NLP-beginner Report

- 本文档为https://github.com/FudanNLP/nlp-beginner中task1-5的Report，同时也是写给自己的NLP入门教程
- 使用nlp-beginner-finish作为baseline
- Written by *SayaGugu*
- 参考资料：
    - 深度学习上手指南
    - 神经网络与深度学习
    - Pytorch中文官方教程
    - Pytorch中文文档
    - TensorFlow官方文档
    - Torchtext官方文档
    - google/stackoverflow/CSDN/知乎/百度

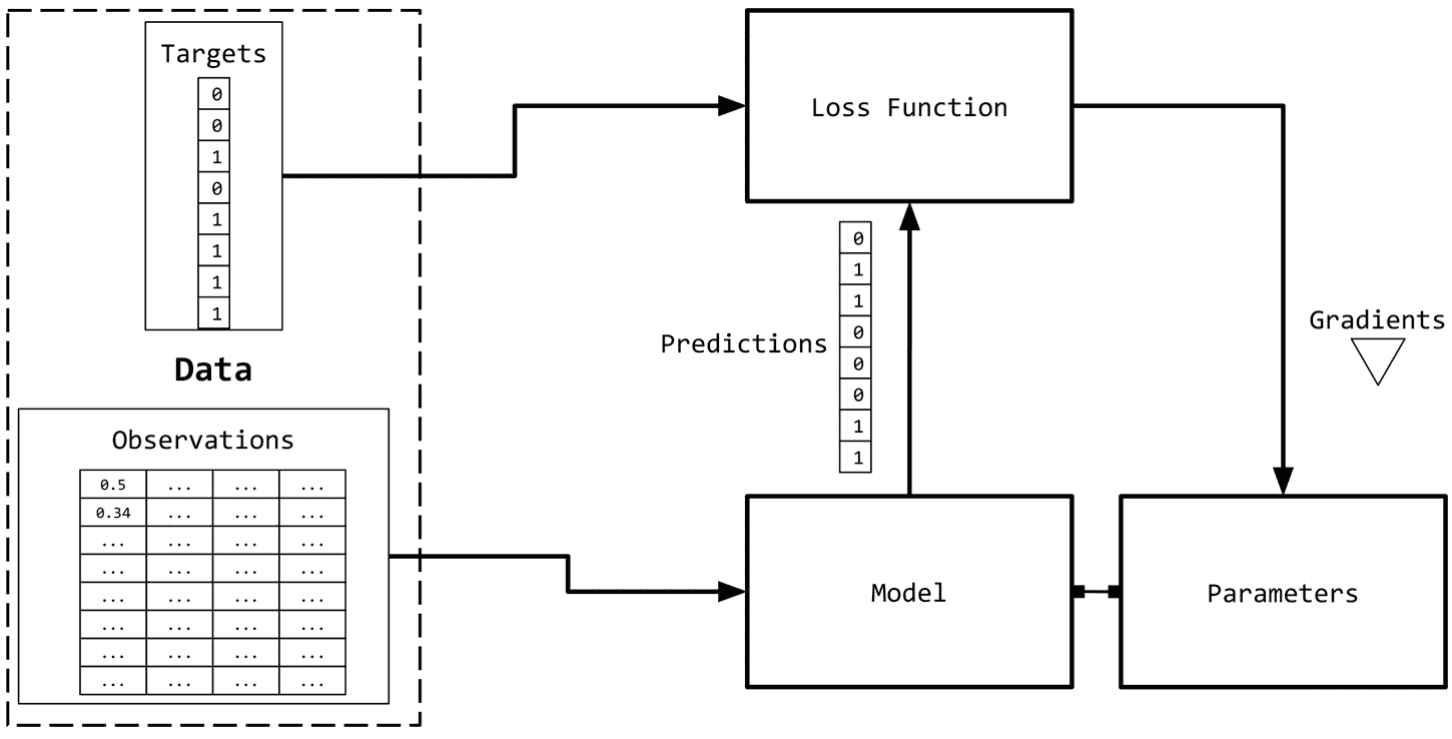
Task1： 基于机器学习的文本分类

任务

实现基于Softmax回归的文本分类

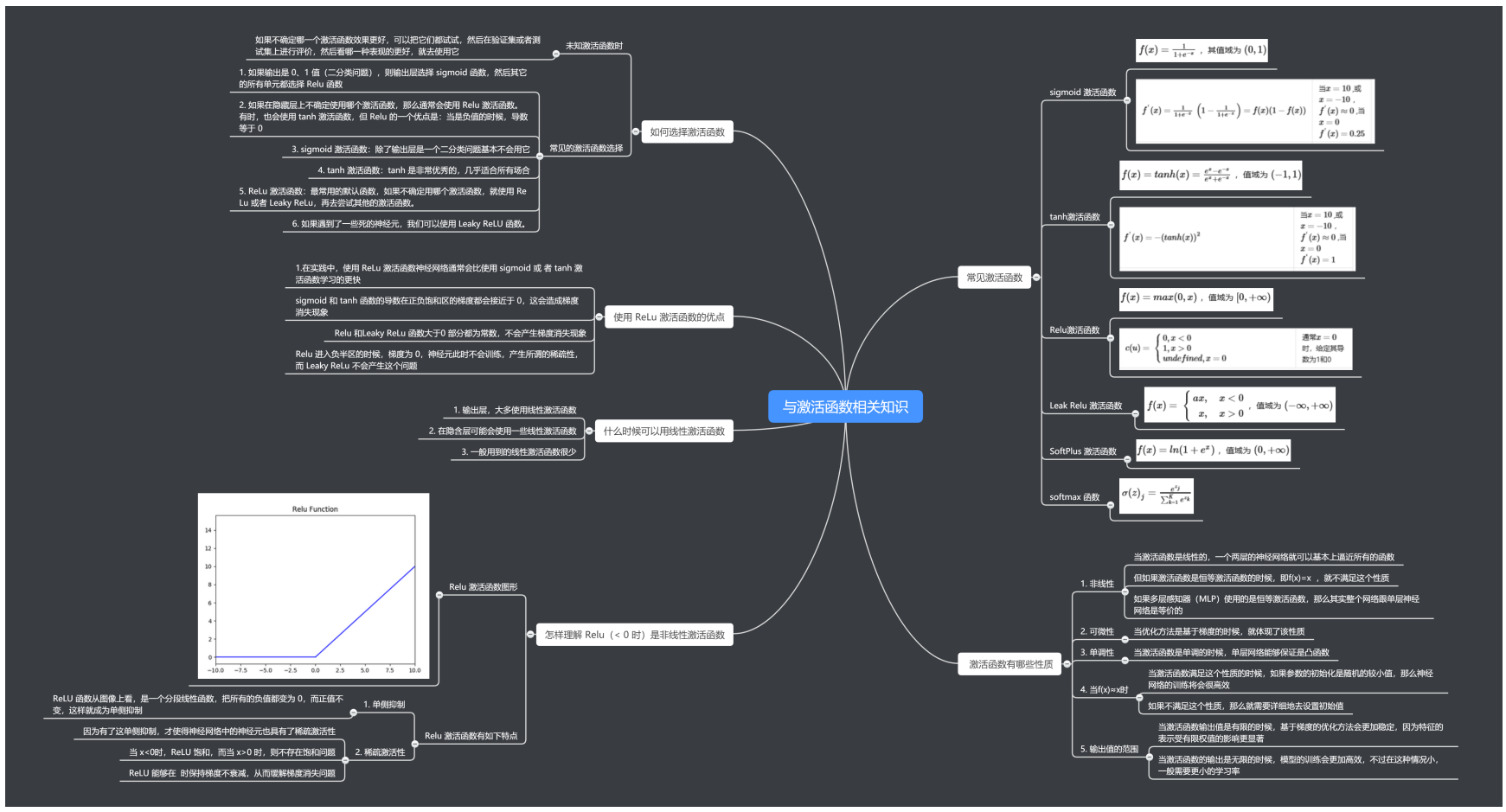
预备知识

- **监督学习范式**：监督学习，是指将目标（被预测的内容）的真实情况用于观察（输入）的情况。例如，在文档分类中，目标是一个分类标签，观察（输入）是一个文档。一个典型的范式如下图：



- **文本编码**：在 `NLP` 任务中，需要用数字表示观测值（文本），常见的编码方式如下：
    - `One-hot` **编码**：`One-Hot` 编码是分类变量作为**二进制向量**的表示。这首先要求将分类值映射到整数值。然后，每个整数值被表示为二进制向量，除了整数的索引之外，它都是零值，它被标记为1。
    - `TF-IDF`：`TF-IDF` 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。`TF-IDF` 的主要思想是：如果某个单词在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。
- **分词**：将文本分解为标记（Tokens）的过程称为分词（tokenization）。更详细的内容将在 `task2` 中出现。
- **特征提取**：
    - `bag of word`：在一句话中，存在于句子的单词（不区分大小写），则对应的向量位置上的数字为1，反之为0，通过这种方式，可以把一个句子变成一个由数字表示的0-1向量。虽然转换方式非常简单，但是它没有考虑词序
    - `N-gram`：N 元组是文本中出现的固定长度（ `n` ）的连续标记序列。二元组有两个标记，一元组只有一个标记。

- **激活函数**：激活函数是神经网络中引入的非线性函数，用于捕获数据中的复杂关系。本task中选用`Softmax`函数作为激活函数。
    - 一些常用的激活函数可见于下图：



