Chapter 3

循环神经网络

循环神经网络

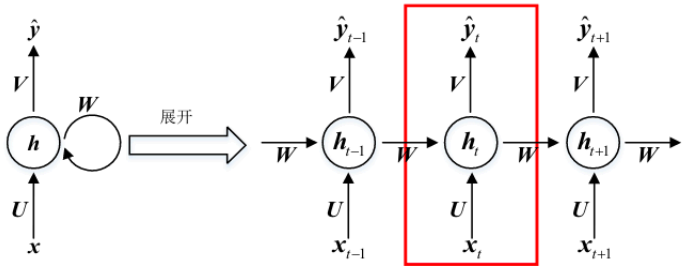
意义：序列数据建模

循环神经网络的定义

• 循环神经网络是一种人工神经网络，它的节点间的连接形成一 个遵循时间序列的有向图。

• 核心思想：样本间存在顺序关系，每个样本和它之前的样本存 在关联。通过神经网络在时序上的展开，我们能够找到样本之 间的序列相关性。

结构



$$\boldsymbol{h}_t = \sigma(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{b}_s)$$

$$\hat{\boldsymbol{y}}_t = \varphi(\boldsymbol{V}\boldsymbol{h}_t + \boldsymbol{b}_y)$$

- x_t是t时刻的输入
- h_t是t时刻的记忆
- y_t是t时刻的输出
- U、V、W是RNN的连接权重
- b_s、b_y是RNN的偏置
- σ、φ是激活函数，σ通常选用Tanh或Sigmoid，φ通常选用Softmax

