

# AI/ML 综合练习题答案

数学基础 · 深度学习 · 大语言模型 · Python 编程

详细解答与代码实现

AI/ML 系列教程

# AI/ML 综合练习题答案

2026 年 1 月 6 日

## 目录

<b>I 数学基础练习题答案</b>	<b>3</b>
1 计算题答案 .....	3
1.1 矩阵运算 .....	3
1.2 概率计算 .....	4
1.3 最大似然估计 .....	4
1.4 信息论 .....	5
1.5 优化问题 .....	6
2 概念题答案 .....	6
2.1 线性代数 .....	6
2.2 概率论 .....	7
<b>II 深度学习练习题答案</b>	<b>7</b>
3 概念题答案 .....	8
3.1 梯度消失问题 .....	8
3.2 卷积神经网络 .....	8
<b>III 大语言模型练习题答案</b>	<b>9</b>
4 概念题答案 .....	9

4.1	Transformer 架构 . . . . .	9
4.2	预训练与微调 . . . . .	10
4.3	提示工程 . . . . .	11
4.4	RAG . . . . .	11
5	编程题答案 . . . . .	12
5.1	实现 Transformer 编码器 . . . . .	12
5.2	实现注意力机制 . . . . .	14

## IV 大语言模型先进技术练习题答案 16

6	概念理解题答案 . . . . .	16
7	计算题答案 . . . . .	18
7.1	困惑度计算 . . . . .	18
7.2	BLEU 计算 . . . . .	19
7.3	ROUGE 计算 . . . . .	21
8	代码实现题答案 . . . . .	24
8.1	LoRA 训练脚本 . . . . .	24
8.2	批量困惑度计算 . . . . .	26

## V Python/NumPy/Pandas 练习题答案 28

9	NumPy 综合例题答案 . . . . .	28
9.1	例题 1：数据预处理 . . . . .	28
10	总结 . . . . .	29

## Part I

# 数学基础练习题答案

## 1 计算题答案

### 1.1 矩阵运算

1. 矩阵乘法：

$$\mathbf{AB} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \quad (1)$$

$$= \begin{bmatrix} 1 \times 5 + 2 \times 7 & 1 \times 6 + 2 \times 8 \\ 3 \times 5 + 4 \times 7 & 3 \times 6 + 4 \times 8 \end{bmatrix} \quad (2)$$

$$= \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix} \quad (3)$$

转置：

$$\mathbf{A}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \quad (4)$$

逆矩阵：

$$\det(\mathbf{A}) = 1 \times 4 - 2 \times 3 = -2 \quad (5)$$

$$\mathbf{A}^{-1} = \frac{1}{-2} \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix} \quad (6)$$

2. 向量范数：

$$\|\mathbf{x}\|_1 = |1| + |2| + |3| = 6 \quad (7)$$

$$\|\mathbf{x}\|_2 = \sqrt{1^2 + 2^2 + 3^2} = \sqrt{14} \approx 3.74 \quad (8)$$

$$\|\mathbf{x}\|_\infty = \max(|1|, |2|, |3|) = 3 \quad (9)$$

3. 点积和余弦相似度：

$$\mathbf{u} \cdot \mathbf{v} = 1 \times 0 + 0 \times 1 + 1 \times 1 = 1 \quad (10)$$

$$\|\mathbf{u}\|_2 = \sqrt{1^2 + 0^2 + 1^2} = \sqrt{2} \quad (11)$$

$$\|\mathbf{v}\|_2 = \sqrt{0^2 + 1^2 + 1^2} = \sqrt{2} \quad (12)$$

$$\cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2} = \frac{1}{\sqrt{2} \times \sqrt{2}} = \frac{1}{2} \quad (13)$$

## 1.2 概率计算

1. 二项分布:

$$P(X=2) = \binom{3}{2} \left(\frac{1}{2}\right)^2 \left(\frac{1}{2}\right)^{3-2} \quad (14)$$

$$= 3 \times \frac{1}{4} \times \frac{1}{2} = \frac{3}{8} = 0.375 \quad (15)$$

2. 标准正态分布:

$$P(-1 < X < 1) = \Phi(1) - \Phi(-1) \quad (16)$$

$$= \Phi(1) - (1 - \Phi(1)) \quad (17)$$

$$= 2\Phi(1) - 1 \quad (18)$$

$$\approx 2 \times 0.8413 - 1 = 0.6826 \quad (19)$$

其中  $\Phi$  是标准正态分布的累积分布函数。

3. 贝叶斯定理:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (20)$$

$$= \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\neg A)P(\neg A)} \quad (21)$$

$$= \frac{0.8 \times 0.3}{0.8 \times 0.3 + 0.2 \times 0.7} \quad (22)$$

$$= \frac{0.24}{0.24 + 0.14} = \frac{0.24}{0.38} \approx 0.6316 \quad (23)$$

## 1.3 最大似然估计

1. 泊松分布 MLE:

