression algorithm in 1994 then adapted to NLP for tokenization.
It starts with a basic set of symbols & merges & consecutive tokens to form new tokens. The merge of frequently occurring in pairs are continued until optimum size is reached. Byte level tokenization is used to improve multilingual Corpus.

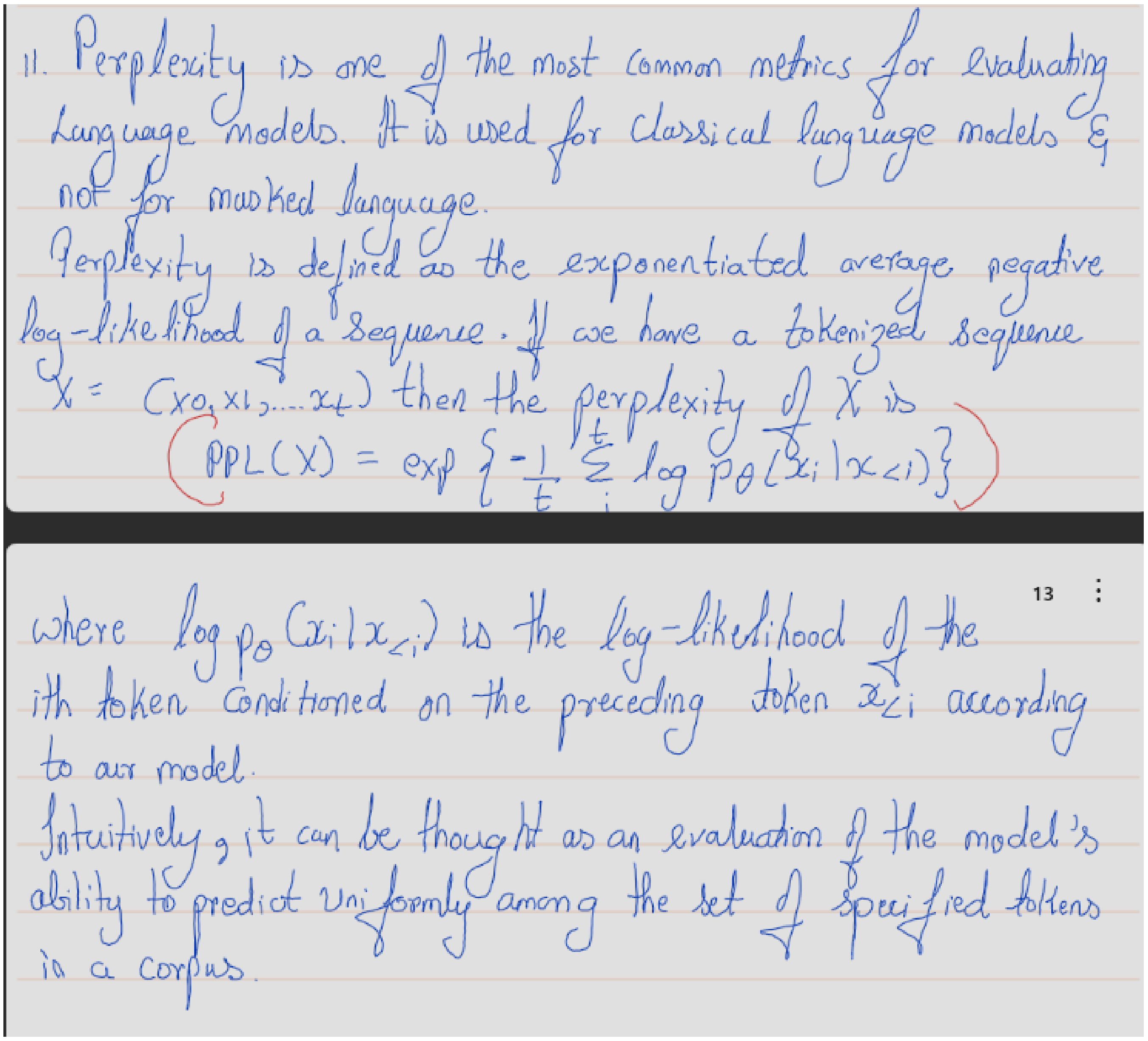
Eg: language models that use BPE tokenization method:
GPT-2, BART & LLAMA.

10. What is the pertaining data used by LLaMA? How many tokens are in the entire training set for training LLaMA?



10. The pretraining data consists of CommonCrawl, C4, Github, Wikipedia, Books, ArXiv & StackExchange.
1.4T tokens are used for training entire dataset.

