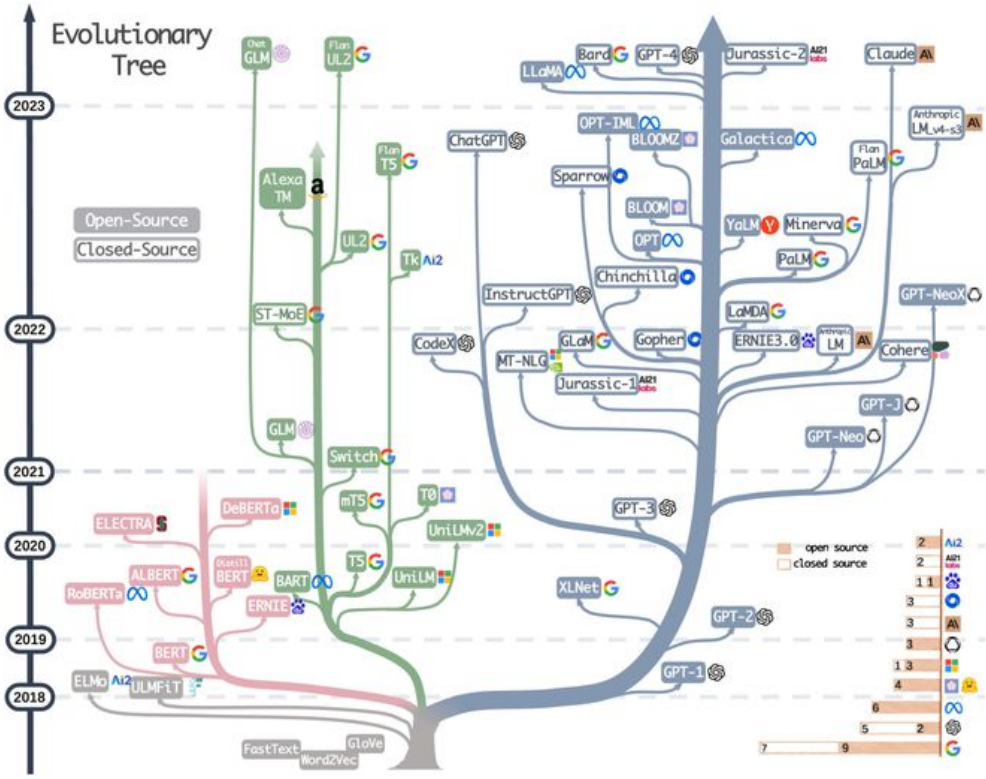
LLM背景知识介绍

学习目标

- 了解LLM背景的知识
- 掌握什么是语言模型

1 大语言模型 (LLM) 背景

大语言模型 (英文：Large Language Model，缩写LLM) 是一种人工智能模型, 旨在理解和生成人类语言. 大语言模型可以处理多种自然语言任务，如文本分类、问答、翻译、对话等等.



通常, 大语言模型 (LLM) 是指包含数千亿 (或更多) 参数的语言模型(目前定义参数量超过10B的模型为大语言模型)，这些参数是在大量文本数据上训练的，例如模型 GPT-3、ChatGPT、PaLM、BLOOM和LLaMA等.

截止23年，语言模型发展走过了三个阶段：

- **第一阶段**： 设计一系列的自监督训练目标（MLM、NSP等），设计新颖的模型架构（Transformer），遵循Pre-training和Fine-tuning范式。典型代表是BERT、GPT、XLNet等；
- **第二阶段**： 逐步扩大模型参数和训练语料规模，探索不同类型的架构。典型代表是BART、T5、GPT-3等；
- **第三阶段**： 走向AIGC（Artificial Intelligent Generated Content）时代，模型参数规模步入千万

亿，模型架构为自回归架构，大模型走向对话式、生成式、多模态时代，更加注重与人类交互进行对齐，实现可靠、安全、无毒的模型。典型代表是InstructionGPT、ChatGPT、Bard、GPT-4等。

---

2 语言模型 (Language Model, LM)

语言模型（Language Model）旨在建模词汇序列的生成概率，提升机器的语言智能水平，使机器能够模拟人类说话、写作的模式进行自动文本输出。

通俗理解: 用来计算一个句子的概率的模型，也就是判断一句话是否是人话的概率.