$$L(\lambda) = \prod_{i=1}^3 \frac{\lambda^{x_i} e^{-\lambda}}{x_i!} \quad (24)$$

$$\ell(\lambda) = \log L(\lambda) = \sum_{i=1}^3 (x_i \log \lambda - \lambda - \log(x_i!)) \quad (25)$$

$$\frac{\partial \ell}{\partial \lambda} = \frac{1}{\lambda} \sum_{i=1}^3 x_i - 3 = 0 \quad (26)$$

$$\hat{\lambda} = \frac{1+2+3}{3} = 2 \quad (27)$$

2. 指数分布 MLE:

$$L(\lambda) = \prod_{i=1}^n \lambda e^{-\lambda x_i} = \lambda^n e^{-\lambda \sum_{i=1}^n x_i} \quad (28)$$

$$\ell(\lambda) = n \log \lambda - \lambda \sum_{i=1}^n x_i \quad (29)$$

$$\frac{\partial \ell}{\partial \lambda} = \frac{n}{\lambda} - \sum_{i=1}^n x_i = 0 \quad (30)$$

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^n x_i} = \frac{1}{\bar{x}} \quad (31)$$

## 1.4 信息论

1. 伯努利分布熵:

$$H(X) = -p \log p - (1-p) \log(1-p) \quad (32)$$

求导找最大值:

$$\frac{\partial H}{\partial p} = -\log p - 1 + \log(1-p) + 1 = \log\left(\frac{1-p}{p}\right) = 0 \quad (33)$$

$$\frac{1-p}{p} = 1 \Rightarrow p = 0.5 \quad (34)$$

当  $p = 0.5$  时, 熵最大,  $H_{\max} = -\log(0.5) = \log 2$ 。

2. 信息论计算:

$$P(X = 0) = 0.3 + 0.2 = 0.5 \quad (35)$$

$$P(X = 1) = 0.1 + 0.4 = 0.5 \quad (36)$$

$$P(Y = 0) = 0.3 + 0.1 = 0.4 \quad (37)$$

$$P(Y = 1) = 0.2 + 0.4 = 0.6 \quad (38)$$

$$H(X) = -0.5 \log(0.5) - 0.5 \log(0.5) = \log 2 = 1 \quad (39)$$

$$H(Y) = -0.4 \log(0.4) - 0.6 \log(0.6) \approx 0.971 \quad (40)$$

$$H(X|Y) = \sum_y P(Y=y) H(X|Y=y) \quad (41)$$

$$= 0.4 \times H(X|Y=0) + 0.6 \times H(X|Y=1) \quad (42)$$

$$= 0.4 \times \left( -\frac{0.3}{0.4} \log \frac{0.3}{0.4} - \frac{0.1}{0.4} \log \frac{0.1}{0.4} \right) \quad (43)$$

$$+ 0.6 \times \left( -\frac{0.2}{0.6} \log \frac{0.2}{0.6} - \frac{0.4}{0.6} \log \frac{0.4}{0.6} \right) \quad (44)$$

$$\approx 0.875 \quad (45)$$

$$I(X;Y) = H(X) - H(X|Y) \approx 1 - 0.875 = 0.125 \quad (46)$$

## 1.5 优化问题

1. 梯度下降：

$$f(x) = x^2 + 2x + 1 \quad (47)$$

$$f'(x) = 2x + 2 \quad (48)$$

迭代过程：

$$x_0 = 0 \quad (49)$$

$$x_1 = x_0 - 0.1 \times f'(0) = 0 - 0.1 \times 2 = -0.2 \quad (50)$$

$$x_2 = -0.2 - 0.1 \times f'(-0.2) = -0.2 - 0.1 \times 1.6 = -0.36 \quad (51)$$

$$x_3 = -0.36 - 0.1 \times f'(-0.36) = -0.36 - 0.1 \times 1.28 = -0.488 \quad (52)$$

$$x_4 = -0.488 - 0.1 \times f'(-0.488) = -0.488 - 0.1 \times 1.024 = -0.5904 \quad (53)$$

$$x_5 = -0.5904 - 0.1 \times f'(-0.5904) = -0.5904 - 0.1 \times 0.8192 = -0.6723 \quad (54)$$

最小值在  $x = -1$  处， $f(-1) = 0$ 。

2. 拉格朗日乘数法：

$$L(x, y, \lambda) = x^2 + y^2 + \lambda(1 - x - y) \quad (55)$$

$$\frac{\partial L}{\partial x} = 2x - \lambda = 0 \Rightarrow x = \frac{\lambda}{2} \quad (56)$$

$$\frac{\partial L}{\partial y} = 2y - \lambda = 0 \Rightarrow y = \frac{\lambda}{2} \quad (57)$$

$$\frac{\partial L}{\partial \lambda} = 1 - x - y = 0 \Rightarrow x + y = 1 \quad (58)$$

