BERT:

基础知识：

BERT（Bidirectional Encoder Representations from Transformers）是谷歌提出，作为一个Word2Vec的替代者，其在NLP领域的11个方向大幅刷新了精度，可以说是近年来自残差网络最优突破性的一项技术了。论文的主要特点以下几点：

1. 使用了双向Transformer作为算法的主要框架，之前的模型是从左向右输入一个文本序列，或者将left-to-right 和 right-to-left 的训练结合起来，实验的结果表明，双向训练的语言模型对语境的理解会比单向的语言模型更深刻；
2. 使用了Mask Language Model(MLM)和 Next Sentence Prediction(NSP) 的多任务训练目标；
3. 使用更强大的机器训练更大规模的数据，使BERT的结果达到了全新的高度，并且Google开源了BERT模型，用户可以直接使用BERT作为Word2Vec的转换矩阵并高效的将其应用到自己的任务中。

BERT **只利用了 Transformer 的 encoder 部分。因为BERT 的目标是生成语言模型，所以只需要 encoder 机制。**

BERT有两个任务：

1. Masked LM (MLM) : 在将单词序列输入给 BERT 之前，每个序列中有 15% 的单词被[MASK] token替换。 然后模型尝试基于序列中其他未被 mask 的单词的上下文来预测被掩盖的原单词。在BERT的实验中，15%的WordPiece Token会被随机Mask掉。在训练模型时，一个句子会被多次喂到模型中用于参数学习，但是Google并没有在每次都mask掉这些单词，而是在确定要Mask掉的单词之后，80%的概率会直接替换为[Mask]，10%的概率将其替换为其它任意单词，10%的概率会保留原始Token。
    1. **80% 的 tokens 会被替换为 [MASK] token**：**是 Masked LM 中的主要部分，可以在不泄露 label 的情况下融合真双向语义信息**；
    2. **10% 的 tokens 会称替换为随机的 token** ： 因为需要在最后一层随机替换的这个 token 位去预测它真实的词，而模型并不知道这个 token 位是被随机替换的，就迫使模型尽量在每一个词上都学习到一个 全局语境下的表征，因而也能够让 BERT 获得更好的语境相关的词向量（**这正是解决一词多义的最重要特性**）；
    3. **10% 的 tokens 会保持不变但需要被预测** ： 这样能够给模型一定的 bias ，相当于是额外的奖励，将模型对于词的表征能够拉向词的 真实表征
2. Next Sentence Prediction (NSP) : 在 BERT 的训练过程中，模型接收成对的句子作为输入，并且预测其中第二个句子是否在原始文档中也是后续句子。
    1. **在训练期间，50% 的输入对在原始文档中是前后关系，另外 50% 中是从语料库中随机组成的，并且是与第一句断开的。**
    2. **在第一个句子的开头插入 `[CLS]` 标记，表示该特征用于分类模型，对非分类模型，该符号可以省去，在每个句子的末尾插入 `[SEP]` 标记，表示分句符号，用于断开输入语料中的两个句子。**

BERT的输入的编码向量（长度是512）是3个嵌入特征的单位和，这三个词嵌入特征是：

1. **位置嵌入（Position Embedding）**：位置嵌入是指将单词的位置信息编码成特征向量，位置嵌入是向模型中引入单词位置关系的至关重要的一环；
2. **WordPiece 嵌入**：WordPiece是指将单词划分成一组有限的公共子词单元，能在单词的有效性和字符的灵活性之间取得一个折中的平衡。例如上图的示例中‘playing’被拆分成了‘play’和‘ing’；

3. **分割嵌入（Segment Embedding）**：用于区分两个句子，例如B是否是A的下文（对话场景，问答场景等）。对于句子对，第一个句子的特征值是0，第二个句子的特征值是1。