11. What is perplexity [4]? What is it used for? How to calculate it?



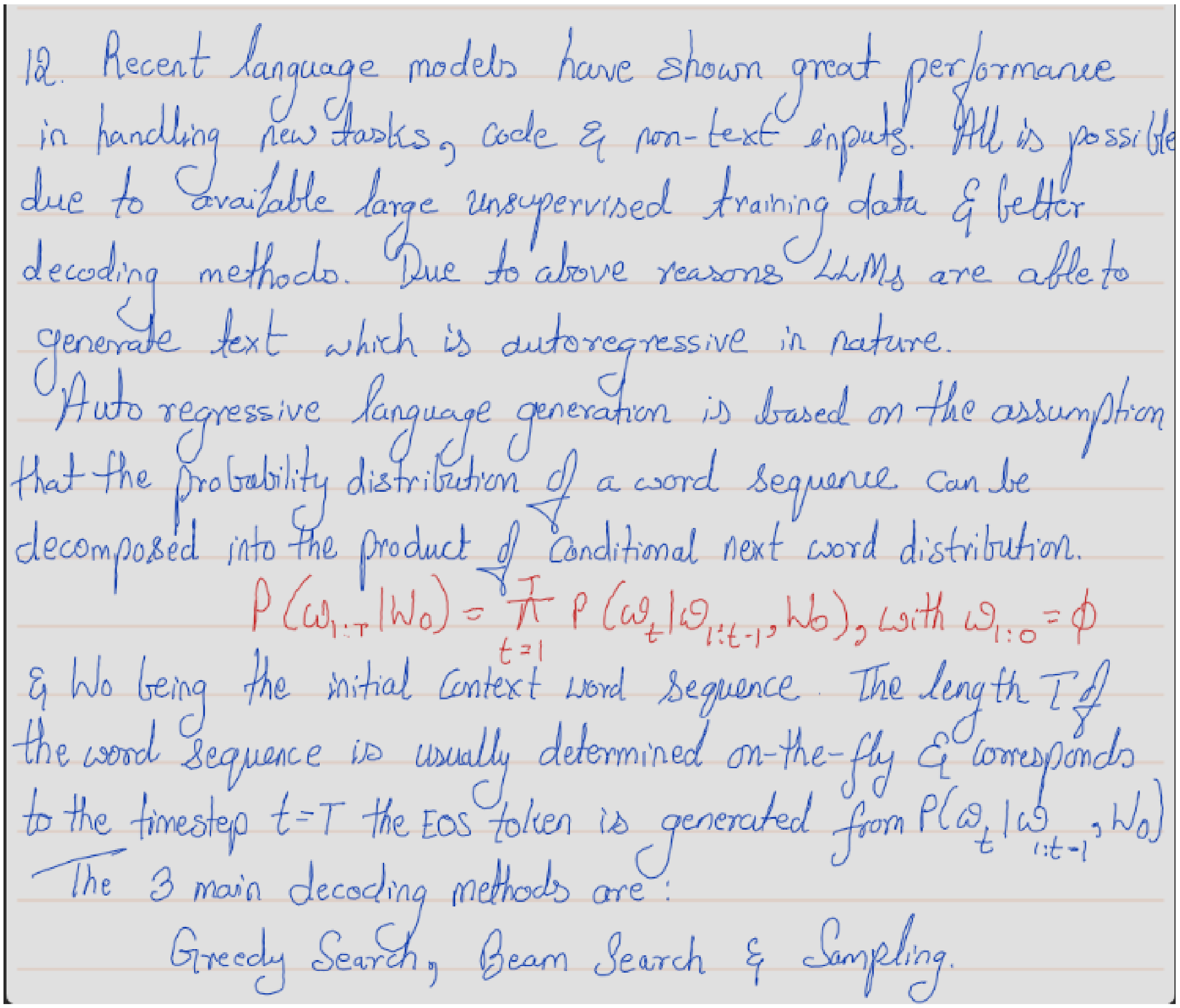
11. Perplexity is one of the most common metrics for evaluating Language models. It is used for classical language models & not for masked language.
Perplexity is defined as the exponentiated average negative log-likelihood of a sequence. If we have a tokenized sequence $X = (x_0, x_1, \ldots, x_t)$ then the perplexity of X is

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

13

where $\log p_\theta(x_i \mid x_{<i})$ is the log-likelihood of the ith token conditioned on the preceding token $x_{<i}$ according to our model.
Intuitively, it can be thought as an evaluation of the model's ability to predict uniformly among the set of specified tokens in a corpus.

12. How to generate text from LLMs [5]? What are the decoding methods for outputs?



12. Recent language models have shown great performance in handling new tasks, code & non-text inputs. All is possible due to available large unsupervised training data & better decoding methods. Due to above reasons LLMs are able to generate text which is autoregressive in nature.

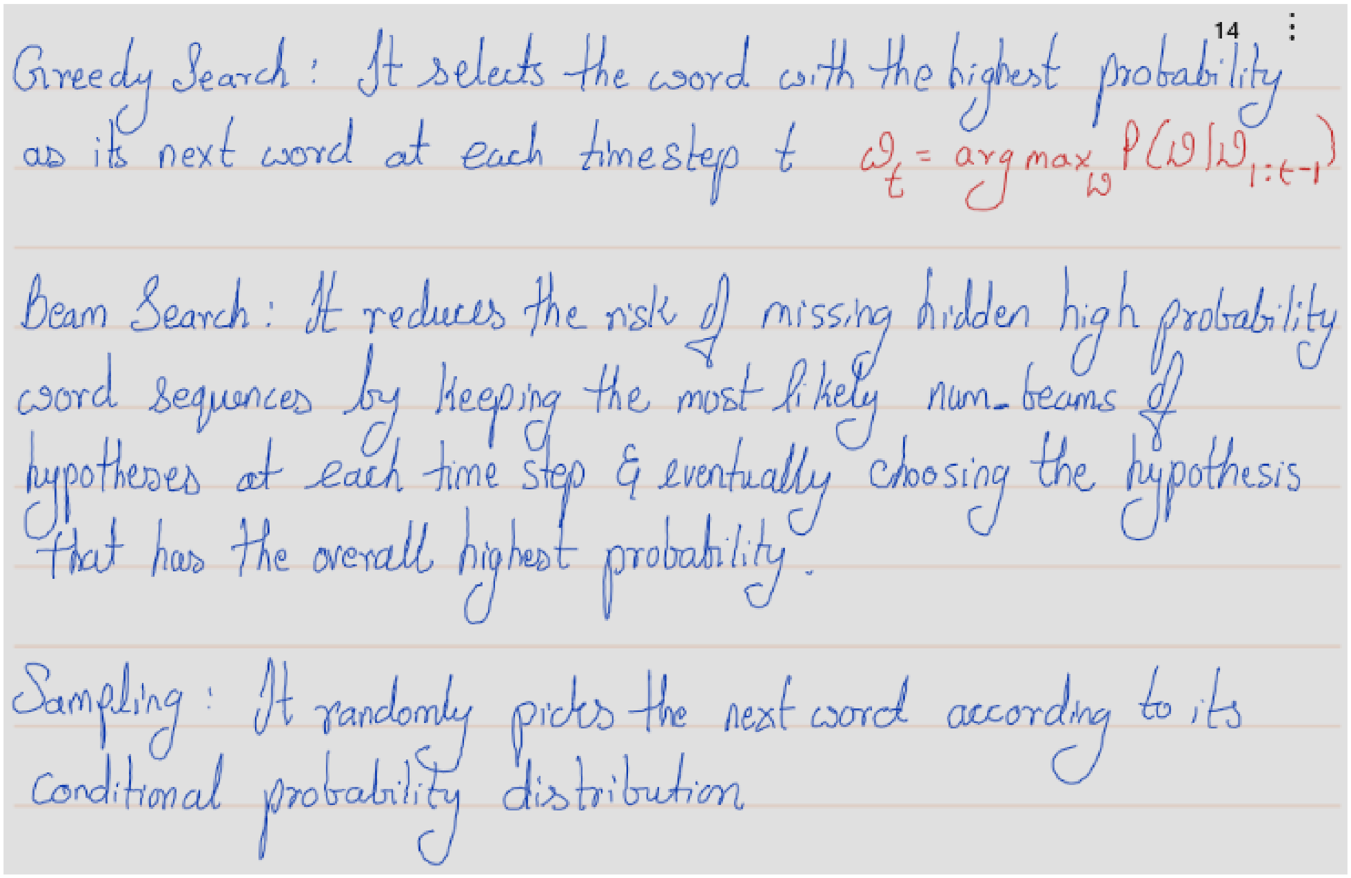
Auto regressive language generation is based on the assumption that the probability distribution of a word sequence can be decomposed into the product of conditional next word distribution.