解得： $x = y = 0.5$ ，最小值为  $0.5^2 + 0.5^2 = 0.5$ 。

## 2 概念题答案

### 2.1 线性代数

1. 特征值和特征向量的几何意义：

- 特征向量是矩阵变换后方向不变的向量
- 特征值是特征向量在变换中的缩放因子
- 几何上，特征向量指向矩阵变换的主要方向

2. 矩阵转置在机器学习中的应用：

- 维度匹配：确保矩阵乘法的维度正确
- 梯度计算：反向传播中需要转置
- 内积计算： $\mathbf{a}^T \mathbf{b}$  计算向量内积

3. 矩阵乘法不满足交换律：

- 一般情况下  $\mathbf{AB} \neq \mathbf{BA}$
- 在神经网络中，前向传播的顺序是固定的： $\mathbf{W}^{(l)} \mathbf{h}^{(l-1)}$
- 这种顺序设计是有意的，符合数据流动的方向

## 2.2 概率论

1. 先验、似然、后验：

- 先验概率  $P(A)$ : 在观察到数据之前的概率
- 似然函数  $P(B|A)$ : 给定参数下观察到数据的概率
- 后验概率  $P(A|B)$ : 观察到数据后参数的概率
- 关系： $P(A|B) \propto P(B|A) \times P(A)$  (贝叶斯定理)

2. 误差服从正态分布的原因：

- 中心极限定理：多个独立随机变量的和近似正态分布
- 数学性质好：便于推导和计算
- 实际中很多误差确实近似正态分布

3. MLE vs MAP：

- **MLE**: 最大化似然函数，不考虑先验
- **MAP**: 最大化后验概率，考虑先验信息
- $MAP = MLE + \text{先验正则化}$

## Part II

# 深度学习练习题答案

## 3 概念题答案

### 3.1 梯度消失问题

#### 1. 什么是梯度消失:

- 在深层网络中，梯度在反向传播时逐渐变小，最终接近 0
- 导致浅层参数几乎不更新，网络无法有效学习
- 数学上： $\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial h^{(L)}} \prod_{l=2}^L \frac{\partial h^{(l)}}{\partial h^{(l-1)}}$ ，如果每个  $\frac{\partial h^{(l)}}{\partial h^{(l-1)}} < 1$ ，乘积会指数级衰减

#### 2. 缓解方法:

- **ReLU 激活函数**: 导数为 1 (正区间)，避免梯度衰减
- **残差连接**: 提供梯度直通路径
- **批量归一化**: 稳定激活值分布
- **梯度裁剪**: 防止梯度爆炸

#### 3. ReLU 缓解梯度消失:

- ReLU 在正区间的导数为 1，不会缩小梯度
- 相比 Sigmoid (导数最大 0.25)，梯度传播更稳定

### 3.2 卷积神经网络

#### 1. 局部连接和权重共享:

- **局部连接**: 每个神经元只连接输入的一个局部区域 (如  $3 \times 3$ )
- **权重共享**: 同一卷积核在整个输入上滑动，共享同一组参数
- 参数量：从全连接的  $d_{in} \times d_{out}$  减少到  $k \times k \times C_{out}$  ( $k$  是卷积核大小)

#### 2. 最大池化 vs 平均池化:

- **最大池化**: 保留最显著特征，对噪声更鲁棒，适合特征检测
- **平均池化**: 保留整体信息，更平滑，适合需要整体特征的场景

### 3. CNN 适合图像数据:

- 局部相关性: 图像中相邻像素相关性强
- 平移不变性: 物体位置变化不影响识别
- 层次化特征: 从边缘 → 形状 → 物体

## Part III

# 大语言模型练习题答案

## 4 概念题答案

### 4.1 Transformer 架构

#### 1. 自注意力 vs 交叉注意力:

- **自注意力:** Query、Key、Value 来自同一序列
- **交叉注意力:** Query 来自一个序列, Key 和 Value 来自另一个序列
- 应用: 编码器用自注意力, 解码器用自注意力和交叉注意力

#### 2. 位置编码的必要性:

- Transformer 没有循环结构, 无法感知位置信息
- 位置编码为每个位置添加位置信息
- 使模型能够理解序列的顺序关系

#### 3. 多头注意力的优势:

- **多视角学习:** 不同头可以关注不同的关系类型 (如语法、语义、位置关系等), 从多个角度理解输入序列
- **表达能力增强:** 通过多个子空间学习, 增强模型的表达能力, 相当于从多个角度理解输入
- **并行计算:** 多个头可以完全并行计算, 虽然时间复杂度与单头相同, 但可以充分利用 GPU 的并行计算能力
- **参数量优化:** 使用共享投影矩阵, 参数量为  $O(d_{model}^2)$ , 比单独定义每个头的投影矩阵更高效