Embedding：

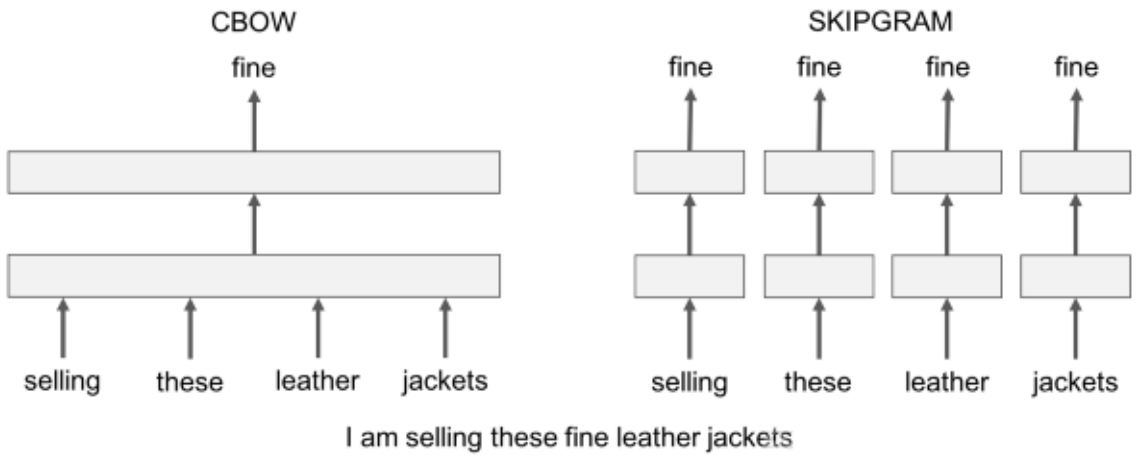
在传统的NLP中，将单词视为离散符号，然后可以用one-hot向量表示。向量的维度是整个词汇表中单词的数量。单词作为离散符号的问题在于，对于one-hot向量来说，没有自然的相似性概念。

因此，另一种方法是学习在向量本身中编码相似性。核心思想是一个词的含义是由经常出现在其旁边的单词给出的。

文本嵌入是字符串的实值向量表示。为每个单词建立一个密集的向量，选择它以便类似于类似上下文中出现的单词的向量。

在词嵌入中最流行的应该是Word2vec，它是由谷歌（Mikolov）开发的模型，其中固定词汇表中的每个词都由一个向量表示。然后，通过文本中的每个位置`t`，其中有一个中心词`c`和上下文词`o`。

Word2vec有两个变体：



1. `Skip-Gram`：考虑一个包含`k`个连续项的上下文窗口。然后，跳过其中一个单词，尝试学习一个神经网络，该网络可以获得除跳过的所有术语外的所有术语，并预测跳过的术语。 因此，如果两个单词在大语料库中反复共享相似的上下文，那么这些术语的嵌入向量将具有相似的向量。
2. `Continuous Bag of Words`：在一个大的语料库中获取大量的句子，每当看到一个词，就会联想到周围的词。然后，将上下文单词输入到神经网络，并预测该上下文中心的单词。当有数千个这样的上下文单词和中心单词时，就有了一个神经网络数据集的实例。训练神经网络，最后编码的隐藏层输出表示一个特定的词嵌入。当通过大量的句子进行训练时，类似上下文中的单词会得到相似的向量。

