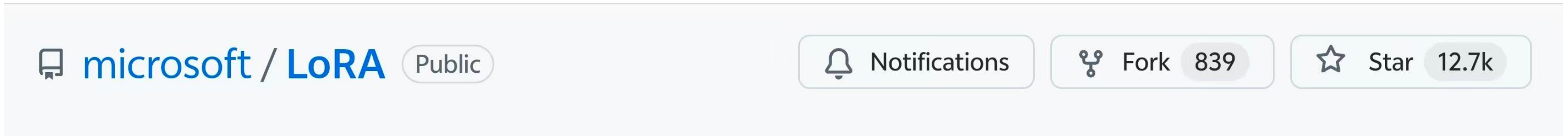
论文&源码浅读
LoRA微调 如何初始化

东川路第一可爱猫猫虫

The Impact of Initialization on LoRA Finetuning Dynamics



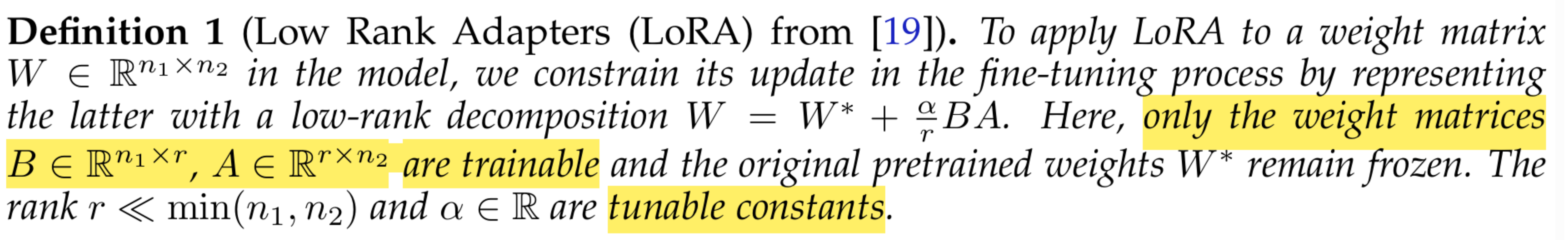
LoRA: Low-Rank Adaptation of Large Language Models

主要内容

- LoRA原仓库代码的初始化方式
- Embedding层和Linear层的不同
- 可能的原因和解释（便于理解）
- 一篇仅供参考的论文

- LoRA没必要太过纠结于初始化的方式，所有初始化最终是结果导向的，不同的数据、任务会有不同的结果，都可以事后解释

背景：什么是LoRA微调



Definition 1 (Low Rank Adapters (LoRA) from [19]). *To apply LoRA to a weight matrix $W \in \mathbb{R}^{n_1 \times n_2}$ in the model, we constrain its update in the fine-tuning process by representing the latter with a low-rank decomposition $W = W^* + \frac{\alpha}{r} BA$. Here, only the weight matrices $B \in \mathbb{R}^{n_1 \times r}$, $A \in \mathbb{R}^{r \times n_2}$ are trainable and the original pretrained weights W^* remain frozen. The rank $r \ll \min(n_1, n_2)$ and $\alpha \in \mathbb{R}$ are tunable constants.*

- 对于预训练权重矩阵W^*，LoRA通过把增量权重矩阵进行低秩分解，分解成两个可训练的低秩矩阵$B \in R^{n_1 \times r}, A \in R^{r \times n_2}$
- 其中α为可调常数，用来控制更新幅度
- Init[A]：B 矩阵初始化为全零，A 矩阵初始化为随机分布；
- Init[B]：A 矩阵初始化为全零，B 矩阵初始化为随机分布。

LoRA源码里的初始化

- Embedding层

```
def reset_parameters(self):
    nn.Embedding.reset_parameters(self)
    if hasattr(self, 'lora_A'):
        # initialize A the same way as the default for nn.Linear and B to zero
        nn.init.zeros_(self.lora_A)
        nn.init.normal_(self.lora_B)
```

- Linear层

```
def reset_parameters(self):
    nn.Linear.reset_parameters(self)
    if hasattr(self, 'lora_A'):
        # initialize B the same way as the default for nn.Linear and A to zero
        # this is different than what is described in the paper but should not affect performance
        nn.init.kaiming_uniform_(self.lora_A, a=math.sqrt(5))
        nn.init.zeros_(self.lora_B)
```

Embedding层和Linear层的不同

- Embedding层

  A全零初始化，B正态初始化

  ```
  nn.init.zeros_(self.lora_A)
  nn.init.normal_(self.lora_B)
  ```