- **实际效果:** 研究表明, 训练好的模型中不同头确实学习到了不同的模式, 有些关注局部依赖, 有些关注长距离依赖

## 4.2 预训练与微调

### 1. 自回归语言建模 vs 掩码语言建模:

- **自回归语言建模 (Autoregressive LM):**
  - 原理: 给定前面的词, 预测下一个词, 如 GPT 系列
  - 优点: 适合生成任务, 可以生成连贯的文本
  - 缺点: 只能利用单向上下文, 无法同时看到前后文
- **掩码语言建模 (Masked LM):**
  - 原理: 随机掩码部分词, 预测被掩码的词, 如 BERT
  - 优点: 可以同时利用双向上下文, 适合理解任务
  - 缺点: 不适合直接生成, 需要额外的生成头

### 2. Few-shot Learning vs In-context Learning:

- **Few-shot Learning:**
  - 定义: 在少量样本上微调模型参数
  - 需要更新模型权重
  - 需要梯度计算和反向传播
- **In-context Learning:**
  - 定义: 通过在提示中包含示例, 让模型学习任务模式
  - 不需要更新模型参数
  - 通过注意力机制从示例中学习
  - 更灵活, 但效果可能不如微调

### 3. LoRA 如何实现参数高效微调:

- **低秩分解:** 将权重更新  $\Delta W$  分解为两个低秩矩阵的乘积:  $\Delta W = BA$ , 其中  $B \in \mathbb{R}^{d \times r}$ ,  $A \in \mathbb{R}^{r \times k}$ ,  $r \ll \min(d, k)$
- **参数量减少:** 从  $d \times k$  减少到  $r \times (d + k)$ , 通常  $r = 8$  或  $16$
- **训练过程:** 冻结原始权重  $W$ , 只训练  $A$  和  $B$ , 前向传播为  $h = Wx + BAx$
- **优势:** 大幅减少可训练参数, 降低内存需求, 可以快速适配多个任务

### 4.3 提示工程

#### 1. Chain-of-Thought 如何提高模型的推理能力:

- **逐步推理**: 将复杂问题分解为多个简单步骤, 引导模型逐步思考
- **中间过程**: 展示推理的中间步骤, 让模型学习逻辑推理模式
- **示例**: 对于数学问题, 先展示计算过程, 再给出答案
- **效果**: 显著提高复杂推理任务的准确率, 特别是在数学、逻辑推理等领域

#### 2. Few-shot Prompting 的工作原理:

- **示例学习**: 在提示中包含几个输入-输出示例, 让模型学习任务模式
- **模式识别**: 模型通过注意力机制识别示例中的模式, 并应用到新输入
- **不需要微调**: 不需要更新模型参数, 只需要设计好的提示
- **适用场景**: 任务模式清晰、示例质量高的场景

#### 3. 如何设计有效的提示:

- **明确任务**: 清晰说明任务类型和要求
- **提供示例**: 包含高质量、多样化的示例
- **结构化格式**: 使用清晰的格式 (如列表、步骤) 组织提示
- **角色设定**: 为模型设定合适的角色 (如“你是一个专家”)
- **输出格式**: 明确指定期望的输出格式

### 4.4 RAG

#### 1. RAG 相比直接生成有什么优势:

- **知识更新**: 可以访问最新的外部知识, 不受训练数据时间限制
- **减少幻觉**: 基于检索到的真实文档生成, 减少编造信息
- **可解释性**: 可以追溯到源文档, 提高可解释性
- **领域适应**: 可以快速适配新领域, 只需更新知识库

#### 2. 如何构建高效的检索系统:

- **文档分块**: 将长文档分割为适当大小的块 (如 256 或 512 tokens)
- **向量化**: 使用嵌入模型将文档块转换为向量
- **索引构建**: 使用向量数据库 (如 FAISS、Milvus) 构建索引

- **检索策略**: 使用相似度搜索（如余弦相似度）检索相关文档
- **重排序**: 使用更精确的模型对检索结果重排序

### 3. RAG 如何解决大模型的幻觉问题:

- **基于事实**: 生成内容基于检索到的真实文档，而不是仅依赖模型记忆
- **引用机制**: 可以引用源文档，用户可以验证信息
- **知识库管理**: 通过维护高质量知识库，确保信息来源可靠
- **混合生成**: 结合检索到的信息和模型知识，生成更准确的内容