    - `Softmax`函数简介：

在 logistic 回归中，我们的训练集由m个已标记的样本构成：$\left\{(x^{(1)},y^{(1)}),(x^{(m)},y^{(m)})\right\}$，其中输入特征$x^{(i)}\in\Re^{n+1}$。（我们对符号的约定如下：特征向量x的维度为n+1，其中$x_0=1$对应截距项。） 由于 logistic 回归是针对二分类问题的，因此类标记$y^{(i)}\in\{0,1\}$。假设函数(hypothesis function) 如下：

$$h_\theta(x)=\frac{1}{1+\exp(-\theta^T x)}$$

我们将训练模型参数θ，使其能够最小化代价函数：

$$J(\theta)=-\frac{1}{m}\left[\sum_{i=1}^{m}y^{(i)}\log h_\theta(x^{(i)})+(1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

在 softmax回归中，我们解决的是多分类问题（相对于 logistic 回归解决的二分类问题），类标y可以取k个不同的值（而不是 2 个）。因此，对于训练集$\{(x^{(1)},y^{(1)}),\ldots,(x^{(m)},y^{(m)})\}$，我们有$y^{(i)}\in\{1,2,\ldots,k\}$。（注意此处的类别下标从 1 开始，而不是 0）。例如，在 MNIST 数字识别任务中，我们有$k=10$个不同的类别。

对于给定的测试输入x，我们想用假设函数针对每一个类别估算出概率值$p(y=j|x)$。也就是说，我们想估计x的每一种分类结果出现的概率。因此，我们的假设函数将要输出一个k维的向量（向量元素的和为1）来表示这k个估计的概率值。 具体地说，我们的假设函数$h_\theta(x)$形式如下：

$$h_\theta(x^{(i)})=\begin{bmatrix}p(y^{(i)}=1|x^{(i)};\theta)\\p(y^{(i)}=2|x^{(i)};\theta)\\\vdots\\p(y^{(i)}=k|x^{(i)};\theta)\end{bmatrix}=\frac{1}{\sum_{j=1}^{k}e^{\theta_j^T x^{(i)}}}\begin{bmatrix}e^{\theta_1^T x^{(i)}}\\e^{\theta_2^T x^{(i)}}\\\vdots\\e^{\theta_k^T x^{(i)}}\end{bmatrix}$$

其中$\theta_1,\theta_2,\ldots,\theta_k\in\Re^{n+1}$是模型的参数。请注意$\frac{1}{\sum_{j=1}^{k}e^{\theta_j^T x^{(i)}}}$这一项对概率分布进行归一化，使得所有概率之和为 1。

为了方便起见，我们同样使用符号θ来表示全部的模型参数。在实现Softmax回归时，将θ用一个$k\times(n+1)$的矩阵来表示会很方便，该矩阵是将$\theta_1,\theta_2,\ldots,\theta_k$按行罗列起来得到的，如下所示：

$$\theta=\begin{bmatrix}---\theta_1^T---\\---\theta_2^T---\\\vdots\\---\theta_k^T---\end{bmatrix}$$

- **损失函数**：损失函数（loss function）是用来估量你模型的预测值f(x)与真实值Y的不一致程度，它是一个非负实值函数,通常使用L(Y, f(x))来表示，损失函数越小，模型的鲁棒性就越好。本task中选用交叉熵作为损失函数