$$P(w_{1:T}|W_0) = \prod_{t=1}^{T} P(w_t|w_{1:t-1}, W_0), \text{ with } w_{1:0} = \phi$$

& W_0 being the initial context word sequence. The length T of the word sequence is usually determined on-the-fly & corresponds to the timestep $t=T$ the EOS token is generated from $P(w_t|w_{1:t-1}, W_0)$

The 3 main decoding methods are:

Greedy Search, Beam Search & Sampling.



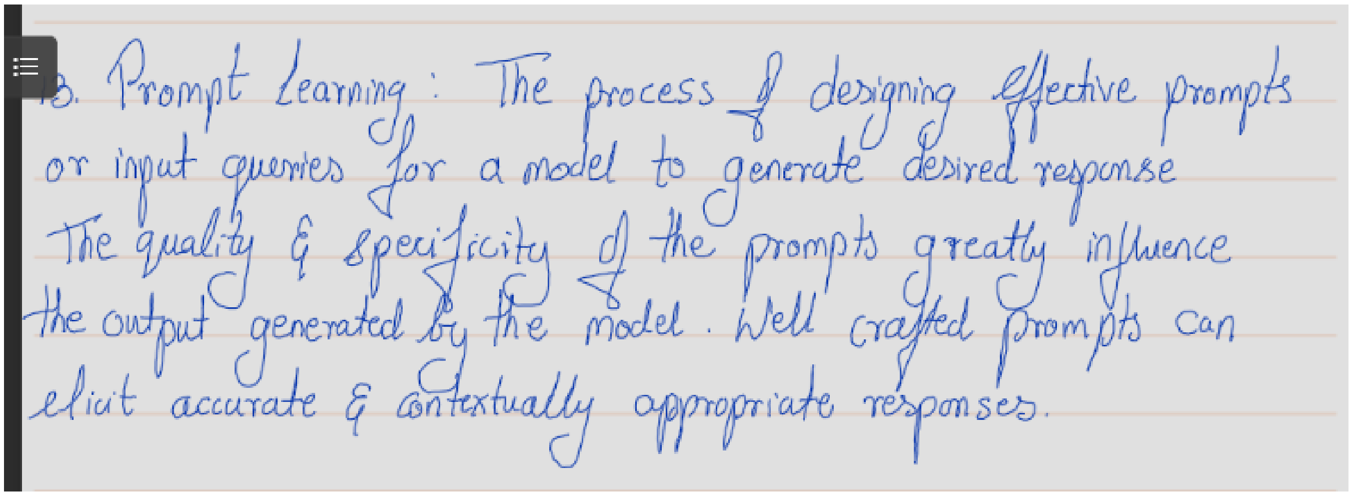
14

Greedy Search: It selects the word with the highest probability as its next word at each timestep t $w_t = \text{argmax}_w P(w|w_{1:t-1})$

Beam Search: It reduces the risk of missing hidden high probability word sequences by keeping the most likely num_beams of hypotheses at each time step & eventually choosing the hypothesis that has the overall highest probability.