## 5 编程题答案

### 5.1 实现 Transformer 编码器

```
import torch
import torch.nn as nn
import math

class TransformerEncoder(nn.Module):
    def __init__(self, vocab_size, d_model=512, nhead=8,
                 num_layers=6, dim_feedforward=2048,
                 max_seq_length=5000, dropout=0.1):
        super().__init__()

        self.d_model = d_model
        # 词嵌入
        self.embedding = nn.Embedding(vocab_size, d_model)
        # 位置编码
        self.pos_encoding = PositionalEncoding(d_model, max_seq_length)

        # Transformer 编码器层
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=d_model,
            nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout,
            activation='relu',
            batch_first=True
```

```
)  
self.transformer_encoder = nn.TransformerEncoder(  
    encoder_layer,  
    num_layers=num_layers  
)  
  
# 分类头  
self.classifier = nn.Linear(d_model, num_classes)  
self.dropout = nn.Dropout(dropout)  
  
def forward(self, x, mask=None):  
    # x: (batch_size, seq_len)  
    # 词嵌入和位置编码  
    x = self.embedding(x) * math.sqrt(self.d_model)  
    x = self.pos_encoding(x)  
  
    # Transformer 编码  
    x = self.transformer_encoder(x, src_key_padding_mask=mask)  
  
    # 池化 (使用 [CLS] token 或平均池化)  
    x = x.mean(dim=1)  # (batch_size, d_model)  
    x = self.dropout(x)  
  
    # 分类  
    output = self.classifier(x)  
    return output  
  
# 使用示例  
model = TransformerEncoder(  
    vocab_size=10000,  
    d_model=512,  
    nhead=8,  
    num_layers=6,  
    num_classes=2  
)
```

Listing 1: Transformer 编码器完整实现

## 5.2 实现注意力机制

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import math

def scaled_dot_product_attention(Q, K, V, mask=None):
    """
    缩放点积注意力

    参数:
        Q: (batch_size, seq_len_q, d_k)
        K: (batch_size, seq_len_k, d_k)
        V: (batch_size, seq_len_v, d_v)
        mask: (batch_size, seq_len_q, seq_len_k) 或 None

    """
    d_k = Q.size(-1)

    # 计算注意力分数
    scores = torch.matmul(Q, K.transpose(-2, -1)) / math.sqrt(d_k)

    # 应用掩码
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)

    # Softmax 归一化
    attn_weights = F.softmax(scores, dim=-1)

    # 加权求和
    output = torch.matmul(attn_weights, V)

    return output, attn_weights

class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, num_heads):
        super().__init__()
        assert d_model % num_heads == 0

        self.d_model = d_model
```

```
    self.num_heads = num_heads
    self.d_k = d_model // num_heads

    # 共享投影矩阵
    self.W_q = nn.Linear(d_model, d_model)
    self.W_k = nn.Linear(d_model, d_model)
    self.W_v = nn.Linear(d_model, d_model)
    self.W_o = nn.Linear(d_model, d_model)

    def forward(self, Q, K, V, mask=None):
        batch_size = Q.size(0)

        # 线性投影并重塑为多头形式
        Q = self.W_q(Q).view(batch_size, -1, self.num_heads, self.d_k).
        transpose(1, 2)
        K = self.W_k(K).view(batch_size, -1, self.num_heads, self.d_k).
        transpose(1, 2)
        V = self.W_v(V).view(batch_size, -1, self.num_heads, self.d_k).
        transpose(1, 2)

        # 计算注意力
        scores = torch.matmul(Q, K.transpose(-2, -1)) / math.sqrt(self.
        d_k)
        if mask is not None:
            scores = scores.masked_fill(mask == 0, -1e9)
        attn_weights = F.softmax(scores, dim=-1)
        attn_output = torch.matmul(attn_weights, V)

        # 拼接多头
        attn_output = attn_output.transpose(1, 2).contiguous().view(
            batch_size, -1, self.d_model
        )

        # 输出投影
        output = self.W_o(attn_output)
        return output, attn_weights

# 可视化注意力权重
import matplotlib.pyplot as plt
```

```

def visualize_attention(attn_weights, tokens):
    """
    可视化注意力权重

    参数:
        attn_weights: (num_heads, seq_len, seq_len) 注意力权重
        tokens: 词列表
    """

    num_heads = attn_weights.size(0)
    fig, axes = plt.subplots(1, num_heads, figsize=(15, 3))

    for i in range(num_heads):
        ax = axes[i] if num_heads > 1 else axes
        im = ax.imshow(attn_weights[i].detach().numpy(), cmap='Blues')
        ax.set_title(f'Head {i+1}')
        ax.set_xticks(range(len(tokens)))
        ax.set_yticks(range(len(tokens)))
        ax.set_xticklabels(tokens, rotation=45)
        ax.set_yticklabels(tokens)
        plt.colorbar(im, ax=ax)

