第一阶段：这一阶段主要是入门深度学习，了解深度学习的基本算法。这个是 markdown 笔记，文件存放在 formula 文件夹中，这里做个总结。

1 深度学习

- 本质：
  - 输入：一堆特征（数字），embedding
  - 输出与 label 做 loss，训练学习

> 线性回归

- 房价预测：
  - 输入（特征）：年份、面积、楼层...
  - 标签：房价

> 逻辑回归

- 葡萄酒分类
  - 输入（特征）：一堆数字
  - 标签：品级
- 鸢尾花分类
  - 输入（特征）：一堆数字
  - 标签：鸢尾花的种类

2 NLP

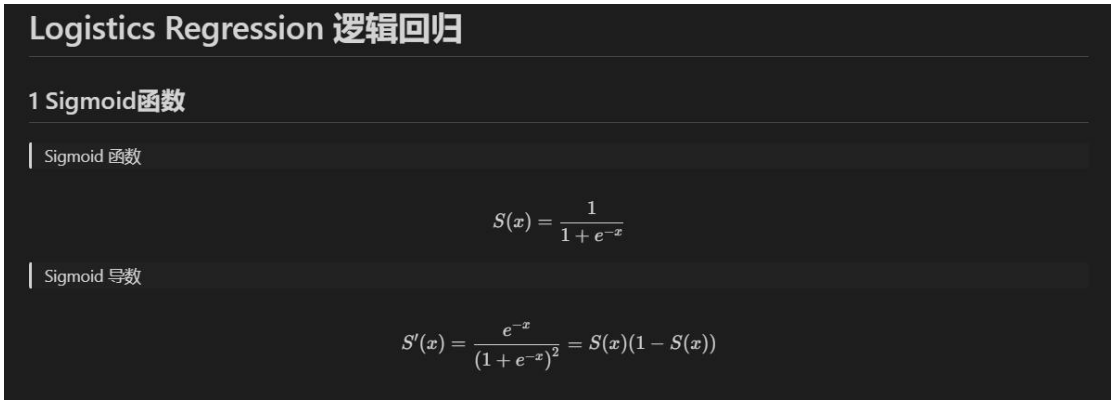
> 清华新闻数据集，文本分类

- 输入：一个新闻标题
  - Embedding: Onehot编码 -> 随机向量 -> nn.Embedding -> 是否能自己训练这个 Embedding（用别人训练好的 Embedding）
- 标签：不同的新闻类别（一共十个）

3 Word2Vec

- 动机：想要训练一个 词向量, 能够对每个词进行编码
- 重要假设：词的距离越近，越相关
- 流程：
  1. 拿到一堆的数据（语料），课上例子：数学原始数据.csv; 各种小说、文章
  2. 构建一个词表（字典）vocab, 用来索引 Embedding，拿到每个词的向量表示
  3. 构建数据集（Skip-gram）
     - win_size: 中心词两边的语境词（上下文词）
     - 负采样
       - 随机
       - 基于词频的负采样

pytorch 框架组件公式，源代码在”第一阶段学习-pytorch 自主复现组件“



Logistics Regression 逻辑回归

1 Sigmoid函数

> Sigmoid 函数

$$S(x) = \frac{1}{1+e^{-x}}$$

> Sigmoid 导数

$$S'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = S(x)(1-S(x))$$

2 BCE（Binary Cross Entropy）二元交叉熵

2.1 单条数据

$$\text{Loss} = -[y \cdot \log(p(y)) + (1-y) \cdot \log(1-p(y))]$$

- y: label
- $p(y)$: predict

> 猜硬币正反面的游戏，不是猜 0 或者 1, 猜 (0, 1) 之间的任意小数

> 当答案是 1 的情况

$$\text{Loss} = -\log(p(y))$$

- 选手1：p(y) = 0.99999, Loss: ->0
- 选手2：p(y) = 0.5, Loss: 0.6931471805599453
- 选手3：p(y) = 0.5, Loss: 0.6931471805599453
- 选手4：p(y) = 0.49, Loss: 0.7133498878774648