- Linear层

  Akaiming初始化，B全零初始化

  ```
  nn.init.kaiming_uniform_(self.lora_A, a=math.sqrt(5))
  nn.init.zeros_(self.lora_B)
  ```

Embedding层

- 原embedding层权重的形状

  [num_embeddings,embedding_dim]
- 输入：形状为[batch_size,seq_len]的token，矩阵每行每列的元素都是一个整数索引，表示词表里的一个词
- 查表后：

  每个token被替换成一个embedding_dim维的向量
- 输出：

  变成[batch_size,seq_len, embedding_dim]维的张量

Embedding层的LoRA

- 低秩分解

  A 形状为[r, num_embeddings]

  B 形状为[embedding_dim,r]
- A 低秩嵌入表　　B 低秩映射回高秩
- A全零初始化，B正态初始化

  如果反过来，让A随机初始化，可能的后果

Linear层

- 原Linear层权重的形状
  [out_features,in_features]
- 输入：
  形状为[batch_size,in_features]的连续向量
- 输出：
  形状为[batch_size,out_features]
  $y = xW^T$

Linear层的LoRA

- 低秩分解

  A形状为[r,in_features]

  B形状为[out_features,r]
- A kaiming初始化，B 全零初始化
- 潜在的缺陷：

  B初始全0，梯度近似为0，参数等到第二轮才会开始更新
- 一个解决办法

  A、B都用非0初始化

  事先将预训练权重减去A_0B_0即可

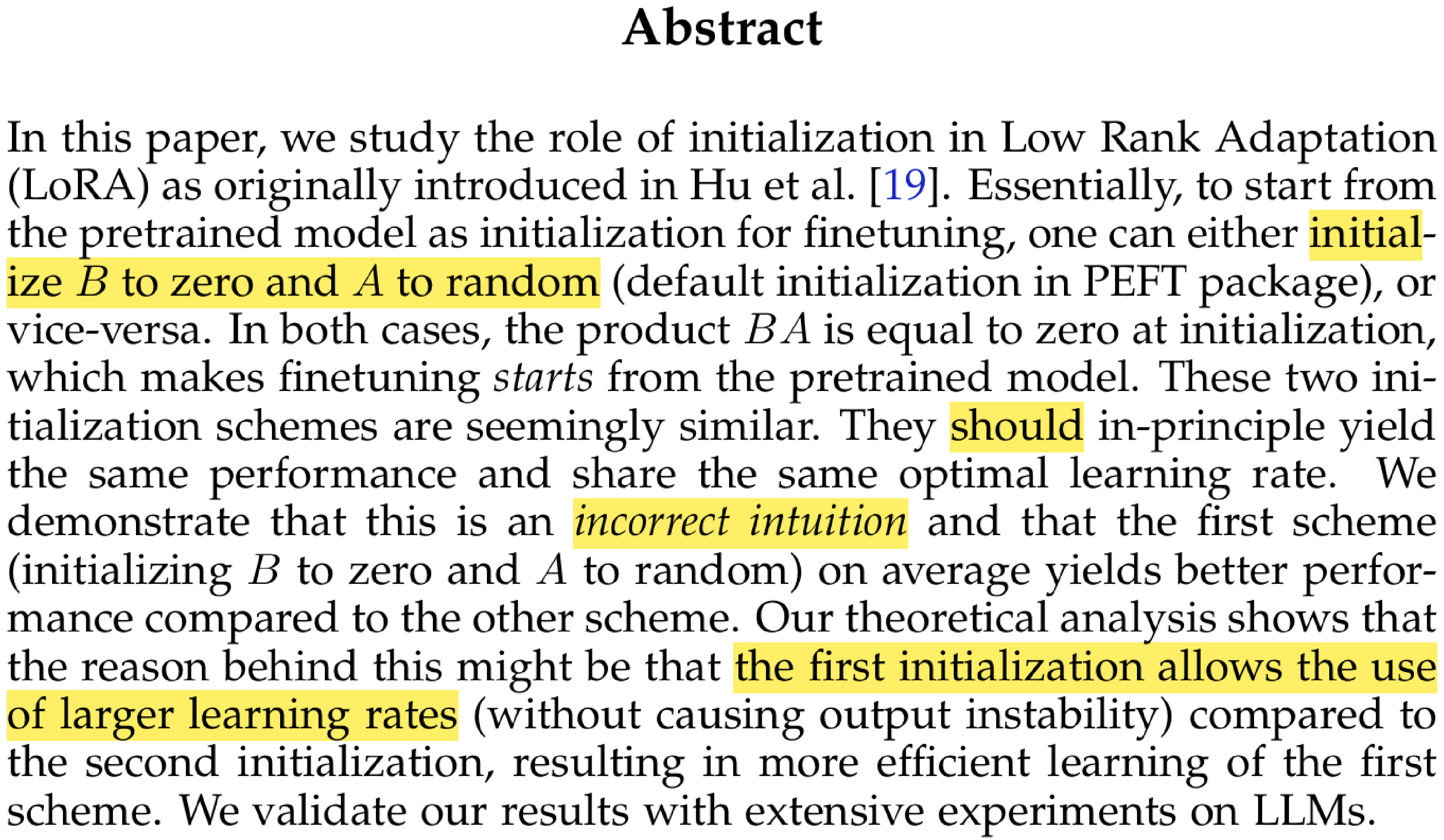
The Impact of Initialization on LoRA Finetuning Dynamics