    plt.tight_layout()
    plt.show()

```

Listing 2: 缩放点积注意力和多头注意力实现

## Part IV

# 大语言模型先进技术练习题答案

## 6 概念理解题答案

### 1. LoRA 的数学原理:

- **低秩假设:** 权重更新矩阵  $\Delta W$  通常是低秩的, 即可以用两个小矩阵的乘积表示
- **数学表示:**  $\Delta W = BA$ , 其中  $B \in \mathbb{R}^{d \times r}$ ,  $A \in \mathbb{R}^{r \times k}$ ,  $r \ll \min(d, k)$
- **参数量:** 从  $d \times k$  减少到  $r \times (d + k)$ , 当  $r = 8$  时, 参数量减少约 100-1000 倍

- **有效性原因:**
  - 预训练模型已经学习了丰富的特征表示
  - 微调只需要小的调整，不需要大幅改变权重
  - 低秩分解可以捕获这些小的调整

### 2. QLoRA vs LoRA:

- **LoRA:** 在 FP16 或 FP32 精度下训练，需要较大内存
- **QLoRA:**
  - 使用 4-bit 量化存储模型权重
- **量化作用:**
  - 减少内存占用：4-bit 量化可以将模型大小减少约 4 倍
  - 允许在消费级 GPU 上微调大模型
  - 通过量化感知训练保持模型性能

### 3. Flash Attention 如何减少内存占用：

- **问题:** 标准注意力需要存储完整的注意力矩阵  $QK^T$ ，大小为  $O(n^2)$
- **解决方案:**
  - 分块计算：将序列分成块，逐块计算注意力
  - 在线 Softmax：使用在线算法计算 Softmax，不需要存储完整矩阵
  - 内存复杂度：从  $O(n^2)$  降低到  $O(n)$
- **效果:** 可以处理更长的序列，训练速度更快

### 4. BLEU 分数的简短惩罚：

- **问题:** 短句子可能获得高精确度，但质量不高
- **简短惩罚 (Brevity Penalty):**

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r \end{cases} \quad (59)$$

其中  $c$  是候选长度， $r$  是参考长度

- **作用:** 惩罚过短的候选文本，鼓励生成与参考长度相近的文本

### 5. ROUGE-L vs ROUGE-N:

- **ROUGE-N:** 基于 N-gram 重叠，如 ROUGE-1 (单词级)、ROUGE-2 (二元组级)

- **ROUGE-L**: 基于最长公共子序列 (LCS)，考虑句子级结构
- **区别**:
  - ROUGE-N 关注词序匹配
  - ROUGE-L 关注句子结构相似性，更灵活
  - ROUGE-L 对词序变化更鲁棒

## 7 计算题答案

### 7.1 困惑度计算

1. 给定条件:

- 序列长度:  $n = 100$
- 平均对数概率:  $\bar{\ell} = -2.5$

计算过程:

$$\text{Perplexity} = \exp\left(-\frac{1}{n} \sum_{i=1}^n \log P(x_i)\right) \quad (60)$$

$$= \exp(-\bar{\ell}) \quad (61)$$

$$= \exp(-(-2.5)) \quad (62)$$

$$= \exp(2.5) \quad (63)$$

$$\approx 12.18 \quad (64)$$

代码验证:

```
import numpy as np

# 给定条件
seq_length = 100
avg_log_prob = -2.5

# 计算困惑度
perplexity = np.exp(-avg_log_prob)
print(f"困惑度: {perplexity:.2f}")
# 输出: 困惑度: 12.18
```

Listing 3: 困惑度计算代码

**解释:** 困惑度约为 12.18, 表示模型平均需要在 12.18 个候选词中选择一个, 值越小表示模型越确定。

## 7.2 BLEU 计算

1. 给定条件:

- 参考: "the cat sat on the mat"
- 候选: "a cat sat on mat"

计算过程:

1-gram 精确度:

- 参考 1-grams: {the(2), cat(1), sat(1), on(1), mat(1)}
- 候选 1-grams: {a(1), cat(1), sat(1), on(1), mat(1)}
- 匹配数: 4 (cat, sat, on, mat)
- 候选总数: 5
- $P_1 = 4/5 = 0.8$

2-gram 精确度:

- 参考 2-grams: {the cat, cat sat, sat on, on the, the mat}
- 候选 2-grams: {a cat, cat sat, sat on, on mat}
- 匹配数: 2 (cat sat, sat on)
- 候选总数: 4
- $P_2 = 2/4 = 0.5$

3-gram 和 4-gram:

- $P_3 = 1/3 \approx 0.333$
- $P_4 = 0/2 = 0$

简短惩罚:

- 候选长度:  $c = 5$
- 参考长度:  $r = 6$
- $BP = e^{1-6/5} = e^{-0.2} \approx 0.819$

BLEU 分数：

$$\text{BLEU} = BP \times \exp \left( \frac{1}{4} \sum_{n=1}^4 \log P_n \right) \quad (65)$$

$$= 0.819 \times \exp \left( \frac{1}{4} (\log 0.8 + \log 0.5 + \log 0.333 + \log 0) \right) \quad (66)$$

$$= 0.819 \times \exp \left( \frac{1}{4} (-0.223 - 0.693 - 1.099 - (-\infty)) \right) \quad (67)$$

$$= 0 \quad (\text{因为 } P_4 = 0) \quad (68)$$

代码实现：

```

from collections import Counter
import math

def bleu_score(candidate, reference):
    # 分词
    cand_tokens = candidate.split()
    ref_tokens = reference.split()

    # 计算各 n-gram 精确度
    precisions = []
    for n in range(1, 5):
        # 生成 n-grams
        cand_ngrams = [' '.join(cand_tokens[i:i+n])
                       for i in range(len(cand_tokens)-n+1)]
        ref_ngrams = [' '.join(ref_tokens[i:i+n])
                      for i in range(len(ref_tokens)-n+1)]