16

RNN训练算法-BPTT

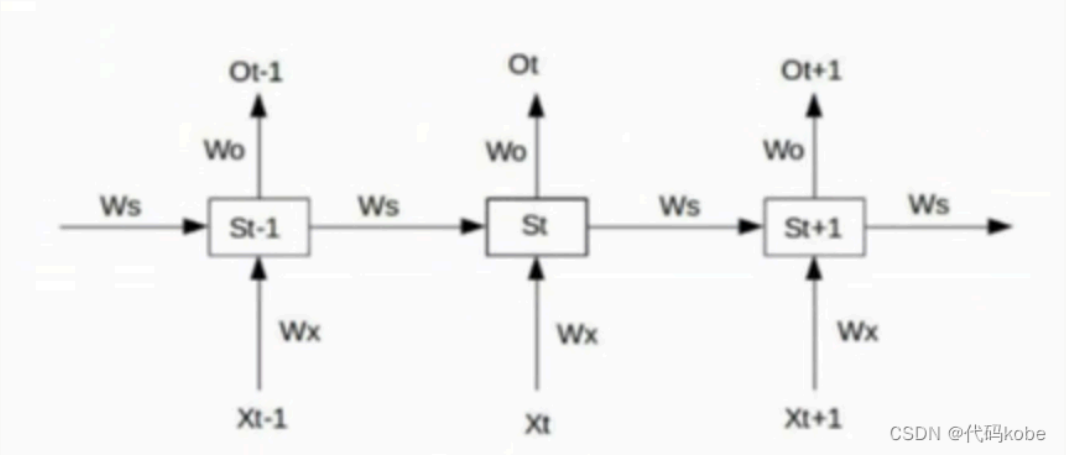
长短时记忆网络

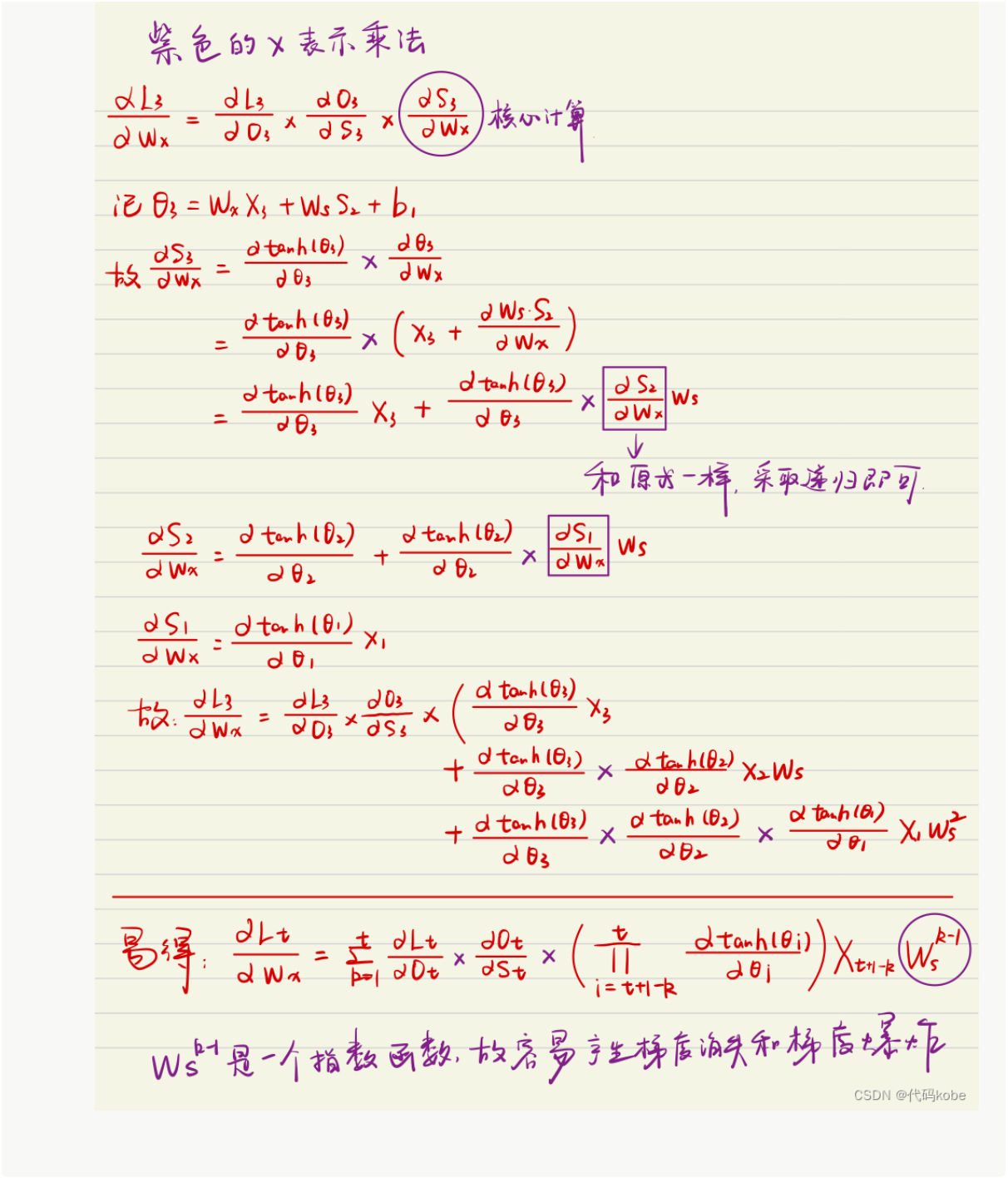
不能有效解决长时依赖问题（由梯度消失引起的）

- 梯度消失的原因
  - BPTT算法
  - 激活函数Ta n h
- 解决方案
  - ReLU函数
  - 门控RNN（LSTM）

LSTM的定义

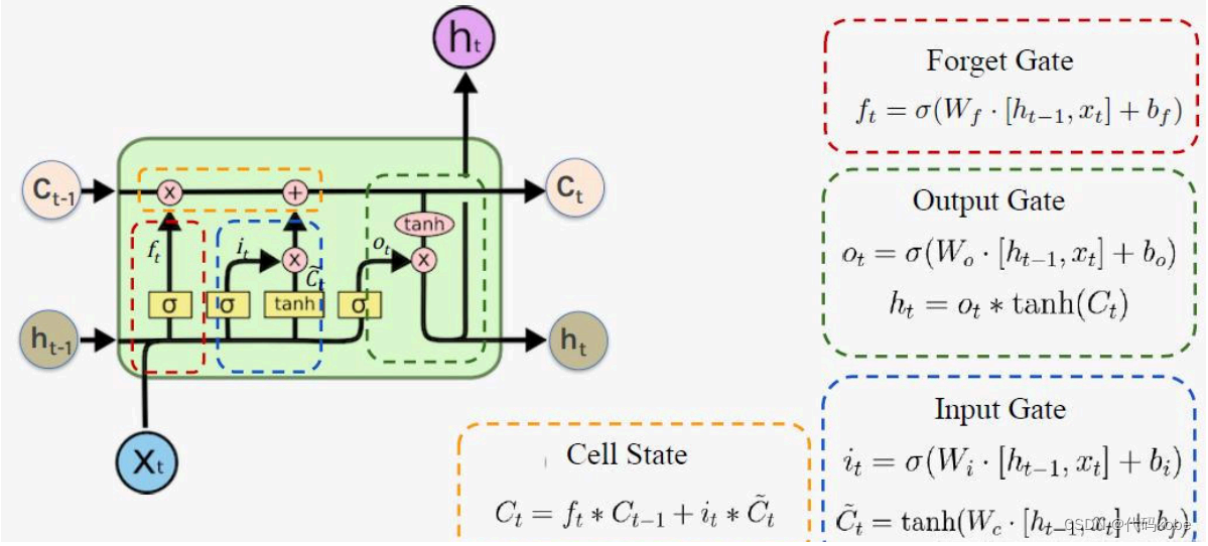
RNN存在的梯度爆炸问题推导



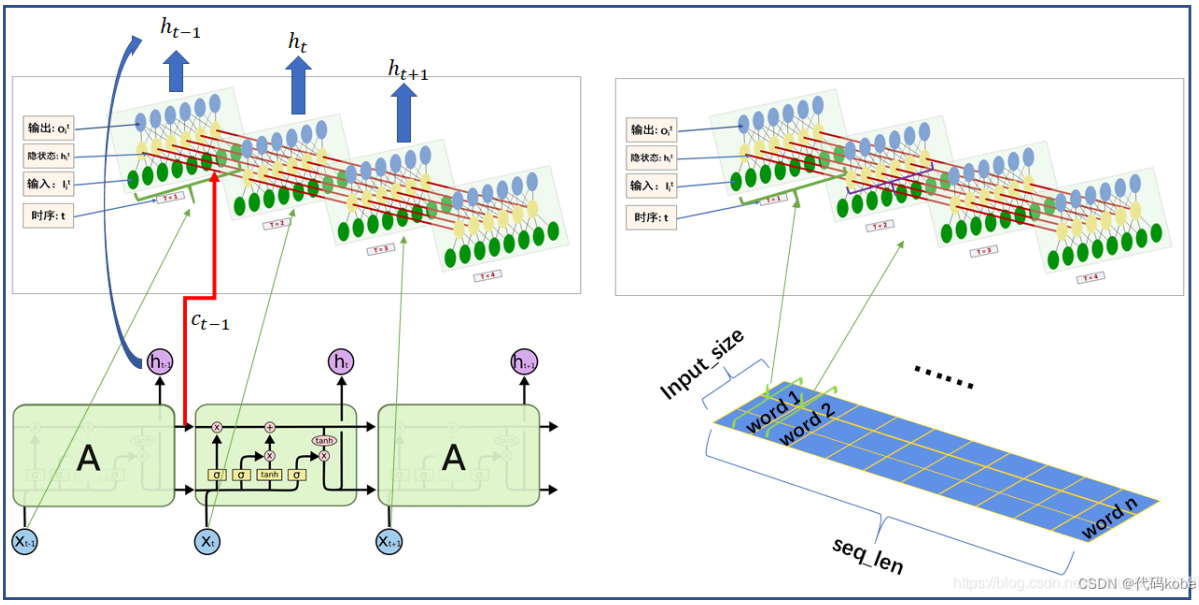


Long short-term memory

• LSTM 单元（ unit ）一般由一个细胞（ cell ），一个输入门 （input gate），一个输出门（output gate）和一个遗忘门 （forget gate）组成.



- 细胞能够记住任意时间间隔上的值，三个门能够控制进出细胞的 信息流动。



在LSTM的输入和输出门中使用tanh函数有以下几个原因：

1. 为了防止**梯度消失问题**，我们需要一个二次导数在大范围内不为0的函数，而tanh函数可以满足这一点

2. 为了便于凸优化，我们需要一个**单调函数**

3. tanh函数一般收敛的更快

4. tanh函数的求导占用系统的资源更少

💡
1. **梯度可以在 cell state 上“直通”传播**，不容易消失
2. **门控机制决定信息的“何时写入、何时忘记、何时输出”**
3. **模型能有选择地记住重要的长期信息，屏蔽不重要的部分**

双向RNN

双向 RNN（**Bidirectional RNN，BiRNN**）是一种**可以同时利用序列的过去和未来信息**的神经网络结构，常用于自然语言处理、语音识别等任务。它的核心思想是：**同时使用正向和反向两个 RNN 对序列进行建模，从而捕获上下文中的双向依赖关系。**

▶️ 输入：序列 $X = [x_1, x_2, \ldots, x_T]$

BiRNN 会构建两个独立的 RNN：

- **正向 RNN**：从 $x_1 \rightarrow x_T$ 依次处理输入序列
- **反向 RNN**：从 $x_T \rightarrow x_1$ 倒序处理输入序列