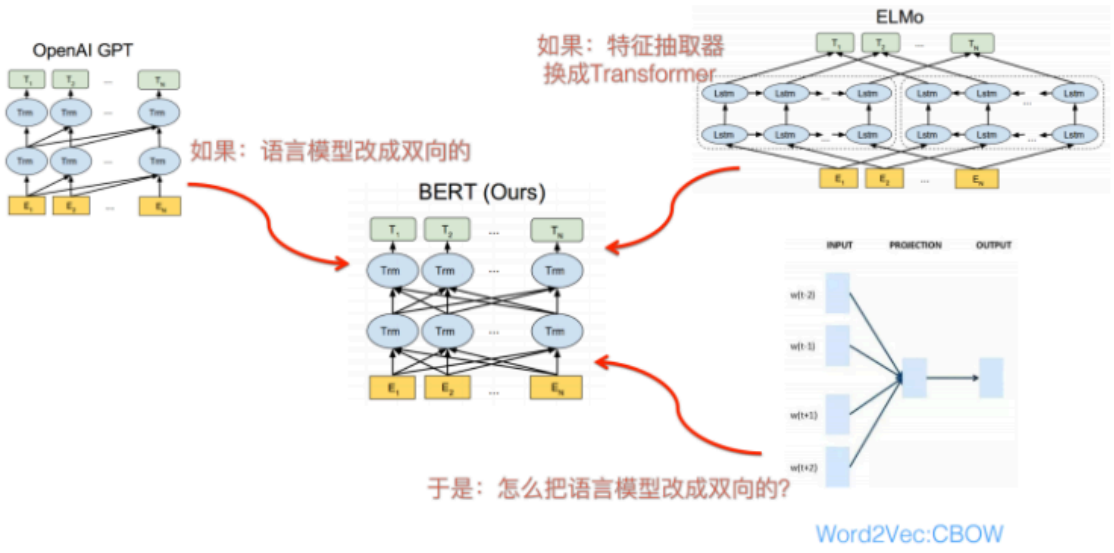
> **Note**
>
> **对`Skip-Gram`和`CBOW`的一个吐槽就是它们都是基于窗口的模型，这意味着语料库的共现统计不能被有效使用，导致次优的嵌入（suboptimal embeddings）。**

BERT OpenAI-GPT ELMo架构之间的差异：

- `BERT`使用双向Encoder，模型的表示在所有层中，共同依赖于左右两侧的上下文
- `OpenAI GPT`使用单向Encoder，利用了 Transformer 的编码器作为语言模型进行预训练的，之后特定的自然语言处理任务在其基础上进行微调即可。

- `ELMo` 使用独立训练的从左到右和从右到左LSTM级联来生成下游任务的特征。是一种双层双向的LSTM 结构，其训练的语言模型可以学习到句子左右两边的上下文信息，但此处所谓的上下文信息并不是真正意义上的上下文。

三种模型中只有BERT表征基于所有层左右两侧语境。



Word2Vec中为什么使用负采样?

负采样是另一种用来提高Word2Vec效率的方法，它是基于这样的观察：训练一个神经网络意味着使用一个训练样本就要稍微调整一下神经网络中所有的权重，这样才能够确保预测训练样本更加精确，**如果能设计一种方法每次只更新一部分权重，那么计算复杂度将大大降低。**

将以上观察引入Word2Vec就是：当通过（"fox", "quick"）词对来训练神经网络时，回想起这个神经网络的"标签"或者是"正确的输出"是一个one-hot向量。也就是说，对于神经网络中对应于"quick"这个单词的神经元对应为1，而其他上千个的输出神经元则对应为0。

使用负采样，通过随机选择一个较少数目（比如说5个）的"负"样本来更新对应的权重。(在这个条件下，"负"单词就是希望神经网络输出为0的神经元对应的单词）。并且仍然为"正"单词更新对应的权重（也就是当前样本下"quick"对应的神经元仍然输出为1）。

负采样这个点引入word2vec非常巧妙，两个作用：

1. **加速了模型计算**
2. **保证了模型训练的效果**，其一 模型每次只需要更新采样的词的权重，不用更新所有的权重，那样会很慢，其二 中**心词其实只跟它周围的词有关系，位置离着很远的词没有关系，也没必要同时训练更新，作者这点非常聪明。**

word2vec 相比之前的 Word Embedding 方法好在什么地方?