标准定义：对于某个句子序列, 如S = {W1, W2, W3, ..., Wn}, 语言模型就是计算该序列发生的概率, 即P(S). 如果给定的词序列符合语用习惯, 则给出高概率, 否则给出低概率.

举例说明：

- 假设我们要为中文创建一个语言模型，V表示词典，V={黑马、程序、员、来、学习}，W_i 属于V。语言模型描述：给定词典V, 能够计算出任意单词序列$S = W_1, W_2, W_3, \ldots, W_n$是一句话的概率$P(S)$, 其中$P >= 0$
- 那么如何计算一个句子的$P(S)$呢？最简单的方法就是计数，假设数据集中共有N个句子，我们可以统计一下数据集中$S = W_1, W_2, W_3, \ldots, W_n$每个句子出现的次数，如果假设为$n$，则$P(S) = \frac{n}{N}$. 那么可以想象一下，这个模型的预测能力几乎为0，一旦单词序列没在之前数据集中出现过，模型的输出概率就是0，显然相当不合理。
- 我们可以根据概率论中的链式法则，将P可以表示为：

$$P(S) = P(W_1, W_2, .., W_n) = P(W_1) * P(W_2|W_1) * \ldots * P(W_n|W_1, W_2, .., W_{n-1}) \quad (1)$$

如果能计算$P(W_n|W_1, W_2, \ldots W_{n-1})$，那么就能轻松得到$P(W_1, W_2, \ldots, W_n)$, 所以在某些文献中，我们也可以看到语言模型的另外一个定义：能够计算出$P(W_1, W_2, \ldots, W_n)$的模型就是语言模型。

从文本生成角度，也可以这样定义语言模型：给定一个短语（一个词组或者一句话），语言模型可以生成（预测）接下来的一个词。

基于语言模型技术的发展，可以将语言模型分为四种类型：

- 基于规则和统计的语言模型
- 神经语言模型
- 预训练语言模型
- 大语言模型

2.1 基于规则和统计的语言模型（N-gram）

由人工设计特征并使用统计方法对固定长度的文本窗口序列进行建模分析，这种建模方式也被称为N-gram语言模型。在上述例子中计算句子序列概率我们使用链式法则计算， 该方法存在两个缺陷：

- 参数空间过大：条件概率$P(W_n|W_1, W_2, \ldots . W_n)$的可能性太多，无法估算，也不一定有用
- 数据稀疏严重：许多词对的组合，在语料库中都没有出现，依据最大似然估计得到的概率为0

为了解决上述问题，引入马尔科夫假设：随意一个词出现的概率只与它前面出现的有限的一个或者几个词有关。

- 如果一个词的出现与它周围的词是独立的，那么我们就称之为unigram也就是一元语言模型.

$$P(S) = P(W_1) * P(W_2) * \ldots * P(W_n) \tag{2}$$

- 如果一个词的出现仅依赖于它前面出现的一个词，那么我们就称之为bigram.

$$P(S) = P(W_1) * P(W_2|W_1) * P(W_3|W_2) * \ldots * P(W_n|W_{n-1}) \tag{3}$$

- 如果一个词的出现仅依赖于它前面出现的两个词，那么我们就称之为trigram.

$$P(S) = P(W_1) * P(W_2|W_1) * P(W_3|W_2, W_1) * \ldots * P(W_n|W_{n-1}, W_{n-2}) \tag{4}$$

- 一般来说，N元模型就是假设当前词的出现概率只与它前面的N-1个词有关，而这些概率参数都是可以通过大规模语料库来计算，比如三元概率：

$$P(W_i|W_{i-1}, W_{i-2}) = Count(W_{i-2}W_{i-1}W_i)/Count(W_{i-2}W_{i-1}) \tag{5}$$

在实践中用的最多的就是bigram和trigram，接下来以bigram语言模型为例，理解其工作原理：

- 首先我们准备一个语料库（简单理解让模型学习的数据集），为了计算对应的二元模型的参数，即$P(W_i|W_{i-1})$，我们要先计数即$C(W_{i-1}, W_i)$，然后计数 $C(W_{i-1})$，再用除法可得到概率。
- $C(W_{i-1}, W_i)$ 计数结果如下：

