# 走近LoRA微调（1）
# LoRA这样节省显存和计算量

东川路第一可爱猫猫虫

# 主要内容

- 预训练-微调范式
- LoRA微调如何节省显存
- LoRA微调如何节省计算量
  - 神经网络如何更新参数

# LoRA: Low-Rank Adaptation of Large Language Models

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Lo**w-**R**ank **A**daptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*.

# 全量微调的改进

- h=Wx 其中W为权重矩阵，也就是参数矩阵
  在训练过程中每次需要更新整个W
  成本高
- 新发现
- LoRA的研究人员们借鉴了这个发现
  冻结预训练权重（记作$W_0$）
  h= $W_0$x+ABx
  对参数的增量做低秩分解
  $W_0$:n×m，A:n×r，B:r×m

# from $W_0$:n×m to A:n×r, B:r×m

- 全量微调的参数量

  nm
- LoRA微调的参数量

  nr+rm=r(n+m)
- r(n+m)≪nm
- 原因

  r可以取得很小

  8，4甚至1

# r很小，LoRA的效果

| | Weight Type | $r=1$ | $r=2$ | $r=4$ | $r=8$ | $r=64$ |
|---|---|---|---|---|---|---|
| WikiSQL($\pm 0.5\%$) | $W_q$ | 68.8 | 69.6 | 70.5 | 70.4 | 70.0 |
| | $W_q, W_v$ | 73.4 | 73.3 | 73.7 | 73.8 | 73.5 |
| | $W_q, W_k, W_v, W_o$ | 74.1 | 73.7 | 74.0 | 74.0 | 73.9 |
| MultiNLI ($\pm 0.1\%$) | $W_q$ | 90.7 | 90.9 | 91.1 | 90.7 | 90.7 |
| | $W_q, W_v$ | 91.3 | 91.4 | 91.3 | 91.6 | 91.4 |
| | $W_q, W_k, W_v, W_o$ | 91.2 | 91.7 | 91.7 | 91.5 | 91.4 |

- 我们发现

  一味增大r并不会产生更好的效果

# LoRA节省显存

- 显存主要来自参数和激活值
  - 优化器状态，模型的梯度
  - 与参数量正相关的
- LoRA微调如何节省计算量？
  - 通过节省反向传播时候的梯度计算量来完成的
- 什么是反向传播？
  - 什么是梯度？
  - 究竟怎么计算？
  - LoRA又是如何节省计算量的？

# 神经网络如何更新参数

- 前馈神经网络FFN
- 多层感知器MLP

  输入层 隐藏层 输出层
- 更新参数的具体过程

  前向传播

  损失函数 如$\frac{1}{2}\left(y_{pred}-y_{true}\right)^2$

  反向传播 求梯度

  更新参数 新参数=旧参数-学习率×梯度

  重复上述过程，不断更新

# LoRA如何反向传播求梯度

- 原始权重$W_0$n×m　$W_{n\times m} = W_0 + A_{n\times r}B_{r\times m}$

  损失函数L=L(Y)　输入$X_{b\times n}$　输出$Y_{b\times m} = X(W_0 + AB)$
- 我们来看LoRA的两种等效实现

  分别计算$\frac{\partial L}{\partial A}, \frac{\partial L}{\partial B}$, 比较两种实现的计算复杂度
- 第一种实现

  $Y = XW = X(W_0 + AB)$
- 第二种实现

  $Y = XW_0 + XAB = XW_0 + ZB$，其中Z=XZ为中间输出

# 第一种实现 $Y = XW = X(W_0 + AB)$

- 求矩阵偏导数

  凑矩阵形状

- 标量函数对矩阵求导

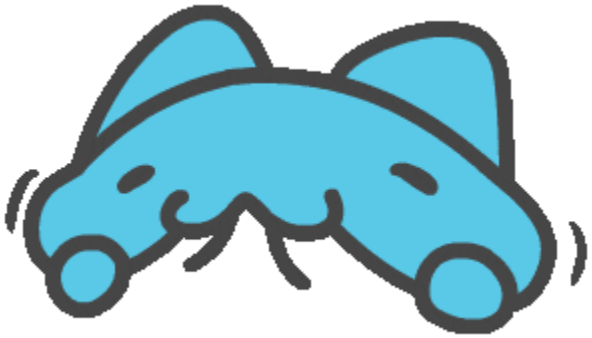
  求得矩阵的形状与原矩阵一致

- $\frac{\partial L}{\partial A} = \frac{\partial L}{\partial \mathrm{W}} \frac{\partial \mathrm{W}}{\partial A}$

  $\frac{\partial L}{\partial A}$ n×r　　$\frac{\partial L}{\partial \mathrm{W}}$ n×m

  由此推出$\frac{\partial \mathrm{W}}{\partial A}$ 形状为 m×r

  $\frac{\partial L}{\partial A} = \frac{\partial L}{\partial \mathrm{W}} B^T$



# 再来一次

- $Y = XW = X(W_0 + AB)$
- $\frac{\partial L}{\partial \mathrm{W}} = \frac{\partial L}{\partial \mathrm{Y}} \frac{\partial \mathrm{Y}}{\partial \mathrm{W}}$

  $\frac{\partial L}{\partial \mathrm{W}}$ n×m　$\frac{\partial L}{\partial \mathrm{Y}}$ b×m

  由此推出$\frac{\partial \mathrm{Y}}{\partial \mathrm{W}}$形状为n×b，并且要左乘在 $\frac{\partial L}{\partial \mathrm{Y}}$ 上

  $\frac{\partial L}{\partial \mathrm{W}} = X^T \frac{\partial L}{\partial \mathrm{Y}}$
- 因此 $\frac{\partial L}{\partial A} = (X^T \frac{\partial L}{\partial \mathrm{Y}})\mathrm{B}^T$
- 同理$\frac{\partial L}{\partial B} = A^T \left( X^T \frac{\partial L}{\partial \mathrm{Y}} \right)$

# 第一种实现 $Y = XW = X(W_0 + AB)$

- $\frac{\partial L}{\partial A} = (X^T \frac{\partial L}{\partial \mathrm{Y}})\mathrm{B}^T \quad \frac{\partial L}{\partial B} = A^T \left(X^T \frac{\partial L}{\partial \mathrm{Y}}\right)$
- 后果

  要算完整梯度$\frac{\partial L}{\partial \mathrm{W}}$，才能算$\frac{\partial L}{\partial A}, \frac{\partial L}{\partial B}$

  算的多了

  又慢又费显存

# 第二种实现$Y = XW_0 + XAB = XW_0 + ZB$

- $\frac{\partial L}{\partial A} = X^T \frac{\partial L}{\partial \mathrm{Z}} = X^T (\frac{\partial L}{\partial \mathrm{Y}} \mathrm{B}^T)$
- $\frac{\partial L}{\partial B} = Z^T \frac{\partial L}{\partial \mathrm{Y}} = (XA)^T \frac{\partial L}{\partial \mathrm{Y}}$
- 只需要对Y和Z求偏导

  不仅节省了显存，还节省了计算量

# LoRA的独特优点

- 很多其他参数高效的微调方法

  降低了显存需求，但没有节省计算量

- Prefix Tuning

  在输入层插入可训练的提示向量，主模型参数冻结

  用极少量可训练参数替代全量模型参数的更新

- Adapter

  在 Transformer 层插入小型适配器模块

  显存节省：适配器参数极少

  添加了Adapter后，模型整体的层数变深，会增加训练速度和推理速度