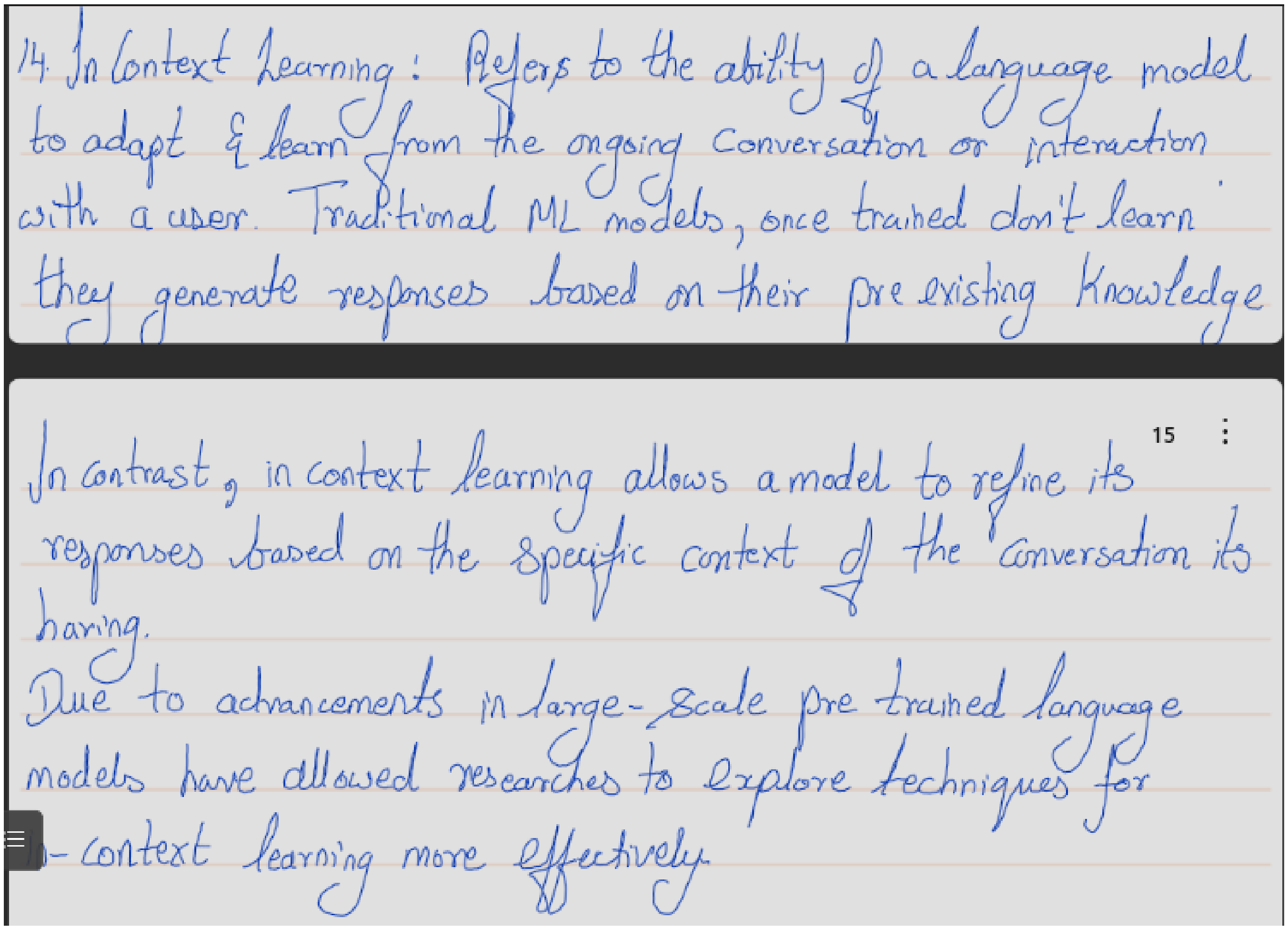
Sampling: It randomly picks the next word according to its conditional probability distribution

13. What is prompt learning?



13. Prompt Learning: The process of designing effective prompts or input queries for a model to generate desired response The quality & specificity of the prompts greatly influence the output generated by the model. Well crafted prompts can elicit accurate & contextually appropriate responses.

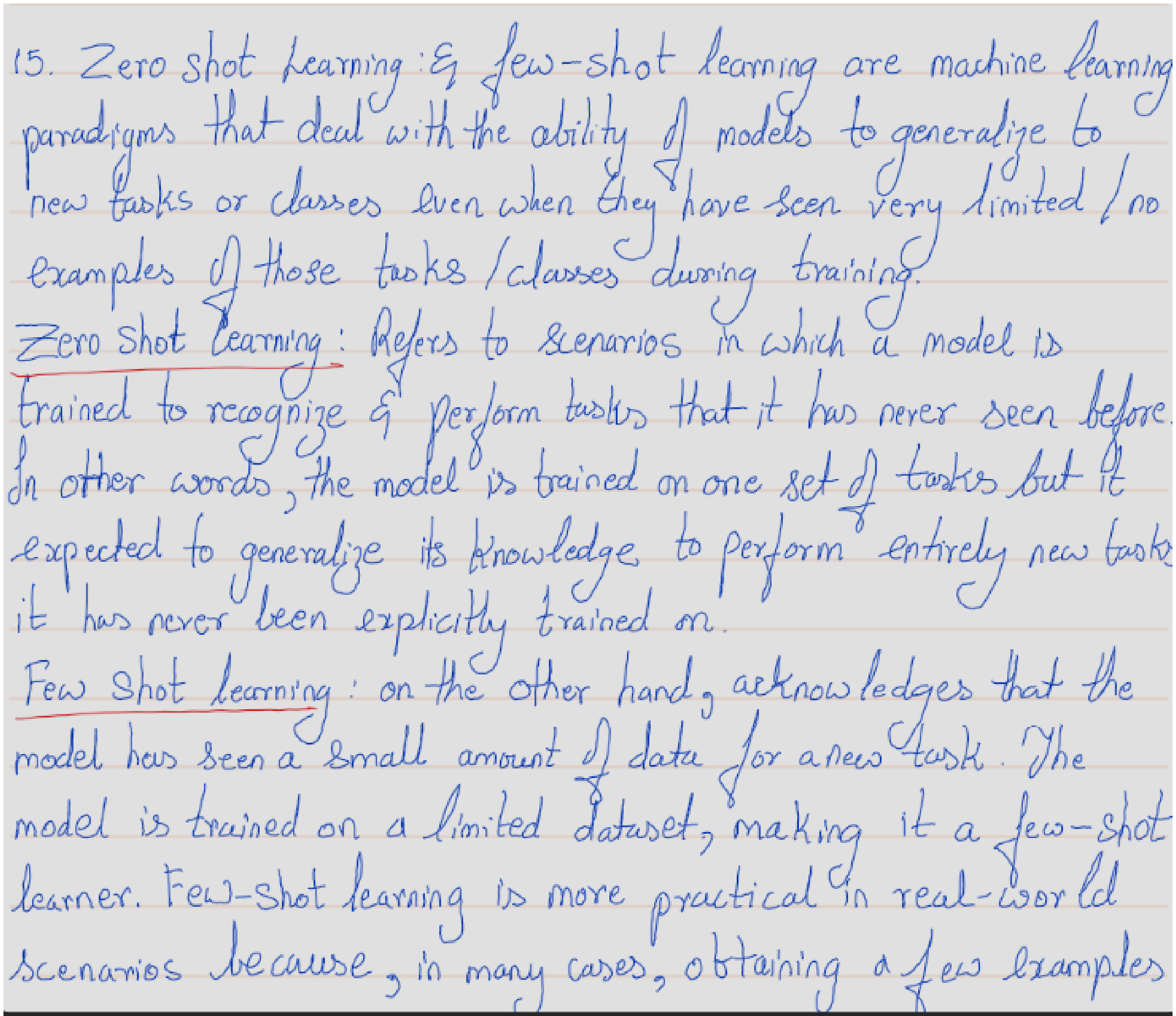
14. What is in-context learning?