Soufiane Hayou
Simons Institute
UC Berkeley
hayou@berkeley.edu

Nikhil Ghosh
Dept of Statistics
UC Berkeley
nikhil_ghosh@berkeley.edu

Bin Yu
Dept of Statistics
UC Berkeley
binyu@berkeley.edu

摘要



Abstract

In this paper, we study the role of initialization in Low Rank Adaptation (LoRA) as originally introduced in Hu et al. [19]. Essentially, to start from the pretrained model as initialization for finetuning, one can either initialize B to zero and A to random (default initialization in PEFT package), or vice-versa. In both cases, the product BA is equal to zero at initialization, which makes finetuning *starts* from the pretrained model. These two initialization schemes are seemingly similar. They should in-principle yield the same performance and share the same optimal learning rate. We demonstrate that this is an *incorrect intuition* and that the first scheme (initializing B to zero and A to random) on average yields better performance compared to the other scheme. Our theoretical analysis shows that the reason behind this might be that the first initialization allows the use of larger learning rates (without causing output instability) compared to the second initialization, resulting in more efficient learning of the first scheme. We validate our results with extensive experiments on LLMs.

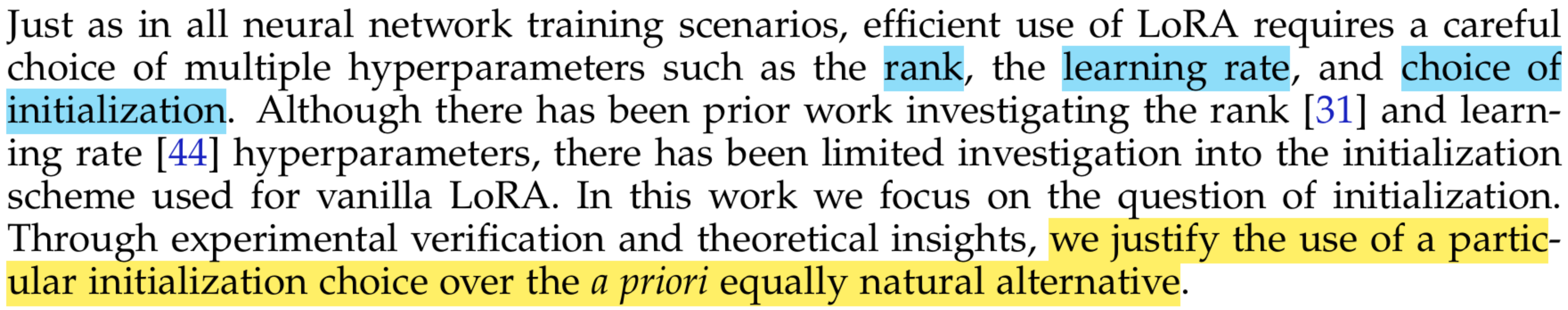
两种
初始化方法

看起来表现
应该一致

可是实际上
A优于B

可能的原因

本文研究LoRA的初始化方法

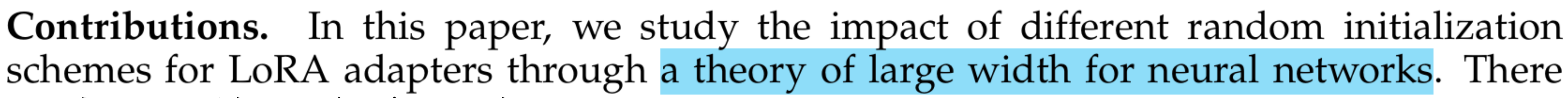

Just as in all neural network training scenarios, efficient use of LoRA requires a careful choice of multiple hyperparameters such as the rank, the learning rate, and choice of initialization. Although there has been prior work investigating the rank [31] and learning rate [44] hyperparameters, there has been limited investigation into the initialization scheme used for vanilla LoRA. In this work we focus on the question of initialization. Through experimental verification and theoretical insights, we justify the use of a particular initialization choice over the *a priori* equally natural alternative.

