3 Preliminary

3.1 Gradient Accumulation

Q: Consider a linear model with MSE loss, and mathematically explain why this division step can yield identical results.

A:
Assume n is number for images in a batch, the MSE can be denoted as:

$$MSE_{batch} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

If Gradient Accumulation is applied during training, where 'm' represents the number of sections it divides into, 'q' signifies the number of images in one section, and 'n' is the total number of images, the MSE can be expressed as:

$$\begin{aligned} MSE_{GA} &= \frac{1}{m}\sum_{j=1}^{m}\frac{1}{q}\sum_{i=(j-1)*q}^{(j-1)*q+q}\left(Y_i - \hat{Y}_i\right)^2 \\ &= \frac{1}{mq}\sum_{i=1}^{mq}\left(Y_i - \hat{Y}_i\right)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 = MSE_{batch} \end{aligned}$$

Q: Mathematically explain the batch normalization layer behavior and why gradient accumulation yields non-identical results in this case.

A:
Assume n is number for images in a batch, the Batch Mean and Batch Variance can be denoted as:

$$\mu_{batch} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma^2_{batch} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_{batch})^2$$

$$x_{BN} = \frac{x - \mu_{batch}}{\sigma_{batch}}$$

If Gradient Accumulation is applied during training, where 'm' represents the number of sections it divides into, 'q' signifies the number of images in one section, and 'n' is the total number of images, the Batch Mean and Batch Variance for each section can be denoted as:

$$\mu_{GA_m} = \frac{1}{q}\sum_{i=1}^{q} x_i$$

$$\sigma_{GA_m}^2 = \frac{1}{q}\sum_{i=1}^{q}\left(x_i - \mu_{GA_m}\right)^2$$

$$x_{BN_m} = \frac{x_{GA_m} - \mu_{GA_m}}{\sigma_{GA_m}}$$

Since:

$$\mu_{batch} \neq \mu_{GA_m}$$
$$\sigma_{batch}^2 \neq \sigma_{GA_m}^2$$

Therefore, the batch normalization layer can lead to non-identical results when gradient accumulation is employed during training.

3.2 Gradient Checkpointing

Q: Explain why model inference does not have such "memory blow-up" problem.

A:
During inference, only a single forward pass occurs, and there is no backpropagation. Therefore, inference doesn't involve multiple forward and backward passes, and the model doesn't need to calculate gradients, update parameters, or store all the activation values and gradients. As a result, inference requires significantly less memory compared to model training.

Q: What are the memory requirement and forward computation steps for the two strategies in big O notation?

A:

	Default	Memory-Poor (1st strategy)	$\sqrt{n}$ (2nd strategy)
Memory Requirement	$O(n)$	$O(1)$	$O(\sqrt{n})$
Computation Requirement	$O(n)$	$O(n^2)$	$O(n)$
Forward calcs per node	1	1 to n	1 to 2

3.3 Low-Rank Adaptation (LoRA)

Q1: What is matrix rank?

A:
The rank of a matrix is the maximum number of linearly independent rows or columns it contains, indicating the count of unique rows or columns.

Q2: What are three decomposed matrices by SVD?

A:

$$A = U\Sigma V^T$$

U: Left singular vector, it can be used to transform data into a new basis

Σ: Diagonal singular values, this is a diagonal matrix that contain the singular value of the original matrix
V^T: Right singular vector, it can be used to transform data or perform dimensionality reduction

---

Q3: U and V are orthogonal matrices. Why does it imply $UU^T = I$, $VV^T = I$?

A:
An orthogonal matrix is defined such that when it is multiplied by its transpose, the result is the identity matrix. Therefore, if U and V are orthogonal matrices, it implies that that $UU^T = I$ and $VV^T = I$.

Q4: If a matrix $W \in R_{n*n}$ is full rank, what is its rank?

A:
If a matrix $W \in R_{n*n}$ is full rank, it implies that both its row rank and column rank are equal to n.

Q5: Suppose a full rank matrix $W \in R_{n*n}$ 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)?

A:
When retaining only the top-k singular values to reconstruct the image, you will notice that the image loses some of its finer details, leading to a decrease in image quality compared to the original. As you increase the value of k, the reconstructed image will progressively resemble the original image more closely.

Q6: If a matrix $W \in R_{n*n}$ is low rank, what does its singular matrix look like?

A:
If W is not full rank, the singular value matrix will not be a perfect diagonal identity matrix. Instead, it will have only a limited number of non-zero singular values, while the remaining singular values will be very close to or equal to zero.

Q7: If the top-k singular values of a matrix $W \in R_{n*n}$ 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., $W \approx AB$)?

A:
If top-k singular values of a matrix $W \in R_{n*n}$ are large, and the rest are near zero W can be represent by two low rank matrices A and B.

$$A = U[:, 1:k]$$

$$B = \Sigma[1:k,:] * V^T$$

Where k is the first k columns of the matrix Σ
The approximation of W is giving by $W \approx AB$. The quality of this approximation depends on the choice of k. If the top-k singular values are indeed significantly larger than the rest, then this approximation is often quite good.

Q8: 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.

A:
If the singular values exhibit a relatively uniform distribution, implying that there is no substantial decrease in magnitude beyond the initial top few singular values, truncating to a small k might not effectively capture the majority of the information contained in the original matrix. In such situations, the approximation is likely to be less accurate.

Q: Derive the equation in terms of r that specifies the conditions under which the total number of parameters in matrices A and B is smaller than that of W0.

A:
We know: $W_0 \in R^{n\times n}, A \in R^{n\times r}, B \in R^{r\times n}$
So, the total number of parameters in $W_0 = n^2$
The total number for parameters in A and B is $n \times r + r \times n = 2rn$, where $r \ll n$ so $2rn \ll n^2$
Therefore, the total number of parameters in matrices A and B is smaller than that of W_0

Q: Consider carefully why LoRA can save computation and memory costs during the fine-tuning stage.

A:
During the fine-tuning's backward propagation, we keep the pre-trained weights frozen and only update the weights of matrices A and B. Since the parameters of A and B are significantly fewer than those of the original model, both the computation and memory costs are much lower compared to directly updating the original model.

Q: Can you think of a way to eliminate this drawback?

A:
To address the drawbacks of adding matrices A and B, several approaches can be considered. Firstly, we can apply Mixed Precision Training techniques to the A and B matrices as well as the original parameters. This can reduce space and time costs while ensuring precision. Secondly, the weights of matrices A and B can be merged with the pre-trained matrix during fine-tuning. This would eliminate the A and B matrices, thus maintaining the same forward pass time as before.

3.4 Mixed Precision Training

Q: Check the paper and answer why we need an FP32 master copy of weights and why the memory requirement is reduced despite storing an additional copy of weights in FP16.

A:
Compared to FP16, FP32 requires more memory but offers higher precision. Retaining an FP32 master copy of weights maximizes the preservation of the model's precision. This is because during gradient accumulation, if the gradients are very small, FP16 might choose a value larger or smaller than the actual value. Therefore, choosing FP32 helps to minimize errors caused by loss of information.

The reason why the memory requirement is reduced despite storing an additional copy of weights in FP16 is that the memory space required for FP16 is only half that of FP32. This significantly reduces the total computational demand during forward and backward propagation.