> 当答案是 0 的情况

$$\text{Loss} = -\log(1-p(y))$$

- 选手1：p(y) = 1e-5, Loss: 1.0000050000287824e-05
- 选手2：p(y) = 0.5, Loss: 0.6931471805599453
- 选手3：p(y) = 0.5, Loss: 0.6931471805599453
- 选手4：p(y) = 0.51, Loss: 0.7133498878774648

2.2 Batch 条数据

- N = batch size

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1-y_i) \cdot \log(1-p(y_i))$$

3 BCE & Sigmoid联合求导

$$\frac{\partial \text{BEC}}{\partial z} = p - y$$

Word2vec 复习：

Word2Vec

1 Word2Vec Intro

- Word2Vec 产生的动机：训练词向量，得到一个能够用来表示词的向量矩阵（支持词按索引得到每个词的向量表示）
- 两个模型
  - CBoW: 周围词预测中心词
  - Skip-gram: 中心词预测周围词
- 重点
  - 如何去构建数据集
  - 如何去构建任务

2 Raw Word2Vec

任务的构建

- 多分类任务，类别为：len(vocab)
- 直观理解
  - 取某个词的向量
  - 然后跟一个矩阵（所有词的矩阵）做运算，得到 len(vocab)（类别）的一个值
  - 经过 softmax 转化成概率表示，这个概率的含义就是与 vocab 内每个词的相关程度

数据集的构建

- 格式: (word, another_word)
- inputs:
  - CBoW: word = 周围词
  - Skip-gram: word = 中心词
- labels:
  - CBoW: another_word = 中心词
  - Skip-gram: another_word = 周围词

3 改进之后的版本

任务的构建

- 二分类任务，类别为：0（不相关）；1（相关）
- 直观的理解
  - 取 word 的向量表示
  - 取 another_word 的向量表示
  - 两者做向量乘（相似度）的运算
  - 输出的值，经过 sigmoid 转化成 0-1 的一个值

数据集的构建

- 格式：(word, another_word, label)
- CBoW:
  - input: word = 周围词，another_word = 中心词
  - label: label
- Skip-gram:
  - input: word = 中心词，another_word = 周围词
  - label: label

第二阶段：复现经典框架：RNN 系列、Transformer、Bert、GPT
Tokenize 复习：

Intro

1 分词（Tokenization）

将文本、语料转换成 token（按字分、按词分），输出

- vocab: 在 encode（编码） 的过程使用
- idx2word: 在 decode（解码）的过程使用

> OOV（Out of Vocabulary）

- 不在词表内
- `[UNK]` 记录一些不在词表内的词

2 Embedding

2.1 过程

> 将 token 转换成向量化表示

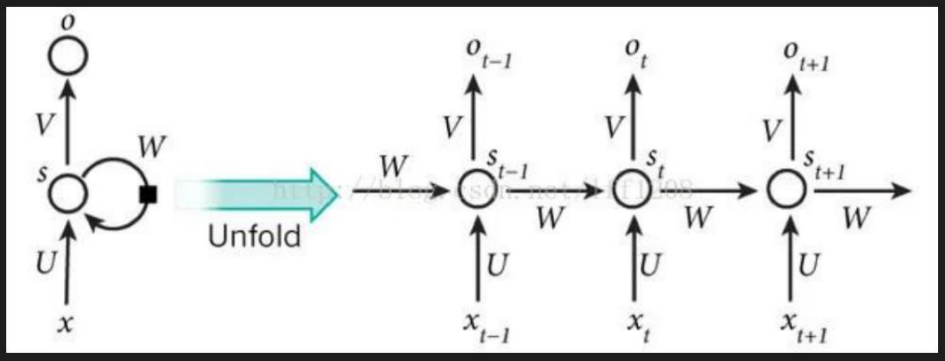
1. 从 vocab 拿到 token 的对应 id
2. 用 id 对 `某个Embedding` 进行索引，拿到 token 的向量表示