14. In Context Learning: Refers to the ability of a language model to adapt & learn from the ongoing Conversation or interaction with a user. Traditional ML models, once trained don't learn they generate responses based on their pre existing Knowledge

In contrast, in context learning allows a model to refine its responses based on the specific context of the conversation its having.
Due to advancements in large-Scale pre trained language models have allowed researches to explore techniques for n- context learning more effectively

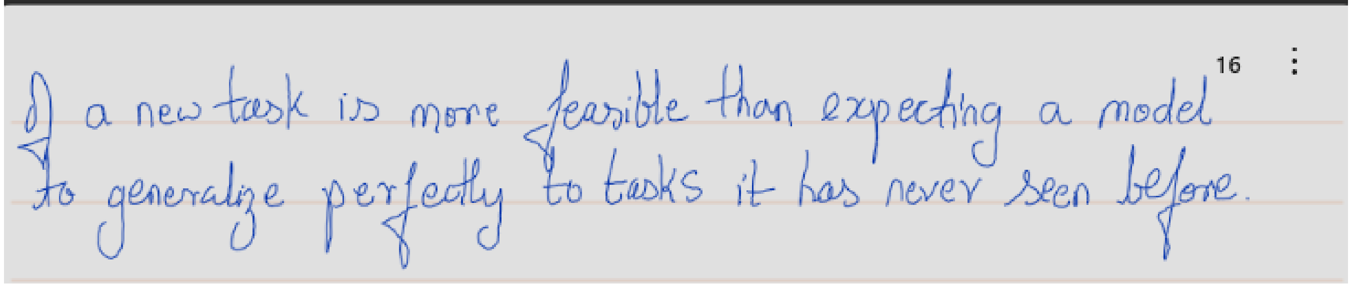
15. What is zero-shot and few-shot learning?



15. Zero shot Learning & few-shot learning are machine learning paradigms that deal with the ability of models to generalize to new tasks or classes even when they have seen very limited / no examples of those tasks / classes during training.

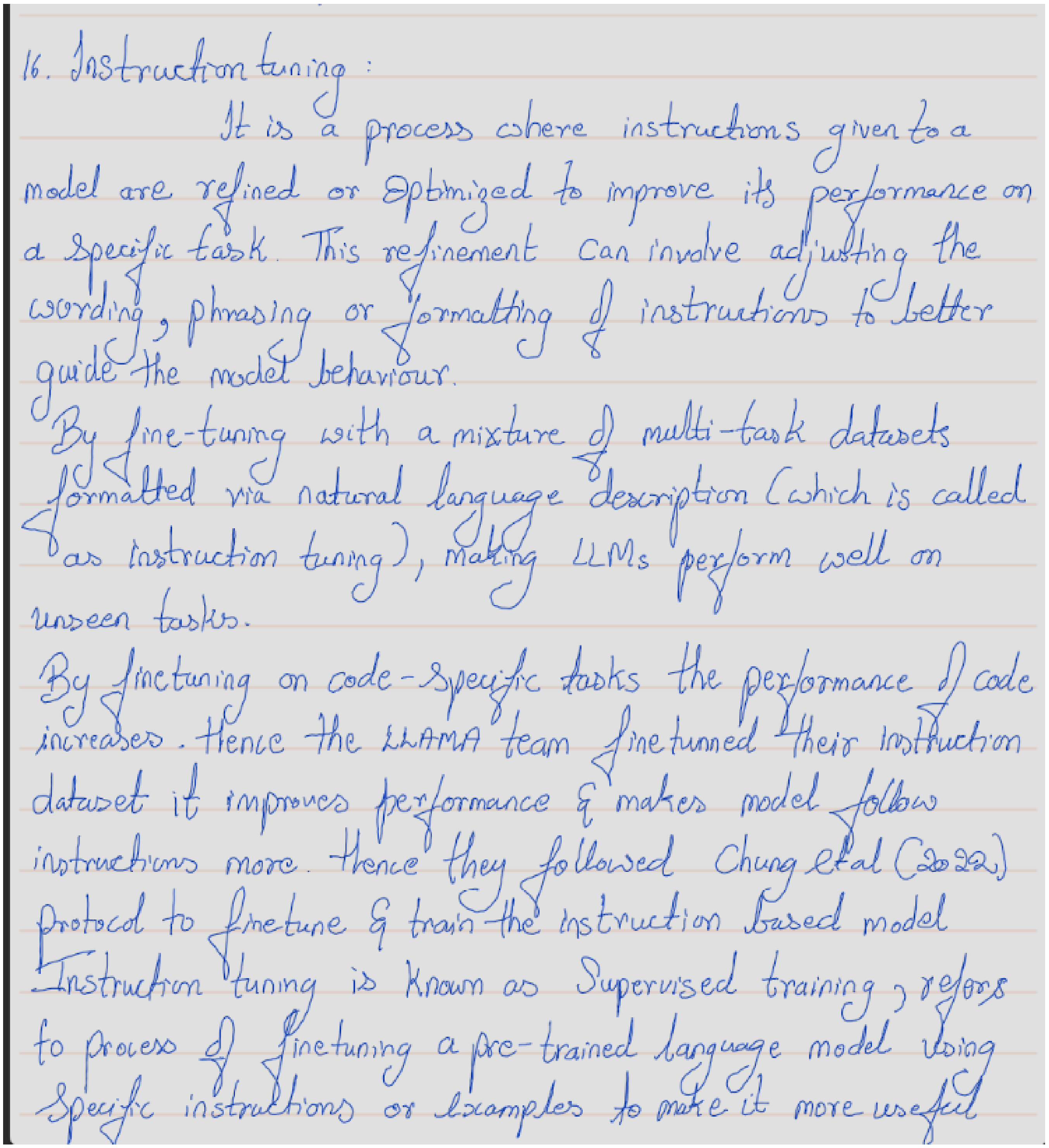
Zero Shot Learning: Refers to scenarios in which a model is trained to recognize & perform tasks that it has never seen before. In other words, the model is trained on one set of tasks but it expected to generalize its knowledge to perform entirely new tasks it has never been explicitly trained on.