✨ 每个时间步的输出：

对于每一个时间步 t，我们会得到：

$$\overrightarrow{h}_t = \mathrm{RNN}_{\text{forward}}(x_1, \ldots, x_t)$$

$$\overleftarrow{h}_t = \mathrm{RNN}_{\text{backward}}(x_T, \ldots, x_t)$$

最终输出为两个隐藏状态的**拼接**或**加权融合**：

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$$

这个 h_t 就是 BiRNN 的输出，可以接入后续的全连接层、CRF层、分类器等。

特点	说明
双向上下文捕捉	能同时获取前文（past）与后文（future）信息
更适合全局依赖任务	如命名实体识别、语义角色标注、语音识别等
增加计算与参数量	两个 RNN，计算量和内存需求翻倍
对时间依赖敏感	不适合在线预测或流式场景（不能等未来）

循环神经网络的典型应用

RNN的不同结构

类型	输入	输出	应用场景
1️⃣ One-to-One	单个输入	单个输出	图像分类、情感极性判断等
2️⃣ One-to-Many	单个输入	序列输出	图像描述生成（Image Captioning）
3️⃣ Many-to-One	序列输入	单个输出	情感分析、视频分类等
4️⃣ Many-to-Many（等长）	序列输入	序列输出（长度相等）	词性标注、语音识别、时间序列预测
5️⃣ Many-to-Many（不等长）	序列输入	序列输出（长度不同）	机器翻译、对话系统、语音生成等

语言模型

语言模型（**Language Model, LM**）是一个**能够对词语序列建模、预测下一个词出现概率的模型**。它是自然语言处理（NLP）的核心组件之一，用于理解、生成和评估语言。

基于文法的语言模型

统计语言模型

N-gram语言模型

💡 **为什么会发生零概率？**

- N-gram 语言模型依赖训练数据中的 **共现频率**。
- 即便训练语料很大，也仍然无法覆盖语言中所有可能的词序列（尤其是长 n-gram）。
- 语言天然稀疏，很多合理的词组组合未必在训练语料中出现过。

如何解决？

为了解决零概率事件，我们采用 **平滑（Smoothing）方法**。

数据平滑

语言模型中的 **困惑度（Perplexity, PP）** 是衡量模型在测试集上预测能力的一个指标，表示模型对语言的“困惑程度”，也可以理解为：

> 平均每个词的“不确定性”有多大 —— 困惑度越低，说明模型越“有把握”预测下一个词，即模型越好。

1️⃣ 加一平滑（Laplace Smoothing）

$$P(w_n \mid w_{n-1}, \ldots) = \frac{\text{Count}(n\text{-gram}) + 1}{\text{Count}(n-1\text{-gram}) + V}$$

其中 V 是词汇表大小。即每个组合都加上 1，避免出现 0。

2️⃣ Kneser-Ney 平滑（最先进）

- 更复杂但效果更好
- 不只是减去固定折扣，而是结合了词的多样性
- 尤其适合实际应用中的语言模型任务

3️⃣ Back-off / Interpolation

- 若 trigram 没出现，则退化到 bigram，甚至 unigram
- 或直接加权融合多阶模型的概率（线性插值）

神经网络语言模型

前馈神经网络语言模型

NNLM

我们将模型$f(w_t,\dots,w_{t-n+1})=\hat{P}(w_t|w_1^{t-1})$分解成**「两个部分」**：

1. 将词表V中的单词i映射为向量的函数$C(i)\in R^m$，这表示每个词的分布式向量表示(distributed feature vectors)。实际上C是一个$|V|\times m$的矩阵,m表示词向量的维度，矩阵的第i行是单词i的词向量。
2. 函数g将词向量表示的输入序列$(C(w_{t-n+1}),\dots,C(w_{t-1}))$映射成一个预测下一个词的概率分布，注意这个分布是在词表V上的。g的输出是一个向量，向量的第i个值代表下一个词是i的概率，即$\hat{P}(w_t=i|w_1^{t-1})$。