- 本文的研究聚焦

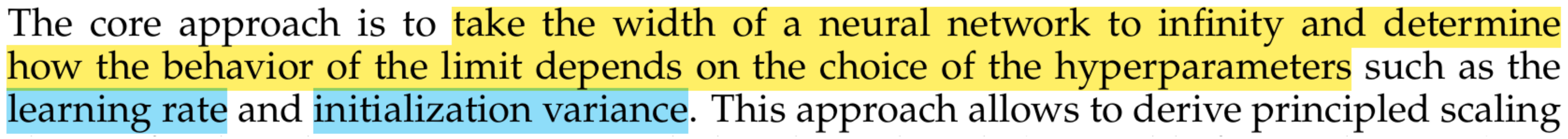
  看起来同样自然的两个初始化选择

  通过实验证实一个比另一个好

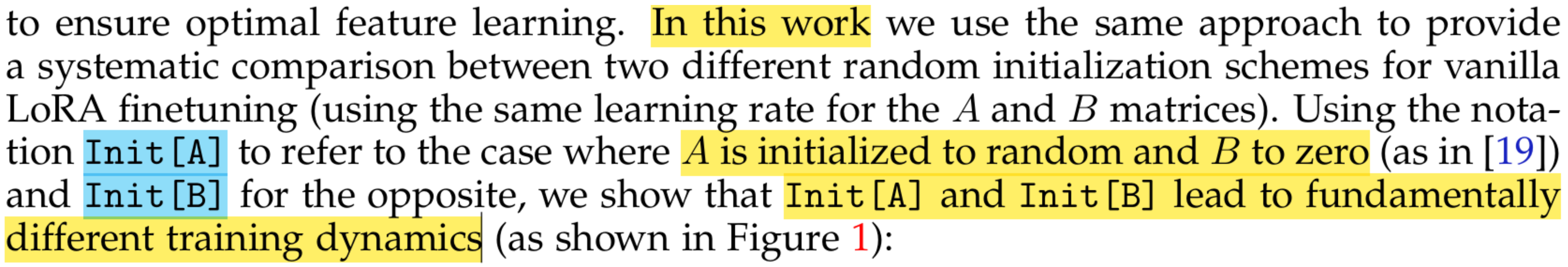
本文贡献


Contributions. In this paper, we study the impact of different random initialization schemes for LoRA adapters through a theory of large width for neural networks. There

- 神经网络大宽度理论


The core approach is to take the width of a neural network to infinity and determine how the behavior of the limit depends on the choice of the hyperparameters such as the learning rate and initialization variance. This approach allows to derive principled scaling

- 通过无穷宽的理想情况，找到超参数的理论最优设置


to ensure optimal feature learning. In this work we use the same approach to provide a systematic comparison between two different random initialization schemes for vanilla LoRA finetuning (using the same learning rate for the A and B matrices). Using the notation `Init[A]` to refer to the case where A is initialized to random and B to zero (as in [19]) and `Init[B]` for the opposite, we show that `Init[A]` and `Init[B]` lead to fundamentally different training dynamics (as shown in Figure 1):

- 本文结论： 两种初始化导致 “根本不同的训练动态”

本文结论

结论 1：Init [A] 比 Init [B]“允许使用更大的学习率”

1. `Init[A]` allows the use of larger learning rates compared to `Init[B]`

Init [A] 存在 “内部不稳定性”，但能提升特征学习效率，且存在 “特征学习 - 稳定性权衡”

2. `Init[A]` can lead to a form of ‘internal instability’ where the features Az (for some input z) are large but LoRA output BAz is small. This form of instability allows more efficient feature learning. We identify a *feature learning / stability tradeoff* in this case and support it with empirical results.

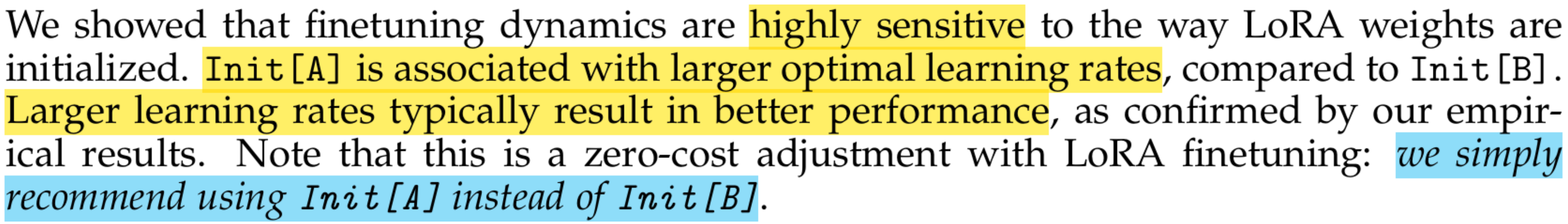
Init [B] 无稳定性问题，但训练次优，因 B 矩阵 “训练不足”

3. `Init[B]` does not cause any instabilities but training is suboptimal in this case (matrix B is undertrained).