Few Shot learning: on the other hand, acknowledges that the model has seen a small amount of data for a new task. The model is trained on a limited dataset, making it a few-shot learner. Few-shot learning is more practical in real-world scenarios because, in many cases, obtaining a few examples



of a new task is more feasible than expecting a model to generalize perfectly to tasks it has never seen before.

16. What is instruction tuning? Particularly, how is it applied to train LLaMA? Why is instruction tuning supervised training?



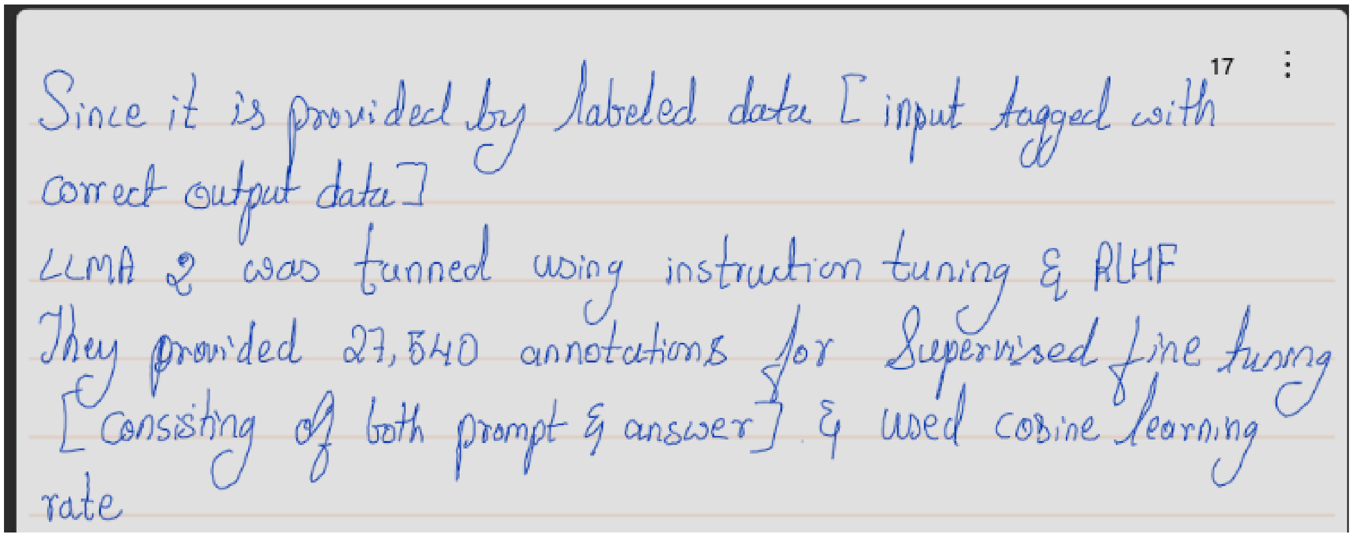
16. Instruction tuning :

It is a process where instructions given to a model are refined or Optimized to improve its performance on a Specific task. This refinement Can involve adjusting the wording, phrasing or formatting of instructions to better guide the model behaviour.

By fine-tuning with a mixture of multi-task datasets formatted via natural language description (which is called as instruction tuning), making LLMs perform well on unseen tasks.

By finetuning on code-Specific tasks the performance of code increases. Hence the LLAMA team finetuned their instruction dataset it improves performance & makes model follow instructions more. Hence they followed Chung etal (2022) Protocol to finetune & train the instruction based model

Instruction tuning is known as Supervised training, refers to process of finetuning a pre-trained language model using Specific instructions or examples to make it more useful



17

Since it is provided by labeled data [input tagged with correct output data]
LLMA 2 was tunned using instruction tuning & RLHF
They provided 27,540 annotations for Supervised fine tuning [Consisting of both prompt & answer] & used cosine learning rate

17. What is Alpaca dataset [6]? Pick a training sample and describe it.

- Alpaca is a dataset of 52,000 instructions which can be used to conduct instruction-tuning for language models and make the language model follow instruction better.

An example of "train" dataset structure is as below:

```
{

    "instruction": "Create a classification task by clustering the given list of items.",

    "input": "Apples, oranges, bananas, strawberries, pineapples",

    "output": "Class 1: Apples, Oranges\nClass 2: Bananas, Strawberries\nClass 3: Pineapples",

    "text": "Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.\n\n### Instruction:\nCreate a classification task by clustering the given list of items.\n\n### Input:\nApples, oranges, bananas, strawberries, pineapples\n\n### Response:\nClass 1: Apples, Oranges\nClass 2: Bananas, Strawberries\nClass 3: Pineapples",

}
```

Explanation:

Instruction - describes the task for the model to execute

Input - Its an optional feature; For eg. For a summarization task, the article could be the input provided by the user

Output - The answer to the task as generated by the model

Text - It is a method used by the user to further fine tune the model (For eg. Changing the font size, font color etc)

18. What is human alignment? Why is it important?

Human alignment refers to the agreement, coordination, or synchronization of artificial intelligence (AI) systems with human values, goals, and preferences. It involves designing AI systems and algorithms in a way that aligns them with human interests, ethical principles, and societal norms. The goal is to ensure that AI technologies operate in a manner that is beneficial, safe, and supportive of human values, rather than working against or diverging from them.

It is important because of the following reasons

- It helps to develop AI systems that adhere to ethical standards.
- It helps to provide the outputs and recommendations that are in line with the user's expectations and preferences.
- It helps to avoid misunderstandings and conflicts in a team where AI is working alongside humans, human intentions should be understood by the AI and provide relevant support.
- It helps to avoid legal issues and ensure responsible AI development

19. Gradient Accumulation:

Let's consider a linear model with Mean Squared Error (MSE) loss. The MSE loss for a single data point can be expressed as:

$$L_i = \tfrac{1}{2}(\hat{y}_i - y_i)^2$$

where hat{y}_i is the predicted value, and yi is the actual target for the i-th data point.