两个部分融合得到f：

$$f(i,w_{t-1},\dots,w_{t-n+1})=g(i,C(w_{t-1}),\dots,C(w_{t-n+1}))$$

第一个部分对应着输入层，第二个部分对应着隐层和输出层。

首先是将词序列$(w_{t-1},w_{t-2},\dots,w_{t-n+1})$映射成向量，将它们首尾拼接得到新的向量：

$$x=(C(w_{t-1}),C(w_{t-2}),\dots,C(w_{t-n+1}))$$

其中C是参数矩阵。然后是将输入序列的向量建模成概率分布，由函数g实现。函数g的实现可以是前向神经网络或者循环神经网络或者其它参数化的函数(parametrized function)。论文中的实现如下：

$$y=b+Wx+Utanh(d+Hx)$$

$tanh(d+Hx)$是隐藏层。这里的W,H,U,b都是参数矩阵，W矩阵包含了从输入层到输出层的直连边。W可以是0，即x对y没有直接影响。实验结果没有表明直连边是否有效，不过在小样本实验中发现，有直连边可以减少迭代次数，没有直连边的模型泛化能力更好一些。

为了使得概率和为1，还要应用softmax函数到输出层：

$$\hat{P}(w_t|w_{t-1},\dots,w_{t-n+1})=\frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

y_i是y的一部分，表示下一个单词是i的概率，这个概率没有经过归一化。

模型训练时，需要最大化下列的式子：

$$L = \frac{1}{T}\sum_t logf(w_t, w_{t-1}, \dots, w_{t-n+1}; \theta) + R(\theta)$$

第一部分是最大化给定序列w_1^{t-1}，下一个词是w_t的概率，也就是最大化真实序列的概率。第二部分的$R(\theta)$是正则项。

模型是采用随机梯度下降法✦进行优化的。

对于OOV（词表之外的词），会给它一个初始的向量，这个向量是和这个词上下文相似的词的向量的加权和。即对于一个在w_{t-n+1}^{t-1}上下文的词j：

$$C(j) \leftarrow \sum_{i \in V} C(i)\hat{P}(i|w_{t-n+1}^{t-1})$$

实验结果表明，NNLM模型比smoothed trigram（平均分布, unigram, bigram, trigram的加权结果）表现更好。接下来就是神经网络模型的天下了~

💡

假设我们有词嵌入矩阵 $C \in \mathbb{R}^{|V| \times d}$，每个词表示为 d-维向量。

给定上下文词 $(w_{t-3}, w_{t-2}, w_{t-1})$，模型进行如下计算：

1. 嵌入层：

$$x = [C(w_{t-3}); C(w_{t-2}); C(w_{t-1})] \in \mathbb{R}^{(N-1) \cdot d}$$

2. 隐藏层：

$$h = \tanh(W^{(1)} x + b^{(1)})$$

3. 输出层：

$$y = \text{softmax}(W^{(2)} h + b^{(2)})$$

4. 模型输出：预测词 w_t 的概率分布 $y \in \mathbb{R}^{|V|}$

优点

- **可泛化**：词嵌入可以捕捉词语的语义相似性，即使没见过某个 n-gram 组合，也能有不错的估计。
- **连续空间建模**：避免了传统 N-gram 的离散计数问题。
- **端到端训练**：词嵌入和语言模型可以联合优化。

缺点

- **固定窗口限制**：只能建模固定长度的上下文，不能捕捉更长距离的依赖。
- **计算复杂**：输出层维度大（词表越大 softmax 越慢），需使用采样或层次 softmax 减少计算。
- **参数多**：词嵌入和隐藏层参数量大，需较多训练样本。

循环神经网络语言模型

LSTM语言模型

基于注意力机制的语言模型

自动文本摘要

1. 抽取式摘要（Extractive Summarization）

抽取式摘要方法直接从原文中提取出最重要的句子或片段作为摘要，而不生成新的文本。其流程包括以下几个步骤：

流程：

1. **文本预处理**：包括去除停用词、标点符号、数字等，以及对文本进行分词（Tokenization）和词性标注（POS Tagging）。
2. **句子表示**：通过对每个句子进行特征提取，获得句子的表示。常见的特征包括：
   - 句子长度
   - 句子中词频的TF-IDF（Term Frequency-Inverse Document Frequency）值
   - 句子中关键词的权重
   - 句子的位置（是否在文中开头或结尾）
   - 句子的相似度（与其他句子的相似度）

   这些特征可以通过向量空间模型表示，每个句子用一个向量来表示。
3. **句子评分**：基于提取的特征，为每个句子分配一个评分。常见的评分方法包括：
   - **TF-IDF加权法**：计算句子中单词的TF-IDF值。
   - **TextRank**：一种基于图的算法，类似于PageRank算法，通过图中节点之间的连接来评估句子的权重。