        # 计算匹配数
        cand_counts = Counter(cand_ngrams)
        ref_counts = Counter(ref_ngrams)

        matches = sum(min(cand_counts[ngram], ref_counts[ngram])
                      for ngram in cand_counts)
        total = len(cand_ngrams)

        precisions.append(matches / total if total > 0 else 0)

    # 简短惩罚
    c, r = len(cand_tokens), len(ref_tokens)

```

```
bp = 1 if c > r else math.exp(1 - r / c)

# BLEU 分数
if any(p == 0 for p in precisions):
    return 0

bleu = bp * math.exp(sum(math.log(p) for p in precisions) /
4)
return bleu

# 测试
reference = "the cat sat on the mat"
candidate = "a cat sat on mat"
print(f"BLEU 分数: {bleu_score(candidate, reference):.4f}")
```

Listing 4: BLEU 分数计算代码

### 7.3 ROUGE 计算

#### 1. 给定条件:

- 参考: "the cat is on the mat"
- 候选: "cat mat"

#### 计算过程:

##### ROUGE-1 (单词级召回率):

- 参考单词: {the(2), cat(1), is(1), on(1), mat(1)}
- 候选单词: {cat(1), mat(1)}
- 匹配单词: {cat, mat}
- 匹配数: 2
- 参考总数: 6
- ROUGE-1 = 2/6 = 0.333

##### ROUGE-2 (二元组级召回率):

- 参考 2-grams: {the cat, cat is, is on, on the, the mat}

- 候选 2-grams: {cat mat}
- 匹配数: 0
- 参考总数: 5
- ROUGE-2 =  $0/5 = 0$

ROUGE-L (最长公共子序列):

- 最长公共子序列: {cat, mat}, 长度 = 2
- 参考长度: 6
- 候选长度: 2
- ROUGE-L =  $\frac{2}{6} = 0.333$  (基于参考)
- ROUGE-L =  $\frac{2}{2} = 1.0$  (基于候选)
- 通常使用 F1 分数:  $F1 = \frac{2 \times 0.333 \times 1.0}{0.333 + 1.0} = 0.5$

代码实现:

```

from collections import Counter

def rouge_n(candidate, reference, n=1):
    """计算 ROUGE-N 召回率"""
    # 生成 n-grams
    def get_ngrams(tokens, n):
        return [' '.join(tokens[i:i+n])
               for i in range(len(tokens)-n+1)]

    cand_tokens = candidate.split()
    ref_tokens = reference.split()

    cand_ngrams = get_ngrams(cand_tokens, n)
    ref_ngrams = get_ngrams(ref_tokens, n)

    # 计算匹配数
    cand_counts = Counter(cand_ngrams)
    ref_counts = Counter(ref_ngrams)

    matches = sum(min(cand_counts[ngram], ref_counts[ngram])
                  for ngram in cand_counts if ngram in ref_counts)

    return matches / len(ref_ngrams) if ref_ngrams else 0

```

```
def lcs_length(s1, s2):
    """计算最长公共子序列长度"""
    m, n = len(s1), len(s2)
    dp = [[0] * (n+1) for _ in range(m+1)]

    for i in range(1, m+1):
        for j in range(1, n+1):
            if s1[i-1] == s2[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])

    return dp[m][n]

def rouge_l(candidate, reference):
    """计算 ROUGE-L F1 分数"""
    cand_tokens = candidate.split()
    ref_tokens = reference.split()

    lcs_len = lcs_length(cand_tokens, ref_tokens)

    if len(ref_tokens) == 0 or len(cand_tokens) == 0:
        return 0

    precision = lcs_len / len(cand_tokens)
    recall = lcs_len / len(ref_tokens)

    if precision + recall == 0:
        return 0

    f1 = 2 * precision * recall / (precision + recall)
    return f1

# 测试
reference = "the cat is on the mat"
candidate = "cat mat"

print(f"ROUGE-1: {rouge_n(candidate, reference, 1):.4f}")
print(f"ROUGE-2: {rouge_n(candidate, reference, 2):.4f}")
```

```
    print(f"ROUGE-L: {rouge_l(candidate, reference):.4f}")
```

Listing 5: ROUGE 分数计算代码

## 8 代码实现题答案

### 8.1 LoRA 训练脚本

```
import torch
import torch.nn as nn
from transformers import AutoModel, AutoTokenizer
from peft import LoraConfig, get_peft_model, TaskType

class LoRATrainer:
    def __init__(self, model_name, lora_config):
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModel.from_pretrained(model_name)