The total loss for a mini-batch of N data points is the average of the individual losses:

$$L_{\text{batch}} = \tfrac{1}{N}\sum_{i=1}^{N} L_i$$

Now, let's say we have a batch size of 8 images, and we accumulate gradients over 4 batches before updating the weights. So, effectively, we are updating the weights based on the accumulated loss:

$$L_{\text{accumulated}} = \tfrac{1}{4}\sum_{j=1}^{4} L_{\text{batch}_j}$$

Here, L_batch_j is the loss for the j-th mini-batch.

Now, let's examine how the gradients are calculated during backpropagation. The gradient of the total loss with respect to a model parameter theta is computed as:

$$\frac{\partial L_{\text{accumulated}}}{\partial \theta} = \frac{1}{4} \sum_{j=1}^{4} \frac{\partial L_{\text{batch}_j}}{\partial \theta}$$

Since the division by 4 is a constant factor, it doesn't affect the direction of the gradient during the weight update. Therefore, the weight update step remains the same whether we update after each mini-batch or accumulate gradients over several mini-batches.

In summary, the division step in gradient accumulation ensures that the accumulated gradient remains equivalent to the non-accumulated gradient on a batch of 32 images, allowing for updates after processing multiple smaller batches.

20. Batch normalization technique:

Batch normalization is a technique used to improve the training of deep neural networks by normalizing the inputs of each layer. It operates by normalizing the inputs to a layer in a mini-batch to have zero mean and unit variance. This normalization is followed by scaling and shifting of the normalized values using learnable parameters, allowing the model to adapt and learn the most suitable scale and shift for each feature.

The mathematical representation of batch normalization for a given layer with input \(x\) is as follows:

$$\text{BN}(x) = \gamma \left(\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} \right) + \beta$$

where:

- μ is the mean of the mini-batch,
- σ^2 is the variance of the mini-batch,
- γ is the scale parameter,
- β is the shift parameter,
- ϵ is a small constant to avoid division by zero.

During training, μ and sigma^2 are calculated for each mini-batch. However, when gradient accumulation is applied, these statistics are not consistently updated for each individual batch. Let's consider the scenario where we accumulate gradients over K mini-batches.

The normalized input BN(x) is a function of μ and sigma^2. Let's denote BN(x) as hat{x} for simplicity:

$$\hat{x} = \gamma \left(\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} \right) + \beta$$

The gradient of the loss with respect to μ and sigma^2 is calculated during backpropagation.

Now, when we accumulate gradients over K mini-batches, the accumulated gradients become:

$$\nabla_{\mu} L_{\text{accumulated}} = \frac{1}{K} \sum_{j=1}^{K} \nabla_{\mu} L_{\text{batch}_j}$$
$$\nabla_{\sigma^2} L_{\text{accumulated}} = \frac{1}{K} \sum_{j=1}^{K} \nabla_{\sigma^2} L_{\text{batch}_j}$$

The inconsistency arises because the statistics μ and sigma^2 used in hat{x} during forward pass were not consistently updated across all mini-batches. As a result, the gradient accumulation may not yield identical results, especially when batch-wise regularization methods like batch normalization are involved. The non-consistent update of batch statistics can impact the training stability and performance of the model.

21. Gradient Checkpointing:

Model inference does not suffer from the memory blow-up problem as the model training because the computations for inference calculations are typically performed layer by layer in a forward pass without the need to store intermediate results for backpropagation as we dont have to perform backpropagation for results.

22. Strategy 1: Retaining all activations:

Memory requirement: O(n)

Retaining all the activations during the forward pass requires storing the activations for each of the n layers, leading to O(n) memory requirement.

Forward computation steps: O(n)

The forward pass requires computing all the activations for each layer sequentially and with each layer requiring 1 unit of computation, the overall forward computation steps to be in the order of n steps.

Strategy2: Optimal placement of checkpoints:

Memory requirement: O(1)

Instead of retaining all the activations, the algorithm strategically selects checkpoints. The memory requirement is constant because the storage is not proportional to the number of layers but is determined by the number of checkpoints.

Forward computation steps: O(n)

The forward path involves computing activations up to each checkpoint. The number of forward computation steps is still O(n) because each layer is visited once and the computation of each layer is O(1).

23. What is a matrix rank?

The rank of a matrix is the maximum number of linearly independent row vectors or column vectors in the matrix.

24. What are three decomposed matrices by SVD?

For a given matrix W of size m x n, the Singular Value Decomposition is given by:

W = UΣV^T

Where U is an m x m orthogonal matrix

Σ is an m x n diagonal matrix with singular values

V^T is an n x n orthogonal matrix

25. U andV are orthogonal matrices. Why does it imply $U^T U$ =I,$V^T V$ =I?

Orthogonal matrices have the property that the transpose of a matrix is equal to the inverse of the matrix. Hence the implication of $U^T U$ =I,$V^T V$ =I is valid where I is the identity matrix.

26. If a matrix W ∈ $R^{n\,x\,n}$is full rank, what is its rank?

If W is a full rank matrix then it means that the rank of W is equal to n.

27. **Suppose a full rank matrix W ∈ $R^{n\,x\,n}$ represents an image. After we apply SVD to this matrix, we modify the singular matrix by only keeping its top-k singular values and discarding the rest (i.e., set the rest of the singular values to zero). Then, we reconstruct the image by multiplying U, modified S, and V. What would the reconstructed image look like? What if you increase the values of k (i.e., keep more singular values)?**

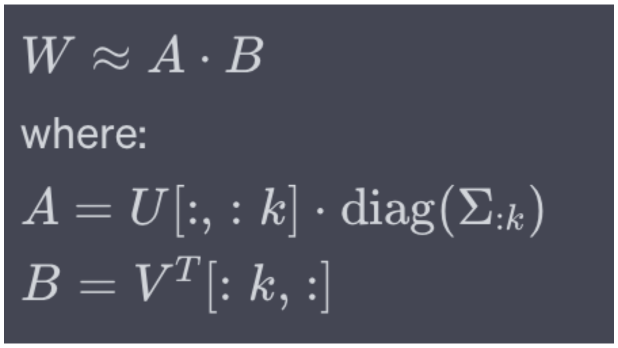
The reconstructed image would be an approximation obtained by using the top-k singular values and their corresponding singular vectors in the SVD decomposition.

