QLoRA (Quantized Low-Rank Adaptation)

QLoRA (Quantized Low-Rank Adaptation)

Introduction

QLoRA is a combination of quantization and LORA techniques that builds on the principles of LoRA while introducing 4-bit NormalFloat (NF4) quantization and Double Quantization techniques.

LoRA (Low-Rank Adaptation)

LoRA is a technique used for fine-tuning pre-trained language models for downstream tasks. It involves adding adapter layers to the original model, which are initialized with weights that are decomposed based on the rank of the matrix. The adapter layers are added above self-attention and feed-forward networks at two places.

Quantization

Quantization is a technique used to reduce the precision of neural network weights from 32-bit floating-point numbers to lower-bit values. In QLoRA, 4-bit NormalFloat (NF4) quantization is used, which exploits the fact that weights in pre-trained models typically follow a zero-centered normal distribution. By aligning this natural distribution to a predefined 4-bit scale (within the range -1 to 1), NF4 avoids computationally intensive statistical calculations.

Double Quantization

Double Quantization is a technique used to reduce the memory cost of storing quantization parameters (e.g., scaling factors). It applies a second quantization step to these parameters, compressing them from 32-bit to 8-bit values. This nested approach minimizes memory overhead while preserving model performance.

Low-Rank Adaptation with Quantization

In LoRA, the original model is modified by adding adapter layers, which are initialized with weights that are decomposed based on the rank of the matrix. The adapter layers are split into two parts, A and B, which are initialized with weights and are decomposed based on the rank of the matrix. For example, if the rank is chosen as 1, 5, and 1, 5, then A and B will be 1x5 and 5x1 matrices, respectively.

The weights of the adapter layers are quantized using NF4 quantization, which reduces the precision of the weights from 32-bit floating-point numbers to 4-bit values. The quantized weights are stored in memory, and during inference, the original weights and the fine-tuned weights are combined to obtain the final weights.

Advantages

QLoRA offers several advantages, including:

- **Parameter-efficient fine-tuning**: QLoRA requires fewer parameters to be fine-tuned, which reduces the memory overhead and computational cost.
- **Improved performance**: QLoRA preserves the performance of the original model while reducing the precision of the weights.
- **Efficient inference**: QLoRA allows for efficient inference by combining the original weights and the fine-tuned weights on the fly.

Conclusion

QLoRA is a powerful technique that combines the benefits of LoRA and quantization to enable efficient fine-tuning of pre-trained language models for downstream tasks. By reducing the precision of the weights and using Double Quantization, QLoRA minimizes memory overhead while preserving model performance.

Fine-Tuning

Fine-Tuning

Introduction

Fine-tuning is a crucial concept in Large Language Models (LLMs). It involves adjusting the pre-trained model's parameters to better align with the desired behavior for a specific task or dataset. In this lecture, we will explore the concept of fine-tuning, its importance, and various methods used to implement it.

What is Fine-Tuning?

Fine-tuning is the process of updating the pre-trained model's parameters to fit a new dataset or task. It involves adjusting the model's weights to better align with the desired behavior. Fine-tuning is essential in LLMs as it allows the model to adapt to new data and improve its performance.

Full Fine-Tuning vs. Parameter Efficient Fine-Tuning (PEFT)

Full fine-tuning involves updating all 176 billion parameters of the pre-trained model. This process is computationally expensive and requires significant memory and resources. On the other hand, PEFT involves updating only a portion of the parameters, making it more efficient and feasible.

Methods of Fine-Tuning

There are several methods of fine-tuning, including:

- **LoRa**: LoRa is a method of PEFT that involves updating a portion of the parameters while freezing the rest. It is a popular method of fine-tuning due to its efficiency and effectiveness.
- **QLoRa**: QLoRa is another method of PEFT that involves updating a portion of the parameters using a quantization technique.
- **Prefix Tuning**: Prefix tuning involves updating the prefix of the input sequence to adapt to the new dataset or task.

General Steps in Implementing Fine-Tuning

The general steps in implementing fine-tuning are:

1. **Prepare the Dataset**: Prepare the dataset for fine-tuning by preprocessing and formatting the data.
2. **Select a Pretrained Model**: Select a pre-trained model that is suitable for the task or dataset.
3. **Format and Tokenize the Data**: Format and tokenize the data to convert it into a format that the model understands.
4. **Configure the Training Process**: Configure the training process by setting up the training parameters such as batch size, number of epochs, etc.
5. **Train the Model**: Train the model by updating the parameters to better align with the desired behavior.