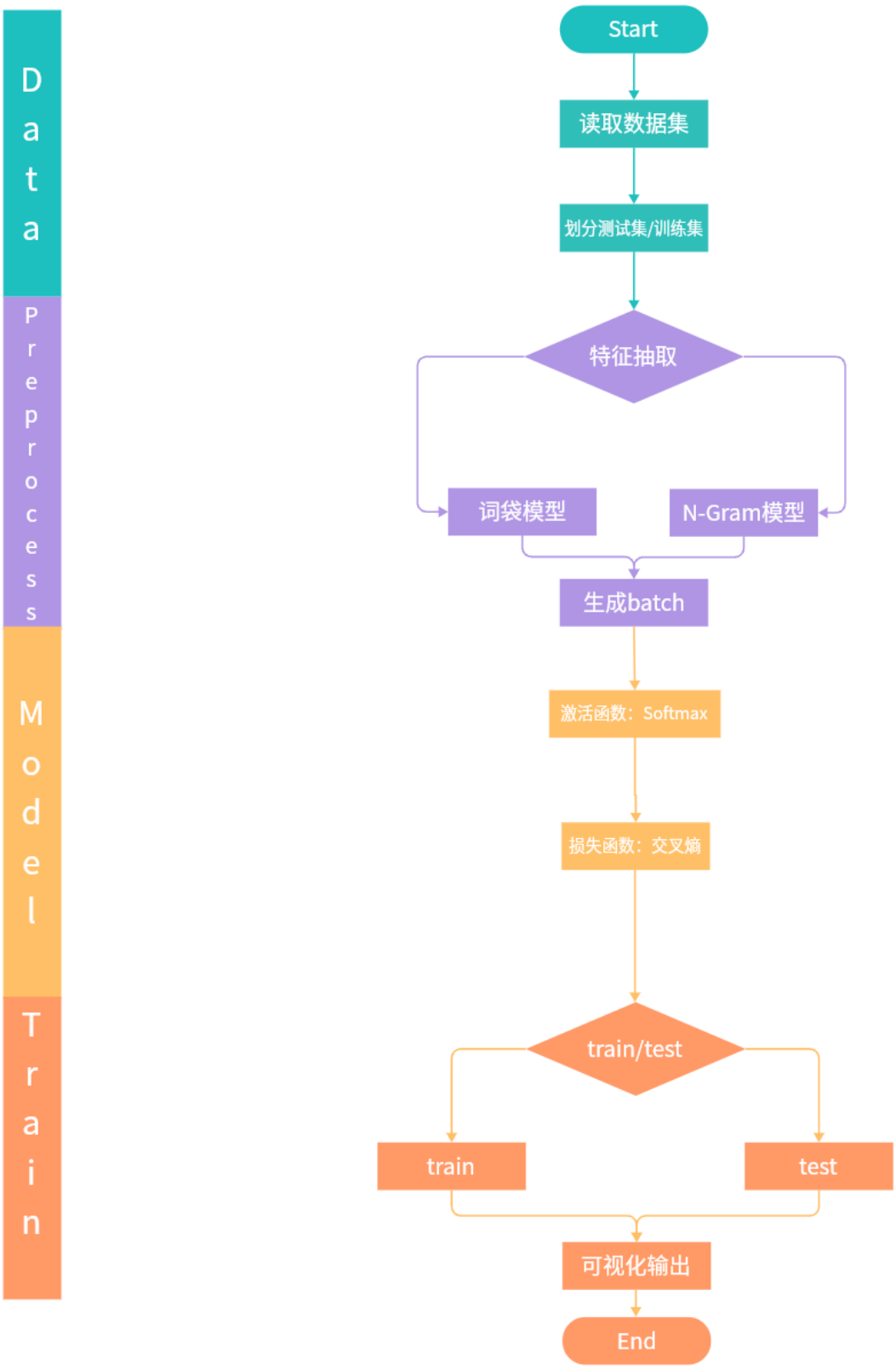
数据集

- `Classify the sentiment of sentences from the Rotten Tomatoes dataset`：从烂番茄数据集中对句子的情感进行分类 。
- `train.tsv` 包含短语及其关联的情绪标签。还提供了一个 SentenceId，以便跟踪哪些短语属于单个句子。
- `test.tsv` 只包含短语。必须为每个短语分配一个情绪标签。

- 情绪标签包括：
    - `0 - 负`
    - `1 - 有点负`
    - `2 - 中性`
    - `3 - 有点正`
    - `4 - 正`

任务思路

- 由于`task2`才要求使用Pytorch框架，可以默认`task1`中要求不使用Pytorch框架。在此情况下，baseline使用了`TensorFlow`框架，本人实现则没有使用框架，仅使用了`numpy`包
- `task1`流程图如下：



实验设置

- `dataset_size`: 1000
- `train_size`: `test_size` = 7:3
- `learning_rate`: 0.001
- `batch_size`: 10

实验结果

baseline

用了tensorflow框架，使用了一层tensorflow提供的naive的Softmax层，Loss函数选用了交叉熵。注意到baseline的task1中使用的数据集与nlp-beginner中提供的数据集不同，实际上baseline在task1中使用了task2的数据集（是一个十分类的文本分类数据集）。

在训练集上的准确度达到了`95%`，但是没写测试函数（~~本人也没有帮它补完~~）。不过由于只过了一个Softmax层，推测会过拟合。

my model

和baseline基本思路相同，手写了一个Softmax函数，并且使用N-Gram模型，在训练集上的准确度将近`100%`。但过拟合很严重，在测试集上的准确度在`50%`左右。说明目前的模型过小、可学习参数过少，如果增大模型深度、增多可学习参数，相信可以改善这个问题。

