1.基本概念

1.微调方法是啥？如何微调？

微调（Fine-tuning）是**一种迁移学习的方法**，用于在一个预训练模型的基础上，通过在特定任务的数据上进行有监督训练，来适应该任务的要求并提高模型性能。微调利用了预训练模型在大规模通用数据上学习到的语言知识和表示能力，将其迁移到特定任务上。

下面是一般的微调步骤：

1. **预训练模型选择**：选择一个在大规模数据上进行预训练的模型作为基础模型。例如，可以选择一种预训练的语言模型，如BERT、GPT等。
2. **数据准备**：准备用于微调的特定任务数据集。这些数据集应包含任务相关的样本和相应的标签或目标。确保数据集与任务的特定领域或问题相关。
3. **构建任务特定的模型头**：根据任务的要求，构建一个特定的模型头（task-specific head）。模型头是添加到预训练模型之上的额外层或结构，用于根据任务要求进行输出预测或分类。例如，对于文本分类任务，可以添加一个全连接层和softmax激活函数。
4. **参数初始化**：将预训练模型的参数作为初始参数加载到微调模型中。这些参数可以被视为模型已经学习到的通用语言表示。
5. **微调训练**：使用特定任务的数据集对模型进行有监督训练。这包括将任务数据输入到模型中，计算损失函数，并通过反向传播和优化算法（如梯度下降）更新模型参数。在微调过程中，只有模型头的参数会被更新，而预训练模型的参数会保持不变。
6. **调整超参数**：微调过程中，可以根据需要调整学习率、批量大小、训练迭代次数等超参数，以达到更好的性能。
7. **评估和验证**：在微调完成后，使用验证集或测试集对微调模型进行评估，以评估其在特定任务上的性能。可以使用各种指标，如准确率、精确率、召回率等。
8. **可选的后续微调**：根据实际情况，可以选择在特定任务的数据上进行进一步的微调迭代，以进一步提高模型性能。

微调的关键是在预训练模型的基础上进行训练，从而将模型的知识迁移到特定任务上。通过这种方式，可以在较少的数据和计算资源下，快速构建和训练高性能的模型。

2.为什么需要 PEFT?

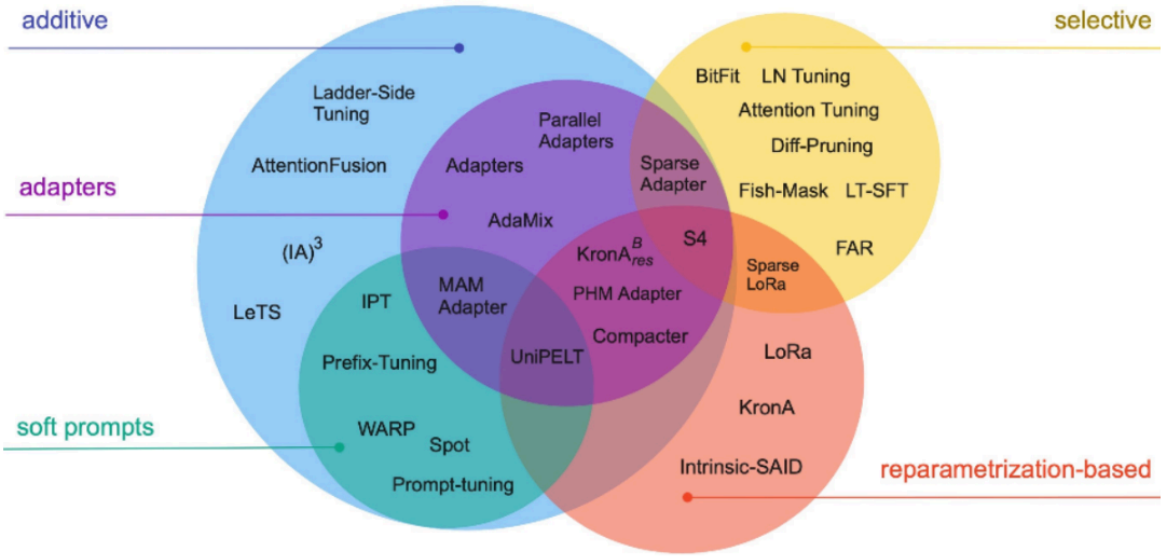
Parameter-Efficient Fine-Tuning（PEFT）是一种微调策略，**旨在仅训练少量参数使模型适应到下游任务**。对大规模PLM（pre-trained language models ）进行微调的成本往往高得令人望而却步。在这方面，PEFT方法只微调了少量(额外的)模型参数，从而大大降低了计算和存储成本。最近最先进的PEFT技术实现了与完全微调相当的性能。

PEFT**通过冻结预训练模型的某些层，并仅微调特定于下游任务的最后几层来实现这种效率。**这样，模型就可以适应新的任务，计算开销更少，标记的例子也更少。尽管PEFT是一个相对较新的概念，但自从引入迁移学习以来，更新最后一层模型已经在计算机视觉领域得到了实践。即使在NLP中，静态和非静态词嵌入的实验也很早就进行了。

参数高效微调旨在提高预训练模型(如BERT和RoBERTa)在各种下游任务上的性能，包括情感分析、命名实体识别和问答。它在数据和计算资源有限的低资源设置中实现了这一点。它只修改模型参数的一小部分，并且不容易过度拟合。

参数高效的微调**在计算资源有限或涉及大型预训练模型的情况下特别有用。**在这种情况下，PEFT可以在不牺牲性能的情况下提供一种更有效的方法来微调模型。然而，需要注意的是，PEFT有时可能会达到与完全微调不同的性能水平，特别是在预训练模型需要进行重大修改才能在新任务上表现良好的情况下。

高效微调技术可以粗略分为以下三大类：增加额外参数（A）、选取一部分参数更新（S）、引入重参数化（R）。而在增加额外参数这类方法中，又主要分为类适配器（Adapter-like）方法和软提示（Soft prompts）两个小类。



Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning

3.微调和参数高效微调之间的区别是什么?