Importance of Fine-Tuning

Fine-tuning is essential in LLMs as it allows the model to adapt to new data and improve its performance. It is particularly important when working with large datasets or tasks that require specialized knowledge. Fine-tuning enables the model to learn from the new data and adjust its parameters to better align with the desired behavior.

Challenges of Fine-Tuning

One of the significant challenges of fine-tuning is the computational resources required. Full fine-tuning requires significant memory and resources, making it computationally expensive. PEFT methods such as LoRa, QLoRa, and prefix tuning provide a more efficient alternative, but they still require significant resources.

In conclusion, fine-tuning is a crucial concept in LLMs that involves adjusting the pre-trained model's parameters to better align with the desired behavior. There are various methods of fine-tuning, including full fine-tuning and PEFT methods such as LoRa, QLoRa, and prefix tuning. Understanding the importance and challenges of fine-tuning is essential for effective implementation in LLMs.

LoRA(Low-Rank Adaptation)

LoRA (Low-Rank Adaptation)

Overview

LoRA is a fine-tuning approach that builds upon the observation that the weights learned by Large Language Models (LLMs) after training often contain redundancies. Instead of fine-tuning the entire set of weights in the LLM, LoRA focuses on a low-rank approximation of the weights, which is a smaller set of weights that eliminates these redundancies.

Key Concept

The core idea behind LoRA is to inject low-rank matrices as two adapter layers, one above self-attention and one above feed-forward network, in every layer of the transformer model. This approach allows for efficient fine-tuning while reducing the risk of overfitting and catastrophic forgetting.

Mathematical Formulation

The LoRA-based fine-tuning process can be succinctly captured by the following equation:

$Wo + \Delta W = Wo + BA$

where:

- Wo denotes the pre-trained parameter weights
- ΔW denotes the learned weights to be used in adjusting the original weights
- W is the final fine-tuned weight that will be used during inference
- B is a matrix of dimension $d \times r$ and A is a matrix of dimension $r \times k$
- The rank $r << \min(d, k)$

Low-Rank Approximation

The approach is to fine-tune the matrix decomposition of ΔW, i.e., matrices B and A, with a rank r significantly less than the min(d, k) from the original matrix. This reduces the number of parameters we need to fine-tune.

Advantages

Fine-tuning the matrices A and B gives the following advantages:

- It takes significantly less memory and storage fine-tuning since the dim r << min(d,k)
- Reduces the risk of catastrophic forgetting, as the rest of the model parameters remain untouched

Example

Original weight matrix: $W \in R^{(500 \times 400)}$ Total parameters = 500 × 400 = 200,000

LoRA decomposition with rank r = 4: $A \in R^{(500 \times 4)}$, $B \in R^{(4 \times 400)}$

Total trainable parameters: (500 × 4) + (4 × 400) = 2,000 + 1,600 = 3,600

This is a significant reduction in the number of trainable parameters, which also mitigates the catastrophic forgetting problem.

Key Takeaways

- LoRA injects low-rank matrices as adapter layers in the transformer model
- It reduces the number of parameters to be fine-tuned, making the process more efficient
- It reduces the risk of overfitting and catastrophic forgetting
- It is a form of PEFT (Prompt Engineering for Fine-Tuning)

References

Fine Tuning of Large Language Models (LLMs)

What is Fine Tuning?

Fine tuning is a method of adjusting pre-trained models to fit specific tasks or datasets. It involves updating a subset of the model's parameters to adapt to the new task, while keeping the majority of the parameters frozen.

Parameter Efficient Fine Tuning (PEFT)

PEFT is a type of fine tuning that updates only a small subset of the model's parameters, rather than the entire model. This approach is particularly useful for large language models (LLMs) with billions of parameters.

Methods of Fine Tuning

There are several methods of fine tuning, but in this course, we will focus on three specific methods:

1. **LoRa (Low Rank Approximation)**
2. **QLoRa (Quantized LoRa)**
3. **Prefix Tuning**

LoRa (Low Rank Approximation)

LoRa is a method of fine tuning that updates only a small subset of the model's parameters. It is a type of PEFT that freezes the majority of the model's parameters and trains a new set of parameters altogether. LoRa is particularly useful for large language models with billions of parameters.