task2：基于深度学习的文本分类

任务

熟悉Pytorch，用Pytorch重写《任务一》，实现CNN、RNN的文本分类

预备知识

- **词嵌入（word embedding）**：“嵌入”是指学习从一种离散类型到向量空间中的一点的映射。当离散类型为词时，密集向量表示称为词嵌入（word embedding）。相比于one-hot编码，进行embedding可以有效降低输入的维度，从而提升
    - 随机embedding：在embedding时随机初始化。这种方法可能会生成交劣的初值，也没有良好的解释性。
    - 预训练的embedding进行初始化：使用预训练好的模型进行初始化，比如glove和bert等。本task中选用了glove模型，生成100维向量
- **CNN & RNN & LSTM**：
    - `CNN`：即卷积神经网络。卷积神经网络主要由这几类层构成：输入层、卷积层、池化层和全连接层，在CV领域应用较多
    - `RNN`：即循环神经网络。特点是RNN的隐藏层的输出会存储在内存中，当下次输入数据时会使用到内存中存储的上次的输出。由于其具有“记忆性”，因此更适合处理序列数据，在NLP领域中的应用较多
    - `LSTM`：一种特殊的RNN。当遇到长序列输入时，RNN由于其“短期记忆”的特点，易发生梯度消失。LSTM正是为了避免这一现象而被创造的。
- **dropout**：Dropout是用来防止模型过拟合的一种手段，我们常见的其他防止模型过拟合的方法还有正则化、数据增强等。

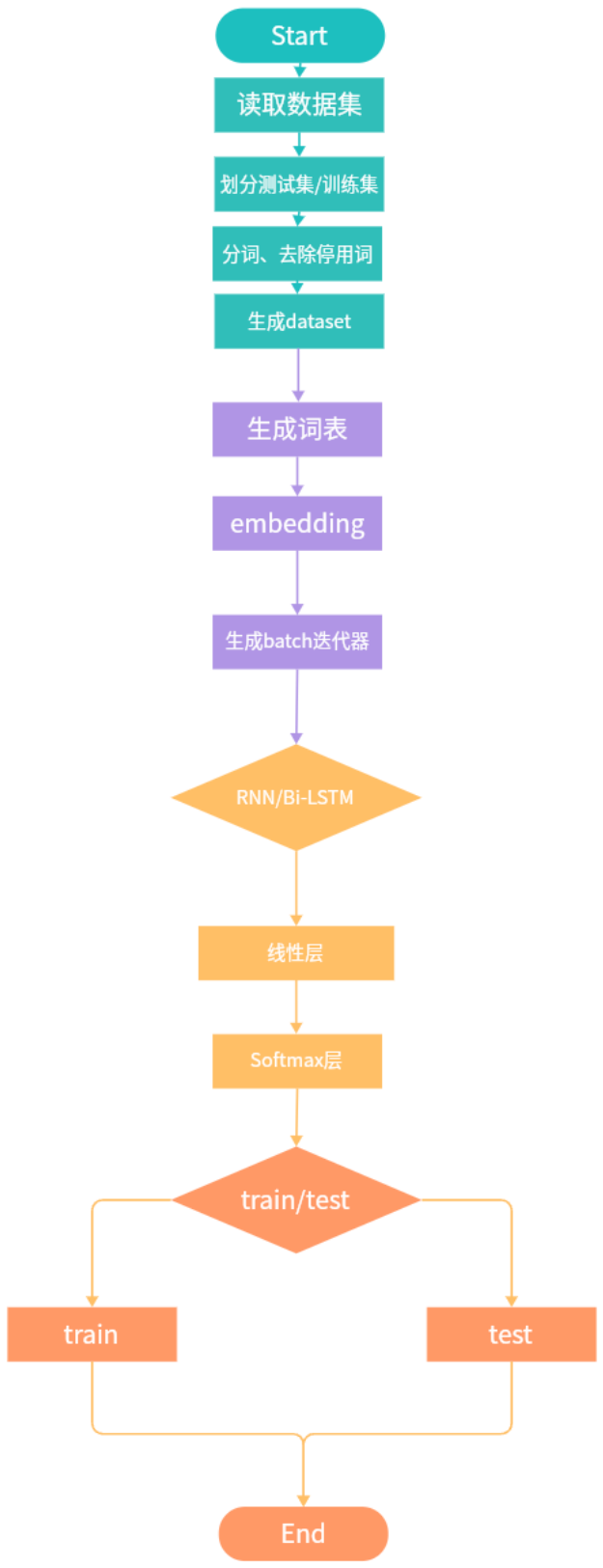
数据集

- 理论上应该是用与task1相同的数据集，然而由于baseline使用了另一个数据集，本人在写的时候没有注意，就也使用了baseline的数据集
- 使用了gaussic的数据集
- 是一个新闻数据集，由新闻文本content和标签label两列组成
- 标签包括：体育|娱乐|家居|房产|教育|时尚|时政|游戏|科技|财经

任务思路

- 使用pytorch框架
- 使用了`torchtext`包来进行便捷的dataset生成、词表生成、分词，词嵌入以及batch生成(然而torchtext的官方文档很新手不友好)
- 词嵌入使用了预训练好的`glove`模型
- 使用`LSTM`来避免梯度消失的问题
- 认为`CNN`效果理应不佳，于是只尝试了`naive RNN`和`Bi-LSTM`模型