	我	想	去	打	篮球	晚饭	食物	喝
我	4	800	2	10	0	0	0	20
想	1	20	600	2	3	6	1	10
去	1	3	4	690	7	0	2	100
打	10	0	1	0	20	2	2	0
篮球	0	0	0	0	0	0	0	0
晚饭	0	2	30	0	0	0	3	0
食物	0	0	3	0	0	0	0	0
喝	0	0	1	0	0	0	0	0

- $C(W_{i-1})$ 的计数结果如下:

我	想	去	打	篮球	晚饭	食物	喝
2100	900	2000	800	1000	120	300	260

- 那么bigram语言模型针对上述语料的参数计算结果如何实现？假如，我想计算$P(\text{想}|\text{我}) \approx 0.38$，计算过程如下显示：（其他参数计算过程类似）

$$P(\text{想}|\text{我}) = \frac{C(\text{我},\text{想})}{C(\text{我})} = \frac{800}{2100} \approx 0.38 \tag{6}$$

- 如果针对这个语料库的二元模型（bigram）建立好之后，就可以实现我们的目标计算。
- 计算一个句子的概率，举例如下：

$$P(\text{我想去打篮球}) = P(\text{想} \mid \text{我}) * P(\text{去} \mid \text{想}) * P(\text{打} \mid \text{去}) * P(\text{篮球} \mid \text{打}) = \frac{800}{2100} * \frac{600}{900} * \frac{690}{2000} * \frac{20}{800} \approx 0.0022 \quad (7)$$

- 预测一句话最可能出现的下一个词汇，比如：我想去打【mask】？思考：mask = 篮球 或者 mask = 晚饭。

$$P(\text{我想去打篮球}) = \approx 0.0022 \quad (8)$$

$$P(\text{我想去打晚饭}) = \approx 0.00022 \quad (9)$$

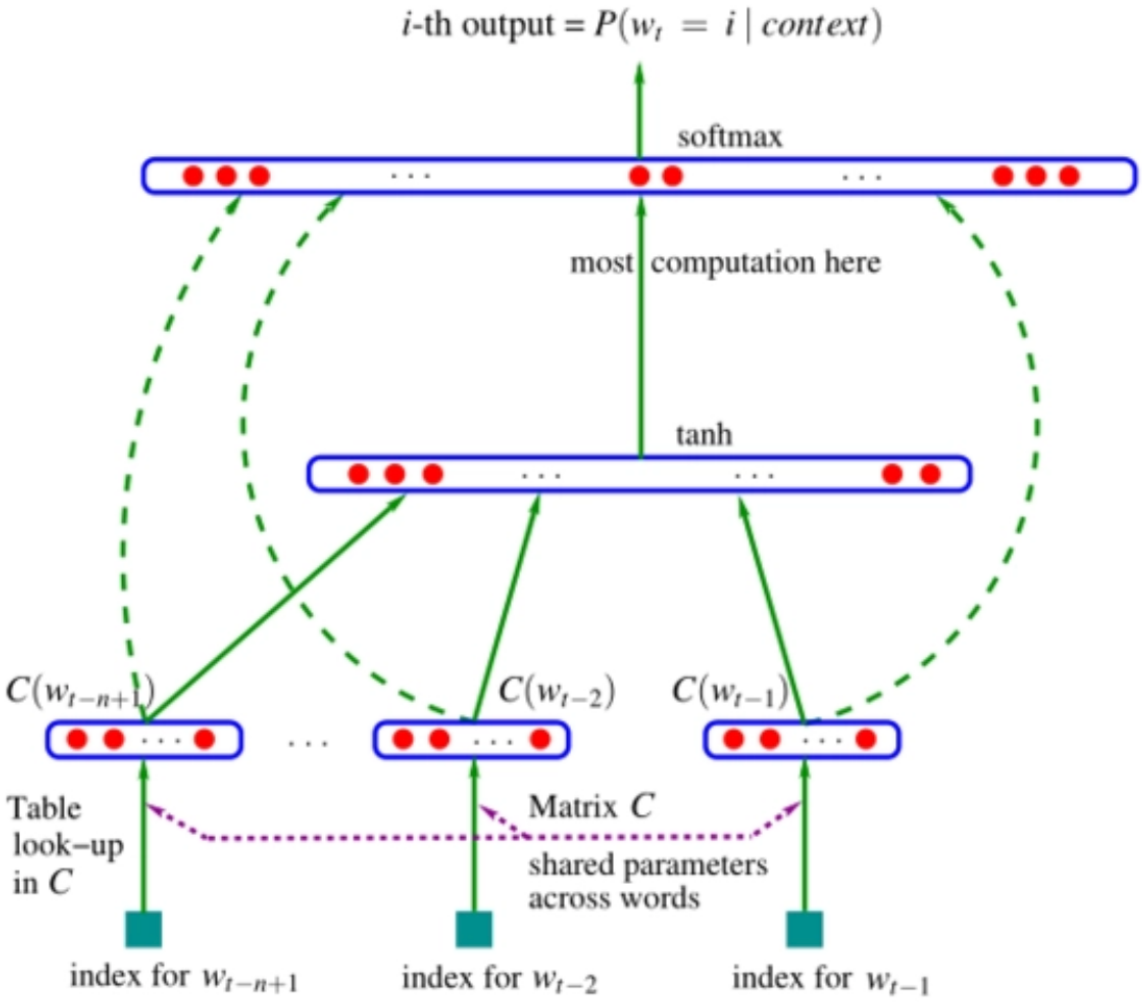
- 可以看出$P(\text{我想去打篮球}) > P(\text{我想去打晚饭})$，因此mask = 篮球，对比真实语境下，也符合人类习惯。