Key Points about LoRa

- LoRa is a type of PEFT
- It freezes the majority of the model's parameters (e.g., 176 billion)
- It trains a new set of parameters altogether
- It is useful for large language models with billions of parameters

Catastrophic Forgetting

Catastrophic forgetting refers to the phenomenon where a model forgets its previous knowledge when fine-tuned on a new task. This can be a problem in fine tuning, especially when the new task is significantly different from the original task.

Quantization in LLMs

Quantization is a technique used to reduce the precision of the model's parameters, making them more efficient to store and compute. In the context of LLMs, quantization can be used to reduce the memory requirements of the model.

QLoRa (Quantized LoRa)

QLoRa is a method of fine tuning that combines LoRa with quantization. It updates only a small subset of the model's parameters, while quantizing the remaining parameters to reduce memory requirements.

Prefix Tuning

Prefix tuning is another method of fine tuning that involves updating a small subset of the model's parameters. It is similar to LoRa, but with a different approach to updating the parameters.

Key Takeaways

- Fine tuning is a method of adjusting pre-trained models to fit specific tasks or datasets
- PEFT is a type of fine tuning that updates only a small subset of the model's parameters
- LoRa, QLoRa, and prefix tuning are three methods of fine tuning that will be covered in this course
- Catastrophic forgetting is a phenomenon that can occur during fine tuning
- Quantization is a technique used to reduce the precision of the model's parameters

---

What is Fine-Tuning?

What is Fine-Tuning?

Definition and Purpose

Fine-tuning in deep learning is a form of transfer learning that involves taking a pre-trained model, which has been trained on a large dataset for a general task such as image recognition or natural language understanding, and making minor adjustments to its internal parameters. The goal is to optimize the model's performance on a new, related task without starting the training process from scratch. Fine-tuning lets you turn a generalist into a specialist, making the model more accurate, aligned, and efficient for the specific use case.

Full Fine-Tuning vs. Parameter Efficient Fine-Tuning (PEFT)

Full fine-tuning involves changing all the parameters of a pre-trained model, which can be computationally expensive and may not be feasible due to infrastructural limitations. On the other hand, PEFT involves updating only a portion of the parameters, freezing the rest, which is a more efficient and practical approach.

Methods of Fine-Tuning

There are several methods of fine-tuning, but in this course, we will focus on three methods under PEFT: LoRa, QLoRa, and prefix tuning.

LoRa (Low-Rank Adaptation)

LoRa is a method of PEFT that involves updating a low-rank approximation of the weight matrix instead of the entire matrix. This approach reduces the number of parameters to be updated, making it more efficient. For example, if we have a 5x5 weight matrix with 25 parameters, we can update a low-rank approximation of this matrix instead of the entire matrix.

Key Takeaways

- Fine-tuning is a form of transfer learning that involves making minor adjustments to a pre-trained model's internal parameters to optimize its performance on a new task.
- Full fine-tuning involves changing all the parameters of a pre-trained model, while PEFT involves updating only a portion of the parameters.
- LoRa is a method of PEFT that involves updating a low-rank approximation of the weight matrix instead of the entire matrix.
- Fine-tuning is useful for solving repetitive tasks at scale, making the model more accurate, aligned, and efficient for the specific use case.

Quantization

Quantization

Definition and Purpose

Quantization is the process of reducing the precision of the numbers used to represent model weights and activations. This reduction in precision is usually from 32-bit floating point (FP32) to lower-bit formats like 8-bit integers (INT8) or 16-bit floats (FP16, BF16). The primary purpose of quantization is to compress the range of continuous signals to a set of discrete values, thereby reducing the memory bandwidth requirement and increasing cache utilization.

Quantization Process

The quantization process involves selecting certain discrete values from a continuous signal. This is done by dividing the continuous signal into ranges, each representing an integer number. For example, if we have 16 levels, it is called 4-bit quantization, as we need 4 bits to represent these numbers.