Init [A] 性能通常优于 Init [B]

4. Empirical results confirm the theory and show that `Init[A]` generally leads to better performance than `Init[B]`.

本文结论



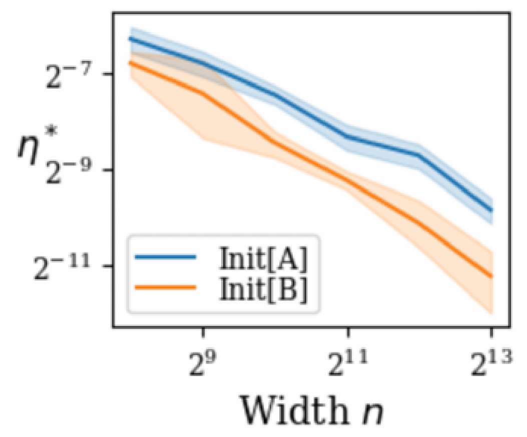
We showed that finetuning dynamics are highly sensitive to the way LoRA weights are initialized. `Init[A]` is associated with larger optimal learning rates, compared to `Init[B]`. Larger learning rates typically result in better performance, as confirmed by our empirical results. Note that this is a zero-cost adjustment with LoRA finetuning: *we simply recommend using `Init[A]` instead of `Init[B]`*.

- 微调动态对 LoRA 权重的初始化方式高度敏感

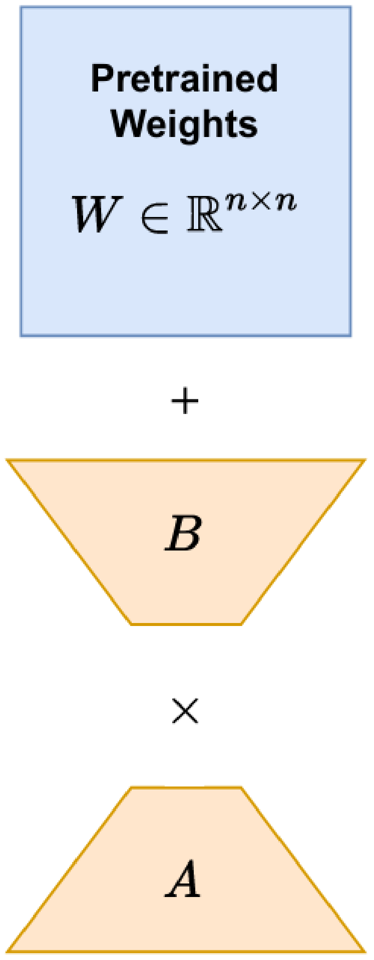
两种方案看似 “只是 A 和 B 的初始化顺序相反”，但在大宽度模型中，因学习率上限、特征更新效率的差异，会导致完全不同的训练动态

- Init [A] 对应的最优学习率更大
- 虽有 “内部不稳定性”，但这种不稳定性是良性的，不会导致输出爆炸，反而能提升特征学习效率

结论可视化

Init	**Init[A]:** $A \sim \mathcal{N}(0, 1/n)$ $B = 0$	**Init[B]:** $A = 0$ $B \sim \mathcal{N}(0, 1/r)$
Optimal Learning Rate	$\eta^*_{Init[A]} > \eta^*_{Init[B]}$ 	Init [A] 能支持的 “最优学习率” 远大于 Init [B]
Finetuning Dynamics	***More Efficient Feature Learning with some instability*** 特征学习更高效，但存在一定不稳定性	***Stability with suboptimal Feature Learning*** 更稳定，但特征学习次优

$\longrightarrow$ **Init[A] (generally) leads to better performance**



总结

- 初始时AB=0，模型输出完全由预训练权重决定
  以此来保证初始训练稳定
- 若A和B都全零初始化，会导致梯度消失，初始训练不稳定
- 若A和B都随机初始化，会引入初始噪声，训练难以收敛
- 其他，视具体任务来选择初始化方法，并做出解释即可