N-gram语言模型的特点：

- 优点：采用极大似然估计, 参数易训练; 完全包含了前n-1个词的全部信息; 可解释性强, 直观易理解。
- 缺点：缺乏长期以来，只能建模到前n-1个词; 随着n的增大，参数空间呈指数增长；数据稀疏，难免会出现OOV问题; 单纯的基于统计频次，泛化能力差.

2.2 神经网络语言模型

伴随着神经网络技术的发展，人们开始尝试使用神经网络来建立语言模型进而解决N-gram语言模型存在的问题。



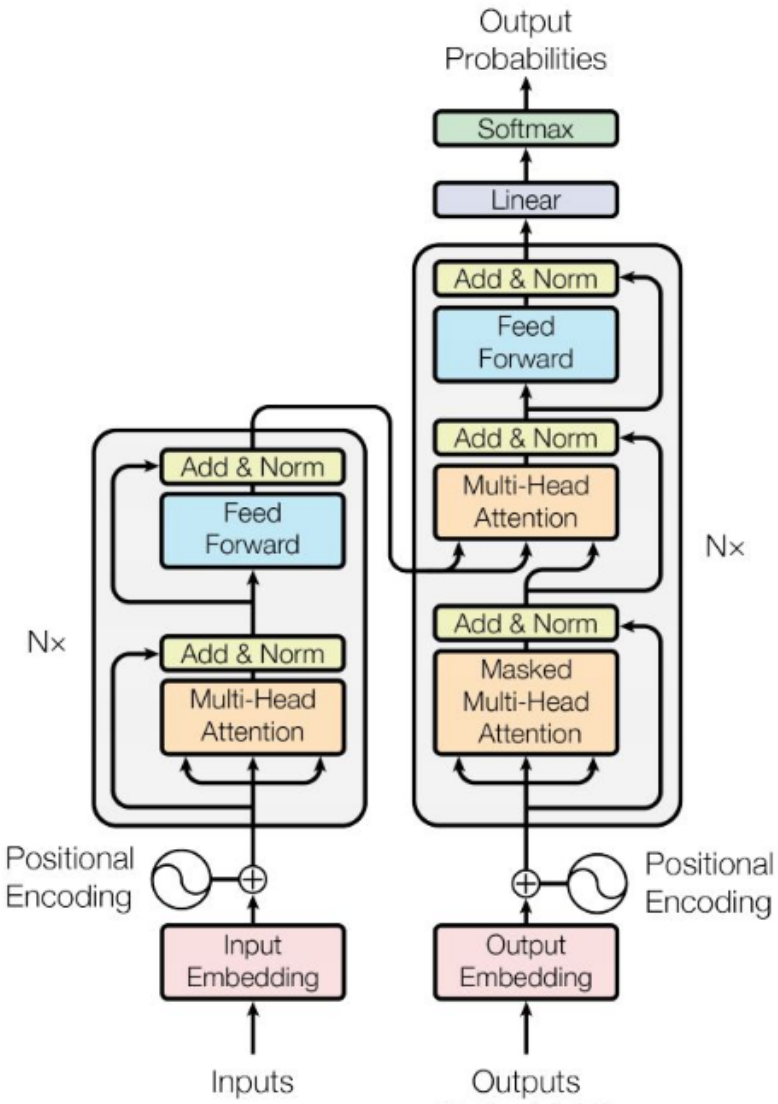
上图属于一个最基础的神经网络架构：

- 模型的输入：$w_{t-n+1},\ldots,w_{t-2},w_{t-1}$就是前n-1个词。现在需要根据这已知的n-1个词预测下一个词w_t。$C(w)$表示w所对应的词向量.
- 网络的第一层（输入层）是将$C(w_{t-n+1}),\ldots,C(w_{t-2}),C(w_{t-1})$这n-1个向量首尾拼接起来形成一个$(n-1)*m$大小的向量，记作$x$.
- 网络的第二层（隐藏层）就如同普通的神经网络，直接使用一个全连接层, 通过全连接层后再使用$tanh$这个激活函数进行处理。
- 网络的第三层（输出层）一共有V个节点 (V 代表语料的词汇)，本质上这个输出层也是一个全连接层。每个输出节点y_i表示下一个词语为 i 的未归一化log 概率。最后使用 softmax 激活函数将输出值y进行归一化。得到最大概率值，就是我们需要预测的结果。

神经网络特点：

- 优点：利用神经网络去建模当前词出现的概率与其前 n-1 个词之间的约束关系，很显然这种方式相比 n-gram 具有更好的泛化能力，只要词表征足够好。从而很大程度地降低了数据稀疏带来的问题。
- 缺点：对长序列的建模能力有限，可能会出现长距离遗忘以及训练时的梯度消失等问题，构建的模型难以进行稳定的长文本输出。