Example

Suppose we have a continuous signal with values ranging from minus infinity to plus infinity. We want to quantize this signal to only certain discrete values. Let's say we have 16 levels, each representing an integer number. If we make a measurement and the value is close to one of these discrete values, we assign that value to it. For instance, if the measurement is 2.3, which is close to 2, we assign the value 2 to it.

Signed Quantization

Signed quantization is a type of quantization where the range of values is divided into positive and negative ranges. For example, we can have 16 levels, with 0th level representing 0.2, 1st level representing -1, 2nd level representing 0.5, and so on.

Advantages

Quantization improves performance by reducing memory bandwidth requirement and increasing cache utilization. It is often a trade-off based on the use case to go with a model that is quantized.

Reversibility

Quantization is a reversible process, meaning that we can always change it back to the original continuous signal. However, there will be some amount of error involved in this process.

Applications

Quantization is used in many systems, including large language models. It is an important step in compressing the range of continuous signals to a set of discrete values, making it an essential concept in machine learning and signal processing.

Quantization(BFLOAT16)

Quantization (BFLOAT16)

Introduction

Quantization is an important step in many systems, which involves compressing the range of floating-point numbers to a smaller range of integers. This process reduces the memory required to store the weights, making it efficient for large-scale training and inference.

Quantization Process

The quantization process involves reducing the range of floating-point numbers from minus infinity to plus infinity to a smaller range of integers. For example, if we use 4-bit quantization, the range is reduced to 16 levels, which can be represented using 4 bits. Each of these 16 levels represents an integer number.

Example of Quantization

Let's take an example of a floating-point weight, 2.0, which has been learned during training. To quantize this weight, we need to find the corresponding integer value. In this case, the integer value is 15. Instead of storing the floating-point weight, we store the integer value 15.

Quantization Levels

The quantization levels can be represented as follows:

Level	Value
0	-1.5
1	-1.0
2	-0.8
3	-0.5
...	...
15	1.5

Advantages of Quantization

The advantages of quantization are:

- Reduced memory usage: By storing the weights as integers, we can reduce the memory usage significantly. For example, if we have 176 billion weights, storing them as floating-point numbers would require 800 GB of memory. However, by using 4-bit quantization, we can reduce the memory usage to approximately 80 GB.
- Faster computation: Quantization also leads to faster computation, as integer operations are faster than floating-point operations.

Types of Quantization

There are different types of quantization, including:

- FP32 (Floating-Point 32): Offers high precision and wide range, but is heavy on memory and compute.
- FP16 (Floating-Point 16): Faster and uses less memory, but has limited range and precision, which can lead to instability during training.
- INT8 (Integer 8): Extremely compact and fast, but is mostly used for inference due to its low precision.
- BFLOAT16 (Brain Floating-Point 16): Strikes a balance between precision and memory usage, making it ideal for efficient, large-scale training without the numerical instability of FP16.

Conclusion

In conclusion, quantization is an important step in reducing the memory usage and improving the computational efficiency of neural networks. By understanding the quantization process and its advantages, we can make informed decisions about the type of quantization to use for our specific application.

UE22AM343BB5

UE22AM343BB5: Floating Point Representation and Quantization

Floating Point Representation

In computer architecture, floating-point numbers are represented using a 32-bit format. This format consists of:

- 1 bit for the sign bit
- 8 bits for the exponent
- 23 bits for the mantissa

This 32-bit representation is used for every number. For example, the number 176 billion can be represented in this format.

Precision and Error

When reducing the precision from 32-bit to 16-bit, the number of bits available to represent the mantissa decreases, leading to a loss of information. This results in an error in the representation of the number. For instance, the number π, which is approximately 3.14159265 in 32-bit precision, degrades to 3.125 in 16-bit precision.

Memory Constraints

In the context of large language models, the memory required to store the model's weights and gradients can be substantial. For example, a model with 176 billion parameters requires approximately 800 GB of memory to store the weights alone. Additionally, the gradients require another 800 GB of memory, making it a significant challenge to store and process these models.

Quantization

To address the memory constraints, quantization can be used to reduce the precision of the weights and gradients. One approach is 4-bit quantization, which discretizes the range of values from -∞ to ∞ to 2^4 (16) levels. This allows the values to be represented using int8. The 16 possible values are:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

Transfer Learning and Fine-Tuning

In transfer learning, a pre-trained model with 176 billion parameters is loaded, and the model is fine-tuned using a new dataset. This involves updating all 176 billion parameters, which requires generating 176 billion gradients. The stochastic gradient update rule is then applied, requiring a significant amount of memory and computational resources.

