PEFT (Parameter-Efficient Fine-Tuning)

Overview: PEFT stands for Parameter-Efficient Fine-Tuning, a class of techniques that aim to adapt large pre-trained models to various downstream tasks and domains without modifying all the model parameters. PEFT techniques can significantly reduce the computational and storage costs of fine-tuning, while achieving comparable or even better performance than full model fine-tuning.

Motivation: Large pre-trained models, such as BERT, GPT-3, and T5, have shown impressive results on a wide range of natural language processing tasks, such as text classification, natural language generation, and natural language understanding. However, these models are often prohibitively expensive to train and fine-tune, as they require huge amounts of data, computational resources, and memory. Moreover, fine-tuning the entire model can lead to overfitting or catastrophic forgetting, especially when the downstream task or domain is different from the pre-training data or objective.

Solution: PEFT techniques offer a solution to these challenges by only updating a small fraction of the model parameters, while keeping the rest of the parameters frozen. This way, PEFT techniques can leverage the general knowledge and capabilities of the pre-trained model, while adding task-specific or domain-specific adaptations. PEFT techniques can also enable multi-task learning, where different tasks can share the same backbone model, but have different fine-tuned parameters.

Types: There are several types of PEFT techniques, depending on how they modify the pre-trained model. Some of the most popular types are:

- **Adapters**: Adapters are small, learnable modules that are inserted between the layers of the pre-trained model. They apply a down-projection, a non-linearity, and an up-projection to the layer outputs, and add them back to the original outputs via residual connections.
- **Soft Prompts**: Soft prompts are learnable embeddings that are used as inputs to the pre-trained model. They are concatenated or added to the original inputs, and act as guides or hints for the model to produce the desired outputs.
- **LoRA and QLoRA**: LoRA and QLoRA are learnable low-rank matrices that are used to approximate the weights of the pre-trained model. They multiply or add the original weights with the low-rank matrices, and reduce the number of parameters and computations.
- **IA3**: IA3 are learnable attention masks that are used to modulate the information flow within the pre-trained model. They multiply or add the intermediate representations of the model with the attention masks, and control the amount of attention that the model pays to different parts of the inputs and outputs.
- **Prefix Tuning, Prompt Tuning, and P-tuning**: These techniques are based on the idea of using natural language prompts to elicit the desired outputs from the pre-trained model. Prefix tuning learns a sequence of tokens that are prepended to the inputs, prompt tuning learns a template that formats the inputs and outputs, and P-tuning learns a set of parameters that are used to generate the prompts.

Adapters

Overview: Adapters are a parameter-efficient method for fine-tuning pre-trained models by inserting small, learnable modules between the layers of the model. Instead of updating all the parameters of the model, adapters only update the parameters of these modules, and the parameters of the original model are kept frozen.

Intuition: Imagine having a large, complex machine that can process different types of inputs and produce different types of outputs. Instead of modifying the machine for a new task, you add small, specialized components that adapt the machine's existing capabilities to perform the new task efficiently.

Definition: Adapters are lightweight modules that are inserted between the layers of a pre-trained model. During fine-tuning, only the parameters of these modules are updated, and the parameters of the original model are kept frozen. This allows for rapid and efficient transfer learning while reducing the risk of overfitting on smaller datasets.

Essential Mathematical Relations:

- **Adapter Module**: Given a layer output z, an adapter module applies a down-projection D, a non-linearity ϕ, and an up-projection U, so the adapter output a is given by $a = U(\phi(D(z)))$.
- **Residual Connection**: The output of the adapter module is typically added back to the original layer output z to form the final output $y = z + a$.

Computational Complexity: Adapters add a small overhead to the computational complexity of the model. If n is the dimensionality of the layer output and m is the dimensionality of the adapter, the complexity is $O(nm)$ for each adapter module.

Space Complexity: The space complexity is $O(nm + m^2)$ per adapter module, which is significantly lower than fine-tuning the entire pre-trained model with $O(n^2)$ parameters.

Things To Note:

- Adapters allow for task-specific fine-tuning without the need to re-train the entire model.
- They are beneficial for multi-task learning where each task can have its own adapter while sharing the backbone model.
- Adapters are also useful for deploying models where storage or memory is limited.

Drawbacks and Downsides:

- The effectiveness of adapters can depend on the choice of bottleneck size m.
- They may not capture as much task-specific information as full model fine-tuning, potentially leading to slightly lower performance on certain tasks.

Soft Prompts

Overview: Soft Prompts are a parameter-efficient method for fine-tuning pre-trained models by learning continuous embeddings that act as inputs to the model. Instead of updating the parameters of the model, Soft Prompts optimize the embeddings that are concatenated or added to the original inputs.

Intuition: Imagine having a large, complex machine that can process different types of inputs and produce different types of outputs. Instead of modifying the machine for a new task, you create a small, flexible device that can generate inputs that are compatible with the machine and guide it to produce the desired outputs.