无论是 `CBOW` 还是 `Skip-Gram`，本质还是要**基于word和context做文章**，即可以理解为模型在学习word和context的co-occurrence。

Word2vec训练方面采用的HSoftmax以及负采样确实可以认为是创新不大。但Word2vec流行的主要原因也不在于此。主要原因在于以下3点：

1. **极快的训练速度。**以前的语言模型优化的目标是MLE，只能说词向量是其副产品。Mikolov应该是第一个提出抛弃MLE和困惑度指标，就是要学习一个好的词嵌入。如果不追求MLE，模型就可以大幅简化，去除隐藏层。再利用HSoftmax以及负采样的加速方法，可以使得训练在小时级别完成。而原来的语言模型可能需要几周时间。

   （1）MLE（最大似然估计，Maximum Likelihood Estimation）

   - **定义**： MLE 是一种统计方法，目标是让模型预测的词概率分布尽可能接近真实数据分布。
   - **在语言模型中的应用**：
     - 模型通过最大化所有词的联合概率（或对数似然）来优化参数。例如，预测句子中每个词出现的概率，使模型更准确地生成符合真实数据的文本。
     - 公式简化表示为：

$$\max \sum_{t=1}^{T} \log P(w_t \mid w_1, w_2, \ldots, w_{t-1})$$

     - **目标**： 让模型对真实数据的预测概率最大化，即“让模型尽可能正确预测下一个词”。

   （2）困惑度（Perplexity）

   - **定义**： 困惑度是衡量语言模型质量的指标，表示模型对未知数据的预测不确定性。困惑度越低，模型越好。
   - **数学理解**：
     - 困惑度是交叉熵的指数形式，公式为：

$$\text{Perplexity} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T} \log P(w_t \mid w_1, w_2, \ldots, w_{t-1})\right)$$

   - **直观理解**： 如果困惑度是 100，意味着模型平均需要从 100 个词中选择正确的一个。

2. **一个很酷炫的man-woman=king-queen的示例。**这个示例使得人们发现词嵌入还可以这么玩，并促使词嵌入学习成为了一个研究方向，而不再仅仅是神经网络中的一些参数。
3. word2vec里有大量的tricks，比如噪声分布如何选？如何采样？如何负采样？等等。这些tricks虽然摆不上台面，但是对于得到一个好的词向量至关重要。

NLP预训练发展史：从Word Embedding到BERT

图像预训练 → word embedding → word2vec → elmo → transformer → gpt → bert → GPT 234

常用参数更新方法:

梯度下降:

在一个方向上更新和调整模型的参数，来最小化损失函数。（沿着当前梯度方向更新）

- **基本思想**：每次迭代使用整个训练集的梯度更新参数，沿负梯度方向最小化损失函数。
- **公式**：

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta_t)$$

随机梯度下降（Stochastic gradient descent，SGD）

对每个训练样本进行参数更新，每次执行都进行一次更新，且执行速度更快。
(每次只用一个样本更新)

- **基本思想**：每次迭代仅使用一个随机样本的梯度更新参数，加速训练但波动较大。
- **公式**：

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J_i(\theta_t)$$

小批量梯度下降：

为了避免SGD和标准梯度下降中存在的问题，一个改进方法为**小批量梯度下降**（Mini Batch Gradient Descent），因为对每个批次中的n个训练样本，这种方法只执行一次更新。 **（每次用一小批样本更新）**

- **基本思想**：每次迭代使用一小批样本的平均梯度更新参数，平衡速度与稳定性。
- **公式**：

$$\theta_{t+1} = \theta_t - \frac{\eta}{B} \sum_{i=1}^{B} \nabla J_i(\theta_t)$$

使用小批量梯度下降的优点是：

1. 可以减少参数更新的波动，最终得到效果更好和更稳定的收敛。
2. 还可以使用最新的深层学习库中通用的矩阵优化方法，使计算小批量数据的梯度更加高效。
3. 通常来说，小批量样本的大小范围是从50到256，可以根据实际问题而有所不同。
4. 在训练神经网络时，通常都会选择小批量梯度下降算法。

动量：

SGD方法中的高方差振荡使得网络很难稳定收敛，所以有研究者提出了一种称为**动量（Momentum）的技术**，通过优化相关方向的训练和弱化无关方向的振荡，来加速SGD训练。

- **基本思想**：引入动量项（历史梯度的加权平均），加速收敛并减少震荡。
- **公式**：

$$v_{t+1} = \gamma v_t + \eta \cdot \nabla J(\theta_t)$$
$$\theta_{t+1} = \theta_t - v_{t+1}$$