Types of Fine-Tuning : PEFT (Parameter-Efficient Fine-Tuning)

Types of Fine-Tuning: PEFT (Parameter-Efficient Fine-Tuning)

Introduction to Fine-Tuning

Fine-tuning is a crucial concept in Large Language Models (LLMs) that involves updating the pre-trained model's parameters to adapt to a new dataset. In this lecture, we will focus on Parameter-Efficient Fine-Tuning (PEFT), a type of fine-tuning that updates only a portion of the model's parameters.

Full Fine-Tuning vs. PEFT

Full fine-tuning involves updating all 176 billion parameters of a pre-trained model, which is computationally expensive and requires significant memory. This approach is not commonly used due to infrastructural limitations. On the other hand, PEFT updates only a subset of the model's parameters, making it a more efficient and feasible approach.

PEFT: Updating a Portion of Parameters

In PEFT, a small subset of model parameters is updated during training. This approach requires significantly less compute and memory, making it faster and more efficient. The optimizer and gradients apply only to the small trainable components, reducing the computational overhead.

Methods of PEFT

There are several methods of PEFT, including:

- **LoRa**: One of the methods of PEFT that updates a portion of the model's parameters. LoRa is a type of parameter-efficient fine-tuning that freezes the pre-trained model's parameters and trains a new set of parameters altogether.
- **Q-Lora**: Another method of PEFT that updates a portion of the model's parameters.
- **Prefix Tuning**: A method of PEFT that updates a portion of the model's parameters.

Advantages of PEFT

PEFT offers several advantages over full fine-tuning, including:

- **Faster Training**: PEFT requires fewer epochs and is generally faster than full fine-tuning.
- **Less Compute and Memory**: PEFT requires significantly less compute and memory, making it more efficient.
- **Effective with Smaller Datasets**: PEFT can be done effectively even with smaller datasets.

Conclusion

In conclusion, PEFT is a type of fine-tuning that updates only a portion of the model's parameters, making it a more efficient and feasible approach than full fine-tuning. LoRa, Q-Lora, and prefix tuning are some of the methods of PEFT that can be used to adapt pre-trained models to new datasets.

Problems with Full Fine-Tuning : Catastrophic Forgetting

Problems with Full Fine-Tuning: Catastrophic Forgetting

Full Fine-Tuning

Full fine-tuning involves updating all the parameters of a pre-trained model based on a new dataset. In the case of a 176 billion parameter model, this would mean updating all 176 billion parameters, generating 176 gradients, and performing 176 stochastic gradient updates. This process requires a significant amount of memory and computational resources.

Limitations of Full Fine-Tuning

However, full fine-tuning is not commonly practiced due to the limitations of infrastructure support. It is not a matter of intelligence, but rather a lack of resources that prevents us from performing full fine-tuning.

Catastrophic Forgetting

One of the significant problems with full fine-tuning is catastrophic forgetting. This phenomenon occurs when a model forgets or loses previously learned information as it is trained on new data or fine-tuned for specific tasks. This happens because the training process prioritizes recent data or tasks at the expense of earlier data, leading to a degradation or overwriting of the model's representations of certain concepts or knowledge.

Consequences of Catastrophic Forgetting

Catastrophic forgetting results in a loss of overall performance, as the model's ability to recall previously learned information is compromised.

Parameter Efficient Fine-Tuning (PEFT)

To address the limitations of full fine-tuning and catastrophic forgetting, parameter efficient fine-tuning (PEFT) is used. PEFT involves updating only a portion of the model's parameters, rather than the entire model. This approach is more efficient in terms of memory and computational resources.

LoRA (Low-Rank Adaptation)

One type of PEFT is LoRA (Low-Rank Adaptation). LoRA involves injecting a low-rank matrix into the adapter layers and learning a smaller number of parameters. This approach requires less memory, as only the smaller number of parameters need to be stored. During inference, the fine-tuned weights are combined with the original pre-trained weights to produce the final output.

Preventing Catastrophic Forgetting

LoRA and multi-task learning are some ways to prevent catastrophic forgetting. By using PEFT and LoRA, we can fine-tune models more efficiently and prevent the loss of previously learned information.