Definition: Soft Prompts are learnable embeddings that are used as inputs to a pre-trained model. During fine-tuning, only the parameters of these embeddings are updated, and the parameters of the original model are kept frozen. This allows for efficient and flexible adaptation of the model to various tasks and domains.

Essential Mathematical Relations:

- **Soft Prompt Embedding**: Given an input x, a soft prompt embedding p is a vector of the same dimensionality as the input embedding. The soft prompt embedding can be either concatenated or added to the input embedding, depending on the task and the model architecture.
- **Soft Prompt Output**: The output of the model with a soft prompt embedding is computed as usual, except that the input is modified by the soft prompt embedding. For example, if the input is a sequence of tokens $x_1, x_2, ..., x_n$, and the soft prompt embedding is p, the output is $f([p, x_1, x_2, ..., x_n])$ or $f([p + x_1, p + x_2, ..., p + x_n])$, where f is the model function.

Computational Complexity: Soft Prompts add a negligible overhead to the computational complexity of the model. If d is the dimensionality of the input embedding and k is the number of soft prompt embeddings, the complexity is $O(kd)$ for each input.

Space Complexity: The space complexity is $O(kd)$ per input, which is much lower than fine-tuning the entire pre-trained model with $O(d^2)$ parameters.

Things To Note:

- Soft Prompts can be used for various tasks, such as text classification, natural language generation, and natural language understanding, by adjusting the number and position of the soft prompt embeddings.
- They can also be used for domain adaptation, by learning soft prompt embeddings that are specific to a certain domain or style of language.
- Soft Prompts can be combined with other PEFT methods, such as adapters, to further improve the performance and efficiency of the model.

Drawbacks and Downsides:

- The quality of soft prompts can depend on the choice of initialization, optimization, and regularization methods.
- They may not be able to capture complex or long-range dependencies between the inputs and the outputs, as they only modify the input embeddings.
- They may require more hyperparameter tuning and experimentation to find the optimal configuration for a given task or domain.

LoRA and QLoRA

Overview: LoRA and QLoRA are parameter-efficient methods for fine-tuning pre-trained models by learning low-rank adaptations of the model's weights. Instead of updating all the parameters of the model, LoRA and QLoRA optimize a small subset of parameters that are used to approximate the full model's weights.

Intuition: Imagine having a large, complex machine that can process different types of inputs and produce different types of outputs. Instead of modifying the machine for a new task, you create a small, flexible device that can generate weights that are compatible with the machine and guide it to produce the desired outputs.

Definition: LoRA and QLoRA are learnable low-rank matrices that are used to modify the weights of a pre-trained model. During fine-tuning, only the parameters of these matrices are updated, and the parameters of the original model are kept frozen. This allows for effective and selective adaptation of the model to various tasks and domains.

Essential Mathematical Relations:

- **LoRA Matrix**: Given a weight matrix W of size $n \times n$, a LoRA matrix L is a low-rank approximation of W obtained by multiplying two smaller matrices A and B of size $n \times m$ and $m \times n$, respectively, where $m < n$. The LoRA matrix is given by $L = AB$.
- **QLoRA Matrix**: Given a LoRA matrix L, a QLoRA matrix Q is a quantized version of L obtained by applying a quantization function q that maps each element of L to a discrete value from a predefined set. The QLoRA matrix is given by $Q = q(L)$.

Computational Complexity: LoRA and QLoRA reduce the computational complexity of the model. If n is the dimensionality of the weight matrix and m is the rank of the LoRA matrix, the complexity is $O(nm)$ for each LoRA or QLoRA matrix.

Space Complexity: LoRA and QLoRA reduce the space complexity of the model. If n is the dimensionality of the weight matrix and m is the rank of the LoRA matrix, the space complexity is $O(nm)$ for each LoRA matrix and $O(m)$ for each QLoRA matrix, which is much lower than fine-tuning the entire pre-trained model with $O(n^2)$ parameters.

Things To Note:

- LoRA and QLoRA can be used for various tasks, such as text classification, natural language generation, and natural language understanding, by adjusting the rank and position of the LoRA or QLoRA matrices.
- They can also be used for domain adaptation, by learning LoRA or QLoRA matrices that are specific to a certain domain or style of language.
- LoRA and QLoRA can be combined with other PEFT methods, such as adapters, to further improve the performance and efficiency of the model.

Drawbacks and Downsides:

- The quality of LoRA and QLoRA matrices can depend on the choice of initialization, optimization, and regularization methods.
- They may not be able to capture complex or long-range dependencies between the inputs and the outputs, as they only modify the weight matrices.
- They may require more hyperparameter tuning and experimentation to find the optimal rank and quantization level for a given task or domain.

Intrinsic Attention Adapters (IA3)

Overview: Intrinsic Attention Adapters (IA3) are a parameter-efficient method for fine-tuning pre-trained models by learning attention masks that modulate the information flow within the model. Instead of updating the parameters of the model, IA3 optimize the attention masks that are applied to the intermediate representations of the model.