   例如，TextRank通过构建句子之间的相似度图，并通过随机游走算法来计算每个句子的“重要性”。
4. **选择摘要**：根据句子的评分，选取得分最高的若干个句子作为摘要。通常会设置一个阈值，筛选出最能代表全文的句子。
5. **输出摘要**：将选出的句子按原文中的顺序连接起来，生成摘要。

公式：

1. **TF-IDF**计算公式：
   - $\mathrm{TF}(t) = \frac{\text{词t在句子中的出现次数}}{\text{句子总词数}}$
   - $\mathrm{IDF}(t) = \log \frac{N}{\text{含有词t的文档数}}$
   - $\text{TF-IDF}(t) = \mathrm{TF}(t) \times \mathrm{IDF}(t)$
2. **TextRank**的评分公式：
   - $S_i = (1-d) + d \sum_{j \in \mathcal{N}(i)} \frac{S_j}{C_j}$
     - S_i：句子i的得分。
     - d：阻尼因子（通常为0.85）。
     - $\mathcal{N}(i)$：与句子i相连的句子集合。
     - C_j：句子j的出度（即连接到句子j的边的数量）。

2. 生成式摘要（Abstractive Summarization）

生成式摘要方法与抽取式摘要不同，它不局限于从原文中提取，而是通过自然语言生成新的句子来概括原文的意思。生成式摘要通常使用深度学习技术，尤其是基于**循环神经网络（RNN）、长短时记忆网络（LSTM）或Transformer**的序列到序列（Seq2Seq）模型。

流程：

1. **文本编码**：将输入的长文本通过编码器进行编码，通常采用RNN或LSTM等结构来处理文本的时序信息。Transformer模型使用多头自注意力机制（Self-Attention）来捕捉文本中的长距离依赖。
2. **上下文建模**：生成式摘要需要理解文本的整体语境。通过上下文信息，模型生成新的句子时不仅考虑当前句子的内容，还要根据全文内容进行推理。
3. **解码生成摘要**：解码器生成一个个词或字符，**通过最大化条件概率来输出每个词的概率分布**，最终生成一段摘要文本。常用的方法包括**Beam Search**来选择最可能的句子。
4. **损失函数**：训练时通常使用交叉熵损失函数来衡量模型输出的摘要与真实摘要之间的差异：

$$L = -\sum_{t=1}^{T} \log P(y_t | y_{1:t-1}, x)$$

- L：损失函数
- T：生成的摘要的长度
- y_t：第t个词
- $P(y_t | y_{1:t-1}, x)$：给定上下文词汇和输入文本，生成第t个词的概率

5. **输出摘要**：解码器生成的输出即为摘要。

Transformer模型中的多头自注意力机制：

Transformer模型中的自注意力机制通过计算输入序列中各个单词之间的相关性来进行信息传递。具体步骤如下：

1. **计算查询、键、值**：
   - 查询（Query）、键（Key）、值（Value）是从输入嵌入（Embeddings）通过线性变换得到的。
2. **计算注意力权重**：
   - 使用点积计算查询与键之间的相似度：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

   - Q：查询矩阵
   - K：键矩阵
   - V：值矩阵
   - d_k：键向量的维度（用于归一化）
3. **多头注意力**：通过多个头并行计算注意力，最后将每个头的输出拼接起来。

$$\text{Multi-Head Attention} = \text{Concat}(head_1, head_2, ..., head_h)W^O$$

   - h：头数
   - W^O：输出的权重矩阵
4. **输出摘要**：多头自注意力机制计算后的信息传递给后续的层（如前馈神经网络等），最终生成文本摘要。

机器阅读理解

R-Net

R-Net（Reading Comprehension Net）是微软亚洲研究院于 2017 年提出的一种 **用于机器阅读理解（Machine Reading Comprehension, MRC）任务的深度神经网络模型**。它主要应用于 **抽取式问答系统**，即从给定的段落中找到一个连续的子句（span）作为答案。R-Net 的目标是模拟人类在阅读问题并从文档中寻找答案的过程。

R-Net 的关键在于它**融合了问题信息**和**文档上下文信息**，并引入了 **gated attention mechanism** 和 **self-matching attention mechanism**，提升了模型理解文档的能力。