Nesterov梯度加速法（NAG）

通过使网络更新与误差函数的斜率相适应，并依次加速SGD，也可根据每个参数的重要性来调整和更新对应参数，以执行更大或更小的更新幅度。

- **基本思想**：先“预测”下一步位置，再计算梯度，避免走偏，提升稳定性。
- **公式**：

$$\theta_{\text{lookahead}} = \theta_t - \gamma v_t$$
$$v_{t+1} = \gamma v_t + \eta \cdot \nabla J(\theta_{\text{lookahead}})$$
$$\theta_{t+1} = \theta_t - v_{t+1}$$

AdaDelta（Adaptive Learning Rate Method）

是AdaGrad的延伸方法，它倾向于解决其学习率衰减的问题。Adadelta不是累积所有之前的平方梯度，而是将累积之前梯度的窗口限制到某个固定大小w。

- **基本思想**：解决 Adagrad 学习率持续下降的问题，使用滑动窗口的梯度平方均值。
- **公式**：

$$E[g^2]_t = \rho \cdot E[g^2]_{t-1} + (1-\rho) \cdot (\nabla J(\theta_t))^2$$
$$\Delta\theta_t = \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot \nabla J(\theta_t)$$
$$\theta_{t+1} = \theta_t - \Delta\theta_t$$

Adam（Adaptive Moment Estimation）

Adam算法即自适应时刻估计方法（Adaptive Moment Estimation），能计算每个参数的自适应学习率。这个方法不仅存储了AdaDelta先前平方梯度的指数衰减平均值，而且保持了先前梯度M(t)的指数衰减平均值，这一点与动量类似。

- **基本思想**：结合动量与 RMSProp，自动调整学习率，计算一阶和二阶矩估计。
- **公式**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \cdot \nabla J(\theta_t)$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) \cdot (\nabla J(\theta_t))^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Adagrad（Adaptive Gradient）

Adagrad方法是通过参数来调整合适的学习率η，对稀疏参数进行大幅更新和对频繁参数进行小幅更新。因此，Adagrad方法非常适合处理稀疏数据。

- **基本思想**：自适应调整学习率，对稀疏特征分配更大步长，频繁特征分配更小步长。
- **公式**：

$$G_t = G_{t-1} + (\nabla J(\theta_t))^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla J(\theta_t)$$

对比总结

优化器	核心思想	公式关键点
SGD	单样本更新	$\theta_{t+1} = \theta_t - \eta \cdot \nabla J_i(\theta_t)$
动量	历史方向加速	$v_{t+1} = \gamma v_t + \eta \cdot \nabla J(\theta_t)$
NAG	预测方向修正	$\theta_{\text{lookahead}} = \theta_t - \gamma v_t$
Adagrad	稀疏特征适配	$G_t = G_{t-1} + (\nabla J(\theta_t))^2$
AdaDelta	滑动窗口均值	$E[g^2]_t = \rho \cdot E[g^2]_{t-1} + (1-\rho) \cdot (\nabla J(\theta_t))^2$

优化器	核心思想	公式关键点
Adam	自适应学习率	$m_t = \beta_1 m_{t-1} + (1-\beta_1) \cdot \nabla J(\theta_t)$

Adagrad vs AdaDelta vs Adam 对比

方法	优点	缺点	适用场景
Adagrad	适合稀疏数据	学习率持续下降	NLP、稀疏特征
AdaDelta	学习率不会无限下降	参数更新较复杂	大规模非凸优化
Adam	自适应学习率，收敛快	参数多	各类深度学习任务

总结：如何选择优化器?

场景	推荐优化器
简单、快速实现	SGD + 动量
稀疏数据（如 NLP）	Adagrad、Adam
非凸优化（如 CNN）	Adam、RMSProp
稳定收敛	Adam、NAG

LLM为什么Decoder only架构?