2.3 基于Transformer的预训练语言模型



Transformer模型由一些编码器和解码器层组成（见图），学习复杂语义信息的能力强，很多主流预训练模型在提取特征时都会选择Transformer结构，并产生了一系列的基于Transformer的预训练模型，包括GPT、BERT、T5等.这些模型能够从大量的通用文本数据中学习大量的语言表示，并将这些知识运用到下游任务中，获得了较好的效果.

预训练语言模型的使用方式：

- 预训练：预训练指建立基本的模型，先在一些比较基础的数据集、语料库上进行训练，然后按照具体任务训练，学习数据的普遍特征。
- 微调：微调指在具体的下游任务中使用预训练好的模型进行迁移学习，以获取更好的泛化效果。

预训练语言模型的特点：

- 优点：更强大的泛化能力，丰富的语义表示，可以有效防止过拟合。
- 缺点：计算资源需求大，可解释性差等

2.4 大语言模型

随着对预训练语言模型研究的开展，人们逐渐发现可能存在一种标度定律（Scaling Law），即随着预训练模型参数的指数级提升，其语言模型性能也会线性上升。2020年，OpenAI发布了参数量高达1750亿的GPT-3，首次展示了大语言模型的性能。

相较于此前的参数量较小的预训练语言模型，例如，3.3亿参数的Bert-large和17亿参数的GPT-2，GPT-3展现了在Few-shot语言任务能力上的飞跃，并具备了预训练语言模型不具备的一些能力。后续将这种现象称为能力涌现。例如，GPT-3能进行上下文学习，在不调整权重的情况下仅依据用户给出的任务示例完成后续任务。这种能力方面的飞跃引发研究界在大语言模型上的研究热潮，各大科技巨头纷纷推出参数量巨大的语言模型，例如，Meta公司1300亿参数量的LLaMA模型以及谷歌公司5400亿参数量的PaLM。国内如百度推出的文心一言ERNIE系列、清华大学团队推出的GLM系列，等等。