2.2 Embedding 种类

1. One-Hot: 稀疏向量（矩阵）会浪费存储空间;
2. Random Vector: 固定的、随机的向量
3. nn.Embedding: PyTorch 里面可学习的 Embedding

RNN, LSTM, GRU 复习:

RNN系列



1 RNN 循环神经网络

$$h_t = \tanh(xW_x + h_{t-1}W_h)$$

- 输出:
  - 最后一个隐藏状态 h_t: 可以用来做文本分类
  - outputs: 前面所有时间步（token）记录的 h_t。可以用来做文本生成、机器翻译、文本分类（需要进行一定的处理）

2 LSTM 门机制

- 门：值域：(0, 1), 表示某个信息的留存程度

> 遗忘门

$$f_t = \sigma(xU_f + h_{t-1}W_f)$$

> 输入门

$$i_t = \sigma(xU_i + h_{t-1}W_i)$$

> 细胞状态

- 候选细胞状态

$$\hat{c}_t = \tanh(xU_c + hW_c)$$

- 计算新的细胞状态

$$c_t = f_t \odot c_{t-1} i_t \odot \hat{c}$$

> 输出门

$$o_t = \sigma(xU_o + h_{t-1}W_o)$$

$$h_t = o_t \tanh(c_t)$$

3 GRU

Reset Gate

$$r_t = \sigma(U_r x_t + W_r h_{t-1})$$

Update Gate

$$z_t = \sigma(U_z x_t + W_z h_{t-1})$$

输出

$$c_t = \tanh(U_n x_t + r_t * (W_n h_{t-1}))$$

$$h_t = (1 - z_t) * c_t + z_t * h_{t-1}$$

4 Seq2Seq

- 翻译任务
- Encoder 和 Decoder 的结构，他们分别是一个 RNN
  - Encoder 对 src 进行 encode
    - 输出：h_t 给 Decoder 用
  - Decoder 对 tgt 进行 decode

5 文本生成（RNN）

5.1 使用 RNN 的输出 outputs

- shape: (batch_size, seq_len, embedding_dim)
- 每次使用的是 seq_len 维度的最后一个

5.2 使用 RNN 的 h_x 参数

- 将上一个 token 过 Model 出来的 hx 塞入到下一个 token 的 hx（包含在模型的 forward 函数里）参数里面

5.3 生成的方式

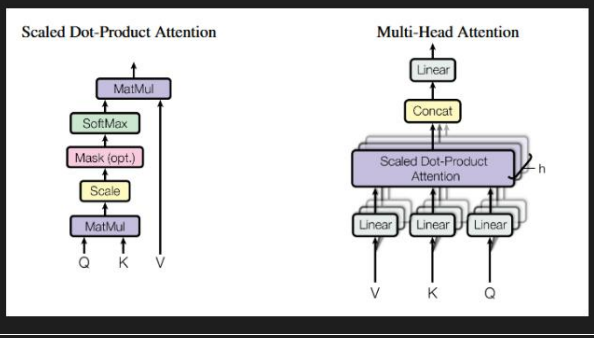
1. 贪婪算法/贪心算法: 选最大概率的那个 token
2. top-k: 从前 k 个里面随机选一个
3. top-p: 累积概率超过 p 就丢弃，从剩下里面随机选一个
4. temperature: 用 $\mathrm{softmax}(logits/temperature)$，按概率分布随机选一个