- `task2`流程图如下：



实验设置

- `embeddind_dim`：100
- `batch_size`:64
- `train_dataset_size`：`validation_dataset_size`=9:1
- `epochs`:100
- `learning_rate`:0.001

实验结果

baseline

使用`Pytorch`框架，随机embedding和双向LSTM，没有做分词；同时也实现了使用CNN的版本。由于使用的数据集和github中要求的不同，文本长度大大增加（平均长度大于1000），因此LSTM的准确率比CNN有优势，在50个epoch左右后训练集上准确率达到了`85%`至`90%`，但在50~100epoch时均未收敛

my model

- 使用`Pytorch`框架，用`glove`模型初始化embedding矩阵，分词使用了`pkuseg`包，和baseline一样使用Bi-LSTM。
- 使用了预训练的词向量后与未使用相比在训练集上几乎无提升，在测试集上有小幅提升(其实应该只是误差，因为后来我下载了glove的词向量文件，发现它只有英文单词->向量，而本task中使用的是中文数据集)

- 最开始出现了梯度消失，loss始终不下降，很疑惑，于是尝试调整学习率，从0.001~0.02之间反复调整，没有发现明显的好转
- 思考之后认为应当是模型参数过多，于是对文本长度进行降维，最开始使用了简单粗暴的截断文本方式，发现把截断长度从1500调整到500之后有好转。
- 然而上述方法应当不是正确的解决方法，于是查阅资料，在LSTM层前面又附加了一个卷积层和最大池化层，达到降维而不损失过多有效信息的目的
- ~~然而baseline里和我一样的处理思路却没有遇到梯度消失的问题，这种事绝对很奇怪啊.jpg~~

task3：基于注意力机制的文本匹配

任务

输入两个句子判断，判断它们之间的关系。参考ESIM（可以只用LSTM，忽略Tree-LSTM），用双向的注意力机制实现。

> 最开始看到这个任务的时候先上百度搜了一下ESIM，发现全是e-sim卡介绍，后来一看原来是要复现论文，很受惊吓，再一看发现ESIM现成的轮子其实不少，遂开始写这个task

预备知识

- ESIM：文中提出的自然语言推断模型包含以下几个主要成分：输入编码（**input encoding**），局部推理模型（**local inference modeling**）和推理合成（**inference composition**）。图 1展示了该框架的整体架构。纵向来看，图中包含3个主要成分，横向来看，图的左半部分代表序列NLI模型—ESIM，右半部分图代表基于语义树的树神经网络模型。其中，**local inference modeling**部分使用了注意力机制。

Figure 1: A high-level view of our hybrid neural inference networks.

- ESIM的详细内容如下所示（图源自CSDN博客）：
- **input encoding层**：

对任意一个句子a, b，经过以下层

1. **词嵌入层**: 句子$x \xrightarrow{\text{词嵌入}}$ 矩阵$X = [x_1, ...x_T]^T \in \mathbb{R}^{T \times l_f}$

   其中l_f 为词特征长度，T 为句子长度。

2. **Dropout层**
3. **LSTM 层**: $X = [x_1, ...x_T]^T \xrightarrow{\text{LSTM}} A \in \mathbb{R}^{T \times (2 * l_h)}$

   其中l_h 为隐藏特征的长度。

- **Local Inference Modeling层**：

对同一项的两个句子A, B，经过以下层

1. **矩阵乘积**: $E = AB^T$
2. **矩阵乘积**: $\tilde{A} = \text{softmax\_column}(E)B,\ \tilde{B} = \text{softmax\_row}(E)^T A$
3. **拼接**: $m_a = [A; \tilde{A}; A - \tilde{A}; A \odot \tilde{A}],\ m_b = [B; \tilde{B}; B - \tilde{B}; B \odot \tilde{B}] \in \mathbb{R}^{T \times (8 * l_h)}$

- **`Inference Composition层`**：

对任意一个句子m_a, m_b，经过以下层

1. **全连接层**: $Y = mP_1 + q_1\vec{1}^T,\ P_1 \in \mathbb{R}^{(8*l_h) \times l_h},\ q_1 \in \mathbb{R}^{l_h},\ Y \in \mathbb{R}^{T \times l_h}$
2. **Dropout层**
3. **LSTM 层**: $Y = [y_1, ...y_T]^T \xrightarrow{\text{LSTM}} V \in \mathbb{R}^{T \times (2*l_h)}$