整个流程可以分为以下几个步骤：

🔷1. Input Layer（输入层）

- 输入包括两个序列：
  - **问题**（Question）：一个词序列 $Q = \{q_1, q_2, ..., q_m\}$
  - **文档/段落**（Passage）：一个词序列 $P = \{p_1, p_2, ..., p_n\}$
- 使用词向量（如 GloVe、ELMo）将词转化为向量：

$$x_i^Q = \text{WordEmbedding}(q_i), \quad x_j^P = \text{WordEmbedding}(p_j)$$

🔷2. Embedding Encoder Layer（编码层）

- 使用 **BiGRU（双向门控循环单元）** 对输入的词向量进行编码，获取上下文表示：

$$h_i^Q = \text{BiGRU}(x_i^Q), \quad h_j^P = \text{BiGRU}(x_j^P)$$

得到的问题表示为 $H^Q \in \mathbb{R}^{m \times 2d}$，文档表示为 $H^P \in \mathbb{R}^{n \times 2d}$。

🔷3. Question-Passage Matching Layer（问题与文档匹配）

这是 R-Net 的 **核心创新之一：Gated Attention-based Matching**。

🧠 **Gated Attention Mechanism：**

- 对于文档中的每个词，通过计算它与整个问题之间的注意力来生成一个问题上下文表示：

$$\alpha_j = \text{softmax}(H^Q W h_j^P) \quad (\text{attention 权重})$$

$$c_j = \sum_{i=1}^{m} \alpha_{j,i} h_i^Q \quad (\text{问题上下文表示})$$

- 然后引入门控机制来融合问题上下文与当前词：

$$g_j = \sigma(W_g [h_j^P; c_j]) \quad (\text{门控向量})$$

$$\tilde{h}_j^P = g_j \odot [h_j^P; c_j]$$

- 再通过 BiGRU 得到匹配后的文档表示 $\hat{H}^P$：

$$\hat{h}_j^P = \text{BiGRU}(\tilde{h}_j^P)$$

🔷4. Self-Matching Attention Layer（文档内部匹配）

此层用来建模文档内部的长距离依赖 —— **把文档的每个位置与整个文档匹配**，使得模型可以理解跨句的信息。

- 类似之前的 attention，但是文档与文档自身之间进行：

$$\beta_j = \text{softmax}(\hat{H}^P W_s \hat{h}_j^P)$$

$$d_j = \sum_{k=1}^{n} \beta_{j,k} \hat{h}_k^P$$

- 再进行融合并输入到 BiGRU：

$$\bar{h}_j^P = \text{BiGRU}([\hat{h}_j^P; d_j])$$

最终得到了**增强语义表示的文档表示序列** $\bar{H}^P$。

🔷5. Output Layer（输出层）

此层用于预测答案的起始位置和结束位置：

- 使用两个独立的 **softmax** 层：

$$P_{start} = \text{softmax}(W_s \bar{H}^P), \quad P_{end} = \text{softmax}(W_e \bar{H}^P)$$

- 使用交叉熵损失函数训练模型：

$$L = -\log P_{start}^{(y_s)} - \log P_{end}^{(y_e)}$$

其中 y_s、y_e 分别是真实的起始和终止位置。

交互层（Interaction Layer）是 R-Net 的 **核心部分**，它包含两层结构，分别处理：

1. **问题（Question）与段落（Passage）之间的交互** —— 强调“问题感知的段落表示”
2. **段落（Passage）与自身的交互** —— 强调“段落的全局理解能力”

第一层：问题-段落交互（Question-Passage Matching）

这一层的目标是获取 **Question-aware Passage Representation** —— 也就是每一个段落词（passage word）都要结合问题的语义重新建模。