Transformer 复现：

Transformer 系列

1 Transformer

1.1 Multi-Head Attention



Multi-Head Attention

```python
class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, num_heads, dropout=0.1):
        super().__init__()
        self.d_k = d_model // num_heads
        self.q_proj = nn.Linear(d_model, d_model)
        self.k_proj = nn.Linear(d_model, d_model)
        self.v_proj = nn.Linear(d_model, d_model)
        self.o_proj = nn.Linear(d_model, d_model)

        self.dropout = nn.Dropout(dropout)


    def forward(self, x, mask=None):
        # 进来做 projection，并 reshape，划分多头
        batch_size, seq_len, _ = x.shape
        Q = self.q_proj(x).view(batch_size, seq_len, num_heads, self.d_k).transpose(1, 2)
        K = self.k_proj(x).view(batch_size, seq_len, num_heads, self.d_k).transpose(1, 2)
        V = self.v_proj(x).view(batch_size, seq_len, num_heads, self.d_k).transpose(1, 2)


        # 计算 attention scores
        attention_scores = (Q @ K.transpose(-1, -2)) / math.sqrt(self.d_k)

        # 是否应用 mask
        if mask is not None:
            # 调整维度
            attention_scores = attention_scores.masked_fill(mask==0, -1e9)
```

```python
        # softmax
        attention_scores = torch.softmax(attention_scores, dim=-1) # shape(batch_size, num_heads, seq_len,
seq_len)


        # 与 V 相乘
        output = attention_scores @ V # shape(batch_size, num_heads, seq_len, d_k)

        # reshape 回去，把多头合并回去
        output = output.transpose(1, 2).contiguous().view(batch_size, seq_len, d_model)

        # 最后输出的 projection

        output = self.o_proj(output)

        # dropout
        return self.dropout(output)
```

1.2 Layer Normalize

- 对 d_model 维度进行求均值和标准差
- 注意除零操作，需要加一个 esp=1e-5
- 可学习的参数
- 如果使用 nn.LayerNorm, 传入的初始化参数是 d_moel

1.3 Positional Encoding

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{\text{model}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right)$$

- PE 的两个维度
  - pos: 取值范围 [0, seq_len]
  - 2i/2i+1: d_model 的奇数偶数位置
- 为了防止溢出，对 sin 和 cos 里面的项进行了一定的数学处理

2 GPT

- 使用 Transformer Decoder-Only 结构

3 BERT

3.1 两个 Task

3.1.1 Pre-Training

1 MLM

15% 的概率, 进行以下处理:

- 80% 概率: 用 [MASK] 替换 token（不包含特殊 token, 如：[CLS] [SEP]）
- 10% 概率: 用随机的 token 替换原来的 token
- 10% 概率: 保持原样

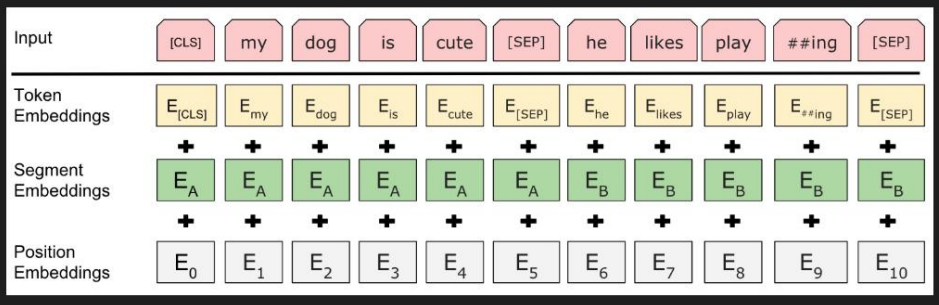
2 NSP

- 正采样 50%：真实的下一句
- 负采样 50%：随机的一句

3.1.2 Fine-Tuning

- 在特定的领域或者业务数据，基于原来的模型进行再训练

3.2 Embedding



Token Embedding

- 已经包含了 MLM 任务里面的 token

Segment Embeddings

- 怎么划分前后句
- 注意 nn.Embedding 的初始化维度是 (2, d_model)

Position Embeddings

- 注意 nn.Embedding 的初始化维度是（seq_len, d_model）

第三阶段：RAG 美食项目

在第三阶段学习了：大模型的微调，主要是意图识别和实体识别。然后就是整个 RAG 项目的搭建。源代码可查看美食 RAG 项目文件。
名称：基于 RAG 的厨房问答系统