大语言模型的特点：

- 优点：像“人类”一样智能，具备了能与人类沟通聊天的能力，甚至具备了使用插件进行自动信息检索的能力
- 缺点：参数量大，算力要求高、生成部分有害的、有偏见的内容等等

---

3 语言模型的评估指标

3.1 BLEU

BLEU （双语评估替补）分数是评估一种语言翻译成另一种语言的文本质量的指标。它将“质量”的好坏定义为与人类翻译结果的一致性程度。

BLEU算法实际上就是在判断两个句子的相似程度. BLEU 的分数取值范围是 0～1，分数越接近1，说明翻译的质量越高。

BLEU有许多变种，根据`n-gram`可以划分成多种评价指标，常见的评价指标有BLEU-1、BLEU-2、BLEU-3、BLEU-4四种，其中`n-gram`指的是连续的单词个数为n，BLEU-1衡量的是单词级别的准确性，更高阶的BLEU可以衡量句子的流畅性.实践中，通常是取N=1~4，然后对进行加权平均。

下面举例说计算过程：

- 基本步骤：
  - 分别计算预测文本和目标文本的N-grams模型，然后统计其匹配的个数，计算匹配度:
  - 公式：预测文本中正确预测的 n−gram 的个数 /预测文本中所有n−gram 的个数
- 假设分别给出一个预测文本和目标文本如下：

```
预测文本：It is a nice day today
目标文本：today is a nice day
```

- 使用1-gram进行匹配

```
预测文本：{it, is, a, nice, day, today}
目标文本：{today, is, a, nice, day}
结果：
  其中{today, is, a, nice, day}匹配，所以匹配度为5/6
```

- 使用2-gram进行匹配

```
预测文本：{it is, is a, a nice, nice day, day today}
目标文本：{today is, is a, a nice, nice day}
结果：
  其中{is a, a nice, nice day}匹配，所以匹配度为3/5
```

- 使用3-gram进行匹配

```
预测文本：{it is a, is a nice, a nice day, nice day today}
目标文本：{today is a, is a nice, a nice day}

结果：
  其中{is a nice, a nice day}匹配，所以匹配度为2/4
```

- 使用4-gram进行匹配

```
预测文本：{it is a nice, is a nice day, a nice day today}
目标文本：{today is a nice, is a nice day}

结果：
  其中{is a nice day}匹配，所以匹配度为1/3
```

上述例子会出现一种极端情况，请看下面示例：

```
预测文本：the the the the
目标文本：The cat is standing on the ground
如果按照1-gram的方法进行匹配，则匹配度为1，显然是不合理的，所以计算某个词的出现次数进行改进
```

- 将计算某个词正确预测次数的方法改为计算**某个词在文本中出现的最小次数**,如下所示的公式：

$$count_k = min(c_k, s_k) \tag{10}$$

- 其中k表示在预测文本中出现的第k个词语, c_k则代表在预测文本中这个词语出现的次数，而s_k则代表在目标文本中这个词语出现的次数。

---

python代码实现：

```
# 第一步安装nltk的包-->pip install nltk
from nltk.translate.bleu_score import sentence_bleu
```

```
def cumulative_bleu(reference, candidate):

    bleu_1_gram = sentence_bleu(reference, candidate, weights=(1, 0, 0, 0))
    bleu_2_gram = sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0,
0))
    bleu_3_gram = sentence_bleu(reference, candidate, weights=(0.33, 0.33,
0.33, 0))
    bleu_4_gram = sentence_bleu(reference, candidate, weights=(0.25, 0.25,
0.25, 0.25))

    # print('bleu 1-gram: %f' % bleu_1_gram)
    # print('bleu 2-gram: %f' % bleu_2_gram)
    # print('bleu 3-gram: %f' % bleu_3_gram)
    # print('bleu 4-gram: %f' % bleu_4_gram)

    return bleu_1_gram, bleu_2_gram, bleu_3_gram, bleu_4_gram

# 预测文本
candidate_text = ["This", "is",  "some",  "generated", "text"]

# 目标文本列表
reference_texts = [["This", "is",  "a",  "reference", "text"],
                   ["This", "is",  "another", "reference", "text"]]

# 计算 Bleu 指标
c_bleu = cumulative_bleu(reference_texts, candidate_text)

# 打印结果

print("The Bleu score is:", c_bleu)
# The Bleu score is: (0.6, 0.387, 1.5949011744633917e-102, 9.283142785759642e-
155)
```