- **`Output层`**：

对任意一个句子V_a, V_b，经过以下层

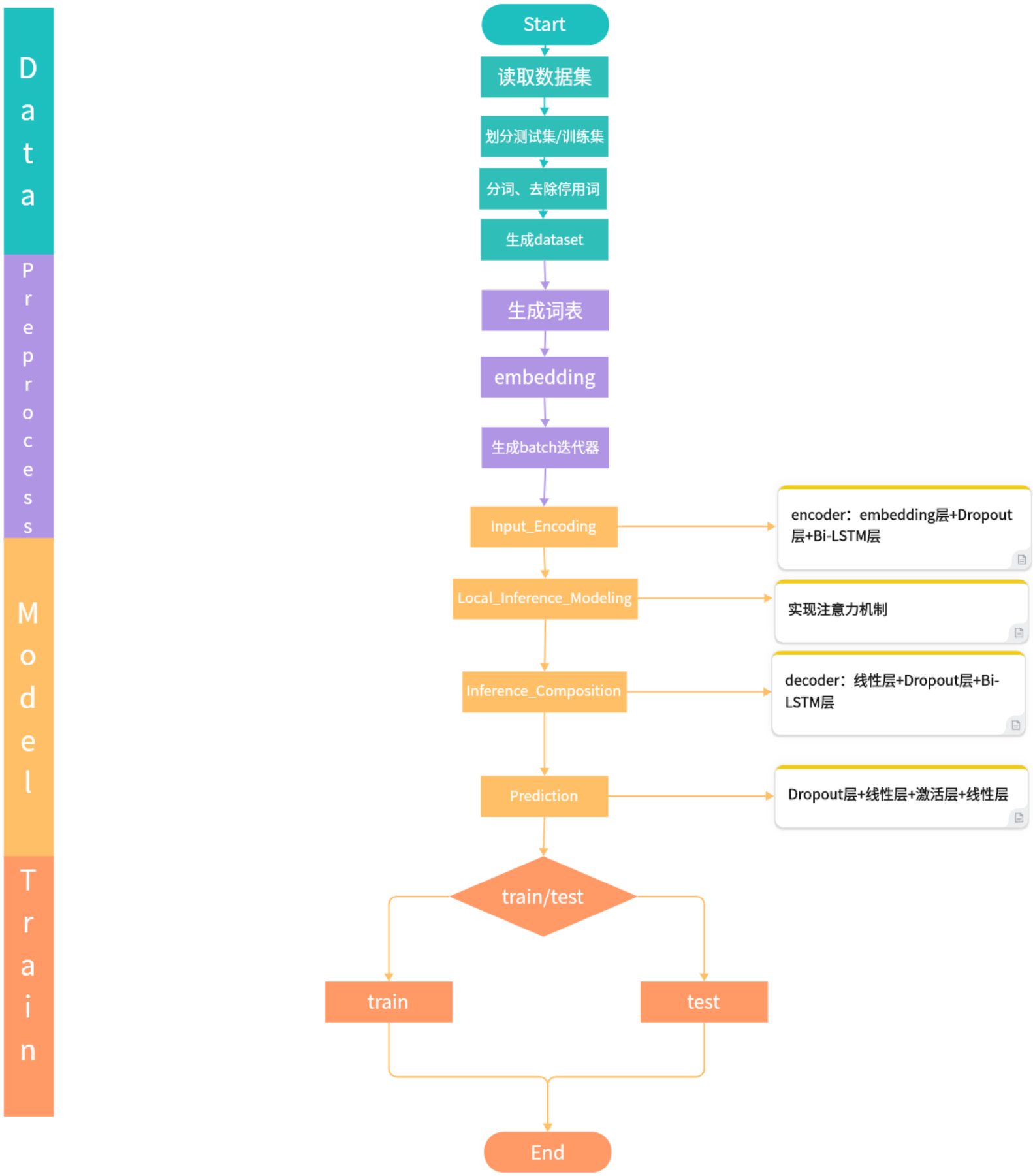
1. **最大汇聚层**: $V \xrightarrow{\text{Row Max Pooling}} v_{\_,max} \in \mathbb{R}^{1 \times (2*l_h)}$
2. **平均汇聚层**: $V \xrightarrow{\text{Row Average Pooling}} v_{\_,avg} \in \mathbb{R}^{1 \times (2*l_h)}$
3. **拼接**: $v = [v_{a,avg}; v_{a,max}; v_{b,avg}; v_{b,max}] \in \mathbb{R}^{1 \times (8*l_h)}$
4. **Dropout层**
5. **全连接层**: $Z = vP_2 + q_2,\ P_2 \in \mathbb{R}^{(8*l_h) \times l_h},\ q_2 \in \mathbb{R}^{l_h},\ Z \in \mathbb{R}^{l_h}$
6. **激活层**: $Tanh(\cdot)$
7. **全连接层**: $logits = ZP_3 + q_3,\ P_3 \in \mathbb{R}^{l_h \times 5},\ q_2 \in \mathbb{R}^5,\ logits \in \mathbb{R}^5$

数据集

- Standford-NLP数据集
- 斯坦福自然语言推理（SNLI）语料库（1.0 版）是 57 万个人工编写的英语句子对的集合，这些句子对被手动标记为平衡分类，标签为 *蕴含* 、 *矛盾* 和 *中性* 。目标是作为评估文本表示系统的基准，特别是包括那些由表示学习方法诱导的系统，以及开发任何类型的 NLP 模型的资源。