LLM 是 “Large Language Model” 的简写，目前一般指百亿参数以上的语言模型， 主要面向文本生成任务。跟小尺度模型（10亿或以内量级）的“百花齐放”不同，目前LLM的一个现状是Decoder-only架构的研究居多，像OpenAI一直坚持Decoder-only的GPT系列就不说了，即便是Google这样的并非全部押注在Decoder-only的公司，也确实投入了不少的精力去研究Decoder-only的模型，如PaLM就是其中之一。那么，为什么Decoder-only架构会成为LLM的主流选择呢?

Transformer 模型一开始是用来做 seq2seq 任务的，所以它包含 Encoder 和 Decoder 两个部分；他们两者的区别主要是，**Encoder 在抽取序列中某一个词的特征时能够看到整个序列中所有的信息，即上文和下文同时看到**；而 **Decoder 中因为有 mask 机制的存在，使得它在编码某一个词的特征时只能看到自身和它之前的文本信息。**

首先概述几种主要的架构:

- 以BERT为代表的**encoder-only**
- 以T5和BART为代表的**encoder-decoder**
- 以GPT为代表的**decoder-only**
- 以UNILM9为代表的PrefixLM**(相比于GPT只改了attention mask，前缀部分是双向，后面要生成的部分是单向的causal mask)**
  局部双向+全局生成： 前缀的双向注意力增强理解，后续生成保持自回归流畅性

	Encoder注意力	Decoder注意力	是否共享参数
GPT	单向	单向	是
UniLM	双向	单向	是
T5	双向	单向	否

然后说明要比较的对象: 首先**淘汰掉BERT这种encoder-only，因为它用masked language modeling预训练，不擅长做生成任务**，做NLUQ一般也需要有监督的下游数据微调: **相比之下decoder-only的模型用next token prediction预训练，兼顾理解和生成，在各种下游任务上的zero-shot和few-shot泛化性能都很好。**我们需要讨论的是，为啥引入了一部分双向attention的encoder-decoder和Prefix-LM没有被大部分大模型工作采用? (它们也能兼顾理解和生成，泛化性能也不错)

Encoder的低秩问题

LLM之所以主要都用Decoder-only架构，除了训练效率和工程实现上的优势外，在理论上是因为**Encoder的双向注意力会存在低秩问题，这可能会削弱模型表达能力**，就生成任务而言，引入双向注意力并无实质好处。而Encoder-Decoder架构之所以能够在某些场景下表现更好，大概只是因为它多了一倍参数。所以，在同等参数量、同等推理成本下，Decoder-only架构就是最优选择了。

低秩问题的定义与检测

- **数学定义**:
  - 若模型的表示矩阵 $H \in \mathbb{R}^{n \times d}$（$n$为序列长度，$d$为维度）的秩 $\text{rank}(H) \ll \min(n, d)$，则称其存在**低秩问题。**
  - **直观理解**：模型学到的表示集中在低维子空间，丢失了高维空间中潜在的语义信息。
- **检测方法**:
  - **奇异值分解（SVD）**：对BERT的表示进行SVD，发现前10个奇异值占比超过90%，表明信息集中在少数方向。
  - **可视化分析**：将词向量投影到二维空间，发现语义相似词（如“苹果”和“水果”）在表示空间中距离过近，难以区分。

低秩问题在Encoder-only模型中的表现

- **BERT的表示压缩现象**:
  - **局部依赖**：MLM（Masked Language Modeling）目标仅需预测少量被遮蔽的词，模型可能过度依赖局部共现模式（如“苹果”常被预测为“公司”而非“水果”）。
  - **注意力退化**：BERT的注意力头多聚焦于局部窗口（如相邻词），全局依赖较弱，导致表示压缩到低维流形。
  - **任务适配性差**：微调后表现优异，但zero-shot/few-shot泛化能力差，因表示中未保留足够的泛化信息。

低秩问题的成因分析

- **预训练目标的局限性**:
  - **MLM的局部性**：仅需预测少量mask token，模型可能忽略全局语义。
  - **负例不足**：MLM的负例是随机采样的非目标词，缺乏对语义区分的挑战性负例。
- **注意力机制的偏差**:
  - **双向注意力的稀疏性**：BERT的注意力权重分布可能集中在高频词（如“the”、“of”），削弱对低频词的表示能力。