3.2 ROUGE

ROUGE指标是在机器翻译、自动摘要、问答生成等领域常见的评估指标。ROUGE通过将模型生成的摘要或者回答与参考答案（一般是人工生成的）进行比较计算，得到对应的得分。

ROUGE指标与BLEU指标非常类似，均可用来衡量生成结果和标准结果的匹配程度，不同的是ROUGE基于召回率，BLEU更看重准确率。

ROUGE分为四种方法：ROUGE-N, ROUGE-L, ROUGE-W, ROUGE-S.

下面举例说计算过程（这里只介绍ROUGE_N）：

- 基本步骤：
  - Rouge-N实际上是将模型生成的结果和标准结果按N-gram拆分后，计算召回率
- 假设模型预测文本和一个目标文本如下：

```
预测文本：It is a nice day today
目标文本：Today is a nice day
```

- 使用ROUGE-1进行匹配

```
预测文本：{it, is, a, nice, day, today}
目标文本：{today, is, a, nice, day}
结果：
  :其中{today, is, a, nice, day}匹配，所以匹配度为5/5=1,这说明生成的内容完全覆盖了参考文本中的所有单词，质量较高。
```

- 通过类似的方法，可以计算出其他ROUGE指标（如ROUGE-2、ROUGE-L、ROUGE-S）的评分，从而综合评估系统生成的文本质量。

---

python代码实现：

```python
# 第一步：安装rouge-->pip install rouge
from rouge import Rouge

# 预测文本
generated_text = "This is some generated text."

# 目标文本列表
reference_texts = ["This is a reference text.", "This is another generated reference text."]

# 计算 ROUGE 指标
rouge = Rouge()
scores = rouge.get_scores(generated_text, reference_texts[1])

# 打印结果
print("ROUGE-1 precision:", scores[0]["rouge-1"]["p"])
print("ROUGE-1 recall:", scores[0]["rouge-1"]["r"])
print("ROUGE-1 F1 score:", scores[0]["rouge-1"]["f"])
# ROUGE-1 precision: 0.8
# ROUGE-1 recall: 0.6666666666666666
# ROUGE-1 F1 score: 0.7272727223140496
```

3.3 困惑度PPL(perplexity)

PPL用来度量一个概率分布或概率模型预测样本的好坏程度。

PPL基本思想:

- 给测试集的句子赋予较高概率值的语言模型较好,当语言模型训练完之后，测试集中的句子都是正常的句子，那么训练好的模型就是在测试集上的概率越高越好.
- 基本公式（两种方式）：

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N}}$$

$$PP(S) = 2^{-\frac{1}{N}\sum log(P(w_i))}$$

- 由公式可知，**句子概率越大，语言模型越好，迷惑度越小。**

---

python代码实现：

```
import math
# 定义语料库
sentences = [
['I', 'have', 'a', 'pen'],
['He', 'has', 'a', 'book'],
['She', 'has', 'a', 'cat']
]
# 定义语言模型
unigram = {
'I': 1/12,
'have': 1/12,
'a': 3/12,
'pen': 1/12,
'He': 1/12,
'has': 2/12,
'book': 1/12,
'She': 1/12,
'cat': 1/12
}
# 计算困惑度
perplexity = 0
for sentence in sentences:
    sentence_prob = 1
    for word in sentence:
        sentence_prob *= unigram[word]
    sentence_perplexity = 1/sentence_prob
    perplexity += math.log(sentence_perplexity, 2) #以2为底
perplexity = 2 ** (-perplexity/len(sentences))
print('困惑度为：', perplexity)
# 困惑度为：  0.0002296
```

小结总结

- 本小节主要介绍LLM的背景知识，了解目前LLM发展基本历程
- 对语言模型的类别分别进行了介绍，如基于统计的N-gram模型，以及深度学习的神经网络模型