Key Takeaways

- Full fine-tuning involves updating all model parameters, which is not commonly practiced due to infrastructure limitations.
- Catastrophic forgetting is a significant problem with full fine-tuning, leading to a loss of overall performance.
- Parameter efficient fine-tuning (PEFT) and LoRA are approaches that address the limitations of full fine-tuning and prevent catastrophic forgetting.

Why Fine-tuning

Why Fine-tuning

Introduction to Fine-tuning

Fine-tuning is a crucial concept in Large Language Models (LLMs) like GPT. These models are trained on massive, general datasets (books, websites, etc.) and are incredibly powerful out-of-the-box. However, they might not perform optimally for specific domains, tasks, or user behavior.

Need for Fine-tuning

LLMs have a need for repetitive prompting instructions. Users must repeatedly include detailed instructions in every prompt to get consistent specialized outputs. This is where fine-tuning comes in, allowing LLMs to adapt to specific tasks or domains.

Full Fine-tuning vs. Parameter Efficient Fine-tuning (PEFT)

Full fine-tuning involves updating all 176 billion parameters of a pre-trained model based on new data. This is not feasible due to memory constraints. Instead, PEFT is used, which updates only a portion of the parameters, freezing the rest.

Methods of PEFT

There are several methods of PEFT, including LoRa, QLoRa, and prefix tuning. These methods update only a portion of the parameters, making fine-tuning more efficient.

LoRa

LoRa is one of the methods of PEFT. It freezes the 176 billion parameters of the pre-trained model and trains a new set of parameters altogether. This approach is more efficient than full fine-tuning and allows for more targeted adaptation to specific tasks or domains.

Challenges of Fine-tuning

Fine-tuning is not without its challenges. One of the main issues is the lack of infrastructural support, which can limit the ability to perform full fine-tuning. This is why PEFT methods like LoRa are used instead.

Key Takeaways

- Fine-tuning is necessary for LLMs to adapt to specific tasks or domains.
- Full fine-tuning is not feasible due to memory constraints.
- PEFT methods like LoRa, QLoRa, and prefix tuning are used instead.
- LoRa freezes the pre-trained model's parameters and trains a new set of parameters altogether.
- Fine-tuning is limited by infrastructural support.

Types of Fine-Tuning : Full Fine-Tuning

Types of Fine-Tuning: Full Fine-Tuning

What is Fine-Tuning?

Fine-tuning is a crucial concept in Large Language Models (LLMs) that involves updating the pre-trained model's parameters to adapt to a new dataset. In the context of LLMs, fine-tuning is essential to achieve optimal performance on a specific task.

Full Fine-Tuning

Full fine-tuning involves updating all the parameters of a pre-trained model, which can be a massive undertaking, especially for large models with billions of parameters. In the case of a 176 billion parameter GPT model, full fine-tuning would require updating all 176 billion parameters based on the new dataset.

Characteristics of Full Fine-Tuning

Full fine-tuning has several characteristics that make it a challenging and resource-intensive process:

- **All model parameters are updated during training**: This means that every single parameter of the pre-trained model is adjusted to fit the new dataset.
- **Requires backpropagation through the entire model**: The entire model needs to be traversed to compute the gradients, which can be computationally expensive.
- **Needs more compute and memory**: Full fine-tuning requires significant computational resources and memory, especially for large models.
- **Training is slower and generally needs longer epochs**: The process of updating all parameters can be slow and may require more training epochs to converge.
- **Typically done on task-specific labeled data**: Full fine-tuning is usually performed on task-specific labeled data to adapt the model to the new task.

Limitations of Full Fine-Tuning

Full fine-tuning is not always feasible due to the following limitations:

- **Infrastructure constraints**: The infrastructure may not support the computational requirements of full fine-tuning, making it impractical.
- **Memory constraints**: The memory required to store the gradients and perform the updates can be prohibitively large.

Parameter Efficient Fine-Tuning (PEFT)

To overcome the limitations of full fine-tuning, parameter efficient fine-tuning (PEFT) methods are used. PEFT involves updating only a portion of the model's parameters, rather than all of them. This approach is more efficient and can be achieved through methods such as LoRa, QLoRa, and prefix tuning.