    - **位置编码的局限性**：固定位置编码难以捕捉长距离依赖，导致表示压缩到局部区域。
- **参数冗余**：
    - 高维表示中存在冗余维度（如某些维度始终接近零），模型通过低秩近似即可完成任务，无需充分利用表达能力。

低秩问题的具体影响

影响类型	具体表现
语义区分度下降	相似语义的输入（如“苹果”和“水果”）在表示空间中距离过近，影响分类或检索任务。
微调依赖性增强	低秩表示缺乏任务无关的语义信息，需通过微调注入新知识（如领域特定特征）。
迁移学习受限	在分布外任务中表现差，因表示中未保留足够泛化能力（如从英语到法语的跨语言迁移）。
生成能力受限	编码器的低秩表示作为生成任务的输入时，解码器难以恢复丢失的语义细节，导致生成连贯性下降。

Decoder-only模型如何缓解低秩问题?

- **自回归生成的动态更新**：
    - 每步生成依赖完整历史（如GPT预测“下一个词”时需理解整个上下文），迫使模型学习更鲁棒的上下文表示。
- **负采样效应**：
    - 预测未来token时需区分多个负例（如区分“苹果”和“水果”），促进高秩表示的学习。
- **位置感知强化**：
    - 位置编码与自回归解码结合，强化顺序信息的表示能力（如区分“猫追狗”和“狗追猫”）。

总结：低秩问题的启示

- **模型设计**：
    - Encoder-only模型（如BERT）适合微调理解任务，但需警惕低秩导致的泛化能力下降。
    - Decoder-only模型（如GPT）通过自回归生成天然缓解低秩问题，更适合少样本和生成任务。

2.更好的Zero-Shot性能、更适合于大语料自监督学习

首先，对 encoder-decoder 与 decoder-only 的比较早已有之。先把目光放放到模型参数动辄100B之前的时代，看看小一点的模型参数量下、两个架构各有什么优势——Google Brain 和 HuggingFace联合发表的 What Language Model Architecture and Pretraining Objective Work Best for Zero-Shot Generalization? 曾经在5B的参数量级下对比了两者性能。
论文最主要的一个结论是：**decoder-only 模型在没有任何 tuning 数据的情况下、zero-shot 表现最好，而 encoder-decoder 则需要在一定量的标注数据上做 multitask finetuning 才能激发最佳性能。**
而目前的Large LM的训练范式还是在大规模语料上做自监督学习，很显然，Zero-Shot性能更好的decoder-only架构才能更好地利用这些无标注数据。此外，Instruct GPT在自监督学习外还引入了RLHF作辅助学习。RLHF本身也不需要人工提供任务特定的标注数据，仅需要在LLM生成的结果上作排序。虽然目前没有太多有关RLHF + encoder-decoder的相关实验，直觉上RLHF带来的提升可能还是不如multitask finetuning，毕竟前者本质只是ranking、引入监督信号没有后者强。

3.效率问题

decoder-only支持一直复用KV-Cache，对多轮对话更友好，因为每个Token的表示之和它之前的输入有关，而encoder-decoder和PrefixLM就难以做到。

总结：模型选择的核心权衡

维度	Decoder-only（GPT）	Encoder-Decoder（T5）	Encoder-only（BERT）
预训练目标	自回归生成（Next Token）	MLM + 自回归生成	MLM（双向理解）
生成能力	强（zero-shot/few-shot）	中等（需编码器-解码器协同）	弱（需额外解码器）
理解能力	通过上下文学习隐式理解	显式编码器提供强理解	强理解（微调后）
训练效率	高（单阶段自回归）	低（双阶段联合训练）	高（仅编码器训练）
低秩风险	无（自回归动态更新）	编码器可能低秩	高（MLM目标导致）
适用场景	生成主导、zero-shot任务	理解+生成平衡任务（如翻译）	理解任务（需微调）