微调和参数高效微调是机器学习中用于**提高预训练模型在特定任务上的性能**的两种方法。

微调就是把**一个预先训练好的模型用新的数据在一个新的任务进一步训练它。**整个预训练模型通常在微调中进行训练，包括它的所有层和参数。这个过程在计算上非常昂贵且耗时，特别是对于大型模型。

另一方面，**参数高效微调**是**一种专注于只训练预训练模型参数的子集的微调方法。**这种方法包括为新任务识别最重要的参数，并且只在训练期间更新这些参数。这样，PEFT可以显著减少微调所需的计算量。

4.PEFT 有什么优点?

在这里，只讨论PEFT相对于传统微调的好处。因此，理解为什么参数有效的微调比微调更有益。

1. **减少计算和存储成本**：PEFT只涉及微调少量额外的模型参数，而冻结预训练llm的大部分参数，从而显着降低计算和存储成本。
2. **克服灾难性遗忘**：在LLM的全面微调期间，灾难性遗忘可能发生在模型忘记它在预训练期间学到的知识的地方。PEFT通过只更新几个参数来克服这个问题。
3. **低数据环境下更好的性能**：PEFT方法在低数据环境下的表现优于完全微调，并且可以更好地推广到域外场景。
4. **可移植性**：与全面微调的大检查点相比，PEFT方法使用户能够获得价值几mb的小检查点。这使得来自PEFT方法的训练权重易于部署和用于多个任务，而无需替换整个模型。
5. **与完全微调相当的性能**：PEFT仅使用少量可训练参数即可实现与完全微调相当的性能。

5.多种不同的高效微调方法对比

参数有效策略可能涉及多种技术：

1. **选择性层调整**（**Selective Layer Tuning**）：可以只微调层的一个子集，而不是微调模型的所有层。这减少了需要更新的参数数量。
2. **适配器**（**Adapters**）：适配器并不是在大模型的输出之后加一个简单的模型层，而是插入到**预训练模型的内部层之间**，通常是在模型的每一层或某些关键层中插入小型的神经网络。这些小网络被称为**适配器层**。在训练时，适配器层会根据新任务的需求进行训练，而预训练模型的其他参数（如BERT的Transformer层）则保持冻结，不参与训练。
3. **稀疏微调**（**Sparse Fine-Tuning**）：传统的微调会略微调整所有参数，但稀疏微调只涉及更改模型参数的一个子集。这通常是基于一些标准来完成的，这些标准标识了与新任务最相关的参数。
4. **低秩近似**（**Low-Rank Approximations**）：另一种策略是用一个参数较少但在任务中表现相似的模型来近似微调后的模型。低秩近似是一种常见的降维技术，目的是通过将一个高维的矩阵（或者模型）近似为一个低维的矩阵来减少参数数量，从而降低计算和存储成本，同时尽量保持原模型的表现。具体来说，低秩近似通过分解矩阵为两个低秩矩阵的乘积（例如，奇异值分解，SVD）来近似原始矩阵。
5. **正则化技术**（**Regularization Techniques**）：可以将正则化项添加到损失函数中，以阻止参数发生较大变化，从而以更"参数高效"的方式有效地微调模型。
6. **任务特定的头**（**Task-specific Heads**）：有时，在预先训练的模型架构中添加一个任务特定的层或"头"，只对这个头进行微调，从而减少需要学习的参数数量。

6.当前高效微调技术存在的一些问题

当前的高效微调技术很难在类似方法之间进行直接比较并评估它们的真实性能，主要的原因如下所示：

- **参数计算口径不一致**：参数计算可以分为三类：可训练参数的数量、微调模型与原始模型相比改变的参数的数量、微调模型和原始模型之间差异的等级。例如，DiffPruning更新0.5%的参数，但是实际参与训练的参数量是200%。这为比较带来了困难。尽管可训练的参数量是最可靠的存储高效指标，但是也不完美。 Ladder-side Tuning使用一个单独的小网络，参数量高于LoRA或BitFit，但是因为反向传播不经过主网络，其消耗的内存反而更小。
- **缺乏模型大小的考虑**：已有工作表明，大模型在微调中需要更新的参数量更小（无论是以百分比相对而论还是以绝对数量而论），因此（基）模型大小在比较不同PEFT方法时也要考虑到。
- **缺乏测量基准和评价标准**：不同方法所使用的的模型/数据集组合都不一样，评价指标也不一样，难以得到有意义的结论。
- **代码实现可读性差**：很多开源代码都是简单拷贝Transformer代码库，然后进行小修小补。这些拷贝也不使用git fork，难以找出改了哪里。即便是能找到，可复用性也比较差（通常指定某个Transformer版本，没有说明如何脱离已有代码库复用这些方法）。

7.高效微调技术最佳实践

针对以上存在的问题，研究高效微调技术时，建议按照最佳实践进行实施：

- 明确指出参数数量类型。
- 使用不同大小的模型进行评估。
- 和类似方法进行比较。
- 标准化PEFT测量基准。
- 重视代码清晰度，以最小化进行实现。