28. **If a matrix W ∈ $R^{n\,x\,n}$ is low rank, what does its singular matrix look like?**

If a matrix is low rank then most of its singular values will be close to zero. In the singular value matrix, only the first few singular values will be significant, thereby reducing the rank to 1 or to a less number.

29. **If the top-k singular values of a matrix W ∈ Rn×n are large, and the rest are near zero, this matrix W exhibits low-rank or near-low-rank behavior. Can you represent W by two low-rank matrices, A and B? If so, what are those two matrices' expressions in terms of U, S, and V ? Do you think those two matrices are a good approximation of W (i.e., W ≈ AB)?**

Yes, if W exhibits low-rank behavior, it can be approximated by two low-rank matrices, A and B.



$$W \approx A \cdot B$$

where:

$$A = U[:, :k] \cdot \mathrm{diag}(\Sigma_{:k})$$
$$B = V^T[:k, :]$$

If the discarded singular values are negligible, they can be a good approximation of W whereas if their singular values are non-negligible, then they can't be a good approximation.

30. **The above operation is called truncated SVD. Under what situation do you think truncated SVD fails to make a good approximation? Think about the singular matrix.**

Truncated SVD approximation may fail if the discarded singular values are not negligible. If they are larger, omitting them in the reconstruction can result in a poor approximation of the original matrix.

31. LoRA

To derive the equation in terms of r specifying the conditions under which the total number of parameters in matrices A and B is smaller than that of W0, let's denote the dimensions as follows:

- W0 : n x n
- A : n x r
- B : r x n

The total number of parameters in W0 is n x n. The total number of parameters in A and B are 2nr

For A and B to have fewer parameters than W0, we need:

2nr < n^2

Dividing both sides by n: 2r < n

Dividing both sides by 2: r < n/2

So, the condition under which the total number of parameters in matrices A and B is smaller than that of W0 is r being less than half of n. In the context of the provided information, r typically takes on small values within the range of 4 to 32, ensuring that 2r is significantly smaller than n.

32.

Reduced parameter count: The primary reason for the savings in computation and memory costs is the substantial reduction in the number of trainable parameters. This reduction means that the computational and memory requirements are significantly lower compared to the traditional fine-tuning approach where all parameters are updated.

Matrix multiplication efficiency: The low-rank approximation (A×B) involves fewer computations compared to the full matrix multiplication (W0+ΔW). This is because the rank-*r* approximation allows for more efficient matrix multiplication operations, involving smaller matrices (A and B), reducing the computational load.

Memory Efficiency: With a smaller number of parameters being updated, the memory requirements during fine-tuning are substantially reduced. The memory savings are crucial, especially in scenarios where memory constraints may limit the size of the models that can be fine-tuned.

Faster Convergence: Updating a smaller set of parameters often leads to faster convergence during the fine-tuning stage. This is because the optimization algorithm has to adjust fewer parameters to fit the model to the specific task, resulting in quicker training.

33.

To eliminate this drawback, there are 2 approaches which are highlighted below:

- **Training Stage with LoRA:**During the training stage, employ LoRA to fine-tune the model with the reduced set of parameters (A and B). This allows for the benefits of parameter efficiency and reduced computational and memory costs during the fine-tuning process.
- **Model Compression or Pruning:** After the model has been fine-tuned with LoRA, apply model compression or pruning techniques during a post-processing step. These techniques aim to reduce the size of the model by eliminating unnecessary parameters or weights that contribute minimally to the model's performance.

  Techniques such as weight pruning, quantization, or even more advanced methods like knowledge distillation can be applied to further compress the model. This can be done while preserving the essential knowledge gained during the fine-tuning stage.

34.

In mixed-precision training, the use of both 16-bit (FP16) and 32-bit (FP32) floating-point representations is leveraged to achieve a balance between computational efficiency and model accuracy. The process involves maintaining a single-precision copy of the model weights in FP32, even though computations during the forward and backward passes are performed in lower precision (FP16).

Reasons for an FP32 Master Copy of Weights:

Numerical Precision for Weight Updates:
- While most of the computations are performed in lower precision (FP16) to save memory and computation, maintaining an FP32 master copy of weights is crucial for preserving numerical precision during weight updates.
- Weight updates involve small increments or decrements to the weights based on the calculated gradients. In FP16, the limited precision can result in numerical instability, especially for small values. By updating the master copy in FP32, the precision is higher, mitigating potential issues related to accumulated rounding errors.

Accumulation of Gradients:
- During the backward pass, gradients are calculated in FP16. As gradients are accumulated over multiple mini-batches or time steps, the limited precision of FP16 can lead to a loss of information (vanishing gradients) or numerical instability (overflow or underflow).
- Maintaining an FP32 master copy for accumulating gradients helps mitigate these issues, ensuring more stable and accurate gradient updates over the course of training.

Memory Requirement Reduction: Despite storing an additional copy of weights in FP16, the memory requirement is reduced due to the following factors:

Activation Memory Usage:

- Activations during the forward pass are stored in lower precision (FP16), resulting in reduced memory usage compared to storing them in FP32.

Parameter Memory Usage:

- The primary memory-consuming aspect is the model parameters (weights). Storing a single copy in FP32, even if larger in memory, is offset by the reduced memory footprint of activations.

Reduced Communication Costs:

- In distributed training scenarios, where models are trained across multiple devices or nodes, the reduced memory size of activations and gradients (in FP16) can lead to lower communication costs between devices during the training process.

In summary, maintaining an FP32 master copy of weights ensures numerical stability during weight updates, while the memory requirement is still reduced overall due to the lower precision used for activations and gradients. The use of mixed precision is a trade-off between computational efficiency and maintaining numerical stability throughout the training process.