        # 配置 LoRA
        peft_config = LoraConfig(
            task_type=TaskType.CAUSAL_LM,
            r=lora_config['r'],
            lora_alpha=lora_config['alpha'],
            lora_dropout=lora_config['dropout'],
            target_modules=lora_config['target_modules']
        )

        self.model = get_peft_model(self.model, peft_config)
        self.model.print_trainable_parameters()

    def train(self, train_data, epochs=3, batch_size=4, lr=1e-4):
        optimizer = torch.optim.AdamW(
            self.model.parameters(),
            lr=lr
        )

        self.model.train()
```

```
for epoch in range(epochs):
    total_loss = 0
    for i in range(0, len(train_data), batch_size):
        batch = train_data[i:i+batch_size]

        # 编码输入
        inputs = self.tokenizer(
            batch,
            return_tensors='pt',
            padding=True,
            truncation=True
        )

        # 前向传播
        outputs = self.model(**inputs, labels=inputs['input_ids'])
        loss = outputs.loss

        # 反向传播
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {total_loss/len(train_data):.4f}")

def evaluate(self, test_data):
    self.model.eval()
    total_loss = 0

    with torch.no_grad():
        for text in test_data:
            inputs = self.tokenizer(
                text,
                return_tensors='pt'
            )
            outputs = self.model(**inputs, labels=inputs['input_ids'])

    
```

```
        total_loss += outputs.loss.item()

    avg_loss = total_loss / len(test_data)
    perplexity = torch.exp(torch.tensor(avg_loss))

    print(f"Average Loss: {avg_loss:.4f}")
    print(f"Perplexity: {perplexity:.4f}")

    return avg_loss, perplexity.item()

# 使用示例
lora_config = {
    'r': 8,
    'alpha': 16,
    'dropout': 0.1,
    'target_modules': ['q_proj', 'v_proj']
}

trainer = LoRATrainer('gpt2', lora_config)
train_data = ["Sample text 1", "Sample text 2", ...]
test_data = ["Test text 1", "Test text 2", ...]

trainer.train(train_data)
trainer.evaluate(test_data)
```

Listing 6: 完整的 LoRA 训练脚本

## 8.2 批量困惑度计算

```
import torch
import torch.nn.functional as F

def compute_perplexity_batch(model, tokenizer, texts, batch_size=32):
    """
    批量计算困惑度

    参数:
        model: 语言模型
        tokenizer: 分词器
    
```

```
texts: 文本列表
batch_size: 批次大小

返回：
    perplexity: 平均困惑度
"""

model.eval()
total_loss = 0
total_tokens = 0

with torch.no_grad():
    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i+batch_size]

        # 编码
        inputs = tokenizer(
            batch_texts,
            return_tensors='pt',
            padding=True,
            truncation=True
        )

        # 前向传播
        outputs = model(**inputs, labels=inputs['input_ids'])
        loss = outputs.loss

        # 累计损失和token数
        batch_size_actual = inputs['input_ids'].size(0)
        seq_length = inputs['input_ids'].size(1)
        total_loss += loss.item() * batch_size_actual * seq_length
        total_tokens += batch_size_actual * seq_length

    # 计算平均对数概率
    avg_log_prob = -total_loss / total_tokens

    # 计算困惑度
    perplexity = torch.exp(torch.tensor(avg_log_prob))

return perplexity.item()
```

```
# 使用示例
texts = [
    "The cat sat on the mat.",
    "Machine learning is fascinating.",
    ...
]
perplexity = compute_perplexity_batch(model, tokenizer, texts)
print(f"Perplexity: {perplexity:.2f}")
```

Listing 7: 批量困惑度计算函数

## Part V

# Python/NumPy/Pandas 练习题答案

## 9 NumPy 综合例题答案

### 9.1 例题 1：数据预处理

```
import numpy as np

# 生成模拟数据
np.random.seed(42)
data = np.random.randn(1000, 20)

# 1. 计算每个特征的均值和标准差
means = np.mean(data, axis=0)
stds = np.std(data, axis=0)

# 2. Z-score 标准化
data_normalized = (data - means) / stds

# 3. 检测异常值 (3 原则)
outliers_mask = np.abs(data_normalized) > 3
outliers_count = np.sum(outliers_mask, axis=0)

# 处理异常值：用边界值替换
```

```
data_cleaned = data_normalized.copy()
for i in range(data_cleaned.shape[1]):
    col = data_cleaned[:, i]
    outliers = np.abs(col) > 3
    col[outliers] = np.sign(col[outliers]) * 3
    data_cleaned[:, i] = col

# 4. 重塑为适合神经网络的形状
# 假设需要 (batch_size, features) 的形状
batch_size = 32
n_batches = data_cleaned.shape[0] // batch_size
data_reshaped = data_cleaned[:n_batches * batch_size].reshape(n_batches,
    batch_size, -1)
```

Listing 8: 数据预处理完整代码

## 10 总结

本答案集提供了所有练习题的详细解答，包括：

- 数学计算的完整步骤
- 概念题的详细解释
- 编程题的代码实现
- 理论问题的深入分析

通过练习和参考答案，读者可以：

- 巩固理论知识
- 掌握计算方法
- 理解核心概念
- 提升编程能力