1、数据集：data/recipe_corpus_full.json
一共大概 150 万数据
数据实例：
{"name": "西班牙金枪鱼沙拉", "dish": "金枪鱼沙拉", "description": "", "recipeIngredient": ["超市罐头装半盒金枪鱼(in spring water)", "2 大片生菜", "5 个圣女果", "半根黄瓜", "半个红柿椒", "半个紫洋葱", "1 个七成熟水煮蛋", "适量红酒醋", "适量胡椒", "适量橄榄油"], "recipeInstructions": ["鸡蛋进水煮，七成熟捞出（依个人喜好），同时备其他菜", "生菜撕片，圣女果开半，黄瓜滚刀，红柿椒切丝，紫洋葱切丝，鸡蛋四均分", "金枪鱼去水", "撒黑胡椒，红酒醋和少许橄榄油", "拌匀，拍照，开动"], "author": "author_67696", "keywords": ["西班牙金枪鱼沙拉的做法", "西班牙金枪鱼沙拉的家常做法", "西班牙金枪鱼沙拉的详细做法", "西班牙金枪鱼沙拉怎么做", "西班牙金枪鱼沙拉的最正宗做法", "沙拉"]}

2、milvus 安装

这里用的是单节点的安装，这里租了一个服务器（Linux 系统）来部署 milvus 数据库，阿里云服务器（轻量应用级服务器），由于阿里云的防火墙优先级高于服务器本身的 system，所以要在阿里云防火墙设置可出入的 id，这里我是通过部署 Docker，在 Docker 里面拉 milvus 镜像，这里记得换镜像，否则拉取镜像不成功。可用镜像：Daocloud 镜像加速器地址：https://docker.m.daocloud.io

3、milvus 数据保存

这里我将每个菜的 name 向量化存在 milvus 中，同时每个都有其对应的 id，由于 milvus 是向量数据库，能够快速的检索，同时其本身具有语义理解能力，所以检索的准确率挺高的。这里也是封装了一个类来管理 milvus 数据库。但是其安全性和稳定性都不理想，所以这里另外使用两个数据库来共同存储数据。

4、neo4j 保存数据

这里需要将数据处理一下，主要是存储了两种节点，一种是菜谱，一种是食材，由于原始数据集是没有将基本的食材提取出来的，所以这里选用微调过后的实体识别模型，将 recipeIngredient 里的实体识别出来。然后再建立菜谱和食材的关系。

在 neo4j 使用中有个文件 neo4j-oop.py 里面封装了 Neo4jManager 类，实现了增删改查基本操作，都是 cyper 语言，这样速度会快一点。

5、gradio 界面

这个是一个简单的问答界面，一键启动就可以了。

6、意图识别

对用户提出的问题进行意图识别，这里有数据的增广，一共是 4 个意图，分别是”制作方法“、”材料制作成品“、”询问配料“、”询问厨师“，标签分别对应 0，1，2，3，如果不是这四种意图就返回”其他意图“。这里同样也是进行模型的微调，只是这里自己写得一个框架，使用的预训练模型是“bge-base-zh-v1.5”

7、实体识别

对用户提出的问题进行实体识别，这里其实用调用了大模型接口来进行实体识别的，也可以进行模型的微调。但是数据集没有找到，所以就没有做，如果有数据集就可以全量微调模型。

8、项目整体代码

前面的模块都是前期的准备，实际上项目运行所需的代码在”项目总结“文件夹中，只需运行这里的文件就行了，然后启动前端 gradio 和后端 flask，即可进行问答，前提是数据库要开起来。

总之，系统的学习 NLP 知识之后，对于大模型的认知不仅仅是局限于 copy 人家的代码，我可以根据自己的需求改动模型框架，从而完成任务。未来还有很多需要学习，继续保持学习的心态，脚踏实地地努力吧！

现有流行的工具（自己的泛化了解）

还可以用 VLLM 来进行部署，但是这里最好用 GPU，也可以用 llamafactory 来进行模型微调，都是工具，不过这些框架都进行了优化，是一种很好的工具，接受新工具。还有很多是使用工具流来运行，比如 RAGflow，这个挺火的，但是这个有个缺点就是不能自定义，但是优点就是能够快速地部署好 RAG，而且一些还提供了很好的可视化工具。