Intuition: Imagine having a large, complex machine that can process different types of inputs and produce different types of outputs. Instead of modifying the machine for a new task, you create a small, flexible device that can control the amount of attention that the machine pays to different parts of the inputs and outputs.

Definition: IA3 are learnable attention masks that are used to modify the intermediate representations of a pre-trained model. During fine-tuning, only the parameters of these masks are updated, and the parameters of the original model are kept frozen. This allows for effective and selective adaptation of the model to various tasks and domains.

Essential Mathematical Relations:

- **IA3 Mask**: Given an intermediate representation h, an IA3 mask m is a vector of the same dimensionality as h. The IA3 mask can be either multiplied or added to h, depending on the task and the model architecture.
- **IA3 Output**: The output of the model with an IA3 mask is computed as usual, except that the intermediate representation is modified by the IA3 mask. For example, if the intermediate representation is a sequence of hidden states $h_1, h_2, ..., h_n$, and the IA3 mask is m, the output is $f([m \odot h_1, m \odot h_2, ..., m \odot h_n])$ or $f([m + h_1, m + h_2, ..., m + h_n])$, where f is the model function and $\odot$ is the element-wise product.

Computational Complexity: IA3 add a negligible overhead to the computational complexity of the model. If d is the dimensionality of the intermediate representation and k is the number of IA3 masks, the complexity is $O(kd)$ for each intermediate representation.

Space Complexity: The space complexity is $O(kd)$ per intermediate representation, which is much lower than fine-tuning the entire pre-trained model with $O(d^2)$ parameters.

Things To Note:

- IA3 can be used for various tasks, such as text classification, natural language generation, and natural language understanding, by adjusting the number and position of the IA3 masks.
- They can also be used for domain adaptation, by learning IA3 masks that are specific to a certain domain or style of language.
- IA3 can be combined with other PEFT methods, such as adapters, to further improve the performance and efficiency of the model.

Drawbacks and Downsides:

- The quality of IA3 masks can depend on the choice of initialization, optimization, and regularization methods.
- They may not be able to capture complex or long-range dependencies between the inputs and the outputs, as they only modify the intermediate representations.
- They may require more hyperparameter tuning and experimentation to find the optimal configuration for a given task or domain.

Comparison of PEFT Techniques

Overview: This section compares the different PEFT techniques discussed in the previous sections, such as adapters, soft prompts, LoRA, QLoRA, IA3, prefix tuning, prompt tuning, and P-tuning. It also provides some guidelines on when to use which technique, and what are the pros and cons of each technique.

Criteria: The comparison is based on the following criteria:

- **Parameter Efficiency**: The ratio of the number of trainable parameters to the number of original model parameters. The lower the ratio, the higher the parameter efficiency.
- **Computational Efficiency**: The ratio of the computational complexity of the fine-tuned model to the computational complexity of the original model. The lower the ratio, the higher the computational efficiency.
- **Space Efficiency**: The ratio of the space complexity of the fine-tuned model to the space complexity of the original model. The lower the ratio, the higher the space efficiency.
- **Performance**: The quality of the fine-tuned model on the downstream task, measured by task-specific metrics such as accuracy, F1-score, BLEU, etc. The higher the metric, the better the performance.
- **Flexibility**: The ability of the fine-tuned model to adapt to various tasks and domains, without requiring extensive hyperparameter tuning or re-training. The higher the flexibility, the easier the adaptation.

Table: The table below summarizes the comparison of the PEFT techniques based on the criteria. The values are approximate and may vary depending on the task, the model, and the implementation details.

Technique	Parameter Efficiency	Computational Efficiency	Space Efficiency	Performance	Flexibility
Adapters	High	Low	High	High	High
Soft Prompts	High	High	High	Medium	High
LoRA	Medium	High	Medium	High	Medium
QLoRA	Medium	Very High	Very High	Medium	Medium
IA3	High	High	High	Medium	High
Prefix Tuning	Low	Low	Low	High	Low
Prompt Tuning	Low	Low	Low	High	Low
P-tuning	Low	Low	Low	High	Low

Guidelines: Based on the comparison, some general guidelines on when to use which technique are:

- Use adapters when you want to fine-tune a large model on multiple tasks or domains, and you have enough computational resources to handle the overhead of the adapter layers.
- Use soft prompts when you want to fine-tune a large model on various tasks or domains, and you want to minimize the computational and space costs while maintaining a reasonable performance.
- Use LoRA or QLoRA when you want to fine-tune a large model on a specific task or domain, and you want to achieve a high performance while reducing the computational and space costs. Choose LoRA for higher performance and QLoRA for higher efficiency.
- Use IA3 when you want to fine-tune a large model on various tasks or domains, and you want to modulate the information flow within the model without adding too many parameters or computations.
- Use prefix tuning, prompt tuning, or P-tuning when you want to fine-tune a large model on a specific task or domain, and you want to achieve a high performance without modifying the original model parameters. Choose prefix tuning for generation tasks, prompt tuning for classification tasks, and P-tuning for tasks that require natural language prompts.