🌐实现细节：Gated Attention + BiGRU

对段落中每个词 p_j，通过 attention 机制计算它与问题中所有词的相似度（通常是向量内积或可学习的函数），生成权重：

$$\alpha_{j,i} = \text{softmax}(h_i^Q W h_j^P)$$

根据这些权重，计算问题的上下文表示：

$$c_j = \sum_{i=1}^{m} \alpha_{j,i} h_i^Q$$

然后使用**门控机制**来融合这个问题信息和当前段落词：

$$g_j = \sigma(W_g [h_j^P; c_j])$$

$$\tilde{h}_j^P = g_j \odot [h_j^P; c_j]$$

最后用 BiGRU 对整个段落序列重新编码，得到 **question-aware passage 表示**：

$$\hat{h}_j^P = \text{BiGRU}(\tilde{h}_j^P)$$

第二层：段落-自身交互（Passage Self-Matching）

这一步解决的是：虽然我们现在知道了每个词与问题之间的关系，但段落本身的信息还未整合，比如：

- 一个信息在多个句子中分布
- 某些关键词的上下文意义需要跨句分析

这时候就需要**段落与自己进行注意力匹配**，也就是 self-matching attention。

🌐**实现方式：**

$$\beta_{j,k} = \text{softmax}(\hat{h}_k^P W_s \hat{h}_j^P)$$

$$d_j = \sum_{k=1}^{n} \beta_{j,k} \hat{h}_k^P$$

表示为第 j 个位置从整个段落收集到的上下文信息。

之后，将原始表示和注意力融合表示拼接，再次输入到 BiGRU：

$$\bar{h}_j^P = \text{BiGRU}([\hat{h}_j^P; d_j])$$

这样得到了 **语义增强后的段落表示** $\bar{H}^P$，具备了更强的全局理解能力。

BiDAF

BiDAF（**Bidirectional Attention Flow**）是斯坦福提出的一种经典阅读理解模型，特别适用于问答任务中的 **机器理解段落与问题之间的关系**。相比早期的模型，BiDAF 的主要创新在于：

> 双向注意力机制（Bidirectional Attention） —— 既让段落关注问题，也让问题关注段落。

1️⃣ Input Layer（输入层）

输入包括：

- 段落 P：词序列 $p_1, p_2, \ldots, p_n$
- 问题 Q：词序列 $q_1, q_2, \ldots, q_m$

2️⃣ Embedding Layer（嵌入层）

将每个词转换为词向量，常用：

- **预训练词向量**（如 GloVe）
- **字符级卷积（Char-CNN）**提取词形信息（比如处理拼写变化）

结果是：

- $\mathbf{P} \in \mathbb{R}^{d \times n}$
- $\mathbf{Q} \in \mathbb{R}^{d \times m}$

3️⃣ Contextual Embedding Layer（上下文编码层）

用 **双向 LSTM / GRU** 对词向量进行上下文建模：

$$\mathbf{H}^P = \mathrm{BiLSTM}(\mathbf{P}) \quad (\text{段落})$$

$$\mathbf{H}^Q = \mathrm{BiLSTM}(\mathbf{Q}) \quad (\text{问题})$$

输出：

- $\mathbf{H}^P \in \mathbb{R}^{2d \times n}$
- $\mathbf{H}^Q \in \mathbb{R}^{2d \times m}$

4️⃣ Attention Flow Layer（双向注意力层）🧠🔥

BiDAF 的核心创新层

这一步计算问题和段落之间**双向注意力**，构建两种交互：

（1）段落对问题的注意力（Context-to-Query Attention）

衡量段落中每个词对问题中各词的关注程度：

$$S_{ij} = \alpha(\mathbf{H}_i^P, \mathbf{H}_j^Q)$$

通常是三角函数：

$$S_{ij} = w^\top [\mathbf{H}_i^P; \mathbf{H}_j^Q; \mathbf{H}_i^P \odot \mathbf{H}_j^Q]$$

然后 softmax 后求 attention-weighted sum 得到：

$$\mathbf{A}_i = \sum_j \text{softmax}(S_{ij}) \cdot \mathbf{H}_j^Q$$

这是段落中每个位置与整个问题融合的信息。

（2）问题对段落的注意力（Query-to-Context Attention）

找到"最受问题关注"的段落位置：

$$\mathbf{b} = \text{softmax}(\max_j S_{ij}) \quad \text{（跨列最大）}$$

再计算：

$$\mathbf{C} = \sum_i b_i \cdot \mathbf{H}_i^P$$

然后把这个段落摘要（相同向量）拼接到每个位置。

🔁 融合 Attention 信息

最终每个段落位置的表示是拼接的结果：

$$\mathbf{G}_i = [\mathbf{H}_i^P; \mathbf{A}_i; \mathbf{H}_i^P \odot \mathbf{A}_i; \mathbf{H}_i^P \odot \mathbf{C}]$$

其中 ⊙ 是逐元素乘。

5️⃣ Modeling Layer（建模层）

用 BiLSTM 进一步处理融合后的向量 $\mathbf{G}_i$，学习上下文依赖的综合信息：

$$\mathbf{M} = \text{BiLSTM}(\mathbf{G})$$

6️⃣ Output Layer（输出层）

预测答案的起始位置和结束位置：

$$P_{\text{start}} = \text{softmax}(W_1[\mathbf{G}; \mathbf{M}])$$

$$P_{\text{end}} = \text{softmax}(W_2[\mathbf{G}; \text{BiLSTM}(\mathbf{M})])$$

最终选择一对 (i, j) 使得 $i \leq j$ 且 $P_{\text{start}}(i) \cdot P_{\text{end}}(j)$ 最大。