任务思路

- 使用预训练的 `glove` 模型进行词嵌入
- 词表生成阶段与之前不同的是，使用了一个trick：去除了glove模型中出现而语料库中没有出现的词向量，这样使得词表与训练集词表大小相同
- 没有使用 `torchtext` （因为感觉其可拓展性感觉不是很好），故dataset、dataloader等组件和padding等操作需要自己实现
- 损失函数仍然使用交叉熵
- 激活函数使用 `tanh` 函数
- optimizer使用了 `Adam`
- 流程图如下所示：



实验设置

- `dataset_size`：57w
- `train_size`：`test_size`=7:3
- `learning_rate`:0.001
- `batch_size`:1000

实验结果

baseline

- baseline整体的思路与上述思路相仿，不过是严格按照论文复现的（甚至在注释里出现了某行代码和文中的某个公式相对应的说明）
- 在训练集上的准确率约为`85%`，在测试集上的准确率约为`83%`，在第16个epoch收敛，与论文中的结果相仿
- ~~跑了baseline之后才发现tqdm原来有个`set_description()`函数，发现每次都在for循环里print来看结果的我像个小丑~~

my model

- 与baseline的模型和预处理阶段思路基本相同
- 在训练集上的准确率约为`82%`， 在测试集上的准确率约为`80%`
- 准确率应该可以继续提升（如果用`lr_scheduler()`中的函数动态调整学习率而不是一直使用0.001的话，改变其他参数应该也能够带来一定提升）
- 训练的效率也需要进一步提升，57w的数据集用本机的RTX3080也要跑比较长的时间才能完成一轮epoch

task4：基于LSTM+CRF的序列标注

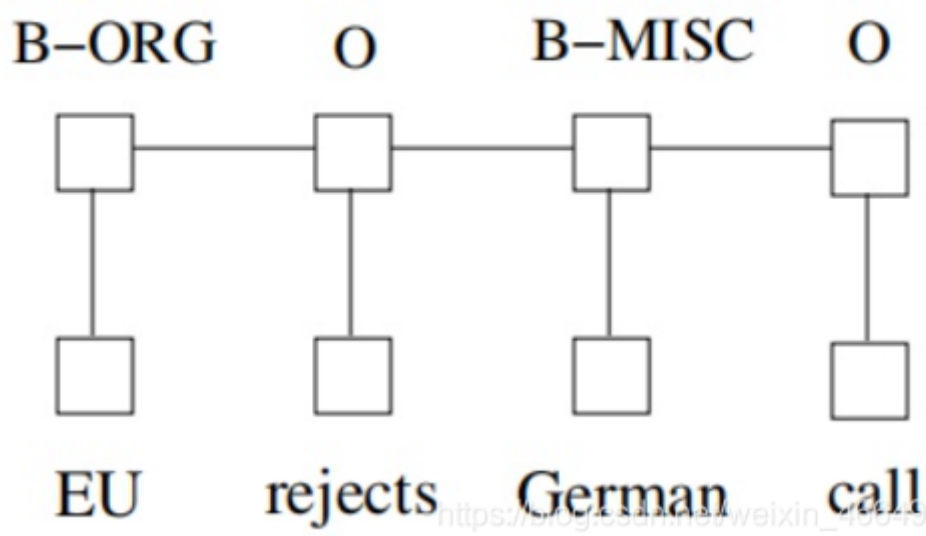
任务

用LSTM+CRF来训练序列标注模型：以Named Entity Recognition为例。

> - 最开始看到任务的时候试图自己手搓一个LSTM+CRF，搓了半天去看教程，发现pytorch官方文档里就有`BiLSTM_CRF`的实现，遂直接copy之
> - copy了之后没跑通，后来发现`pytorch-crf`库里提供了crf的实现，于是直接掉包了

预备知识

- **CRF**：CRF模型是在HMM模型的基础上发展起来的。根据HMM模型的齐次马尔科夫性假设：假设隐藏的马尔科夫链在任意时刻t的状态只依赖于其前一时刻的状态，与其他时刻的状态及观测无关，也与时刻t无关。而CRF模型不仅考虑前一时刻的状态，还考虑其前面与后面的多个状态。一般来说，CRF会具有更好的标记性能。



- **Bi-LSTM + CRF**：BiLSTM也可以做序列标注的问题，只需要在每一步的隐藏层的输出接上一个[n_hidden, n_label]的映射矩阵，然后用softmax做一个多分类。BiLSTM可以捕捉长距离的上下文信息，对于每一步能获得很好的语意向量，CRF是对整个序列进行似然概率统计，并且捕捉转换信息，将两者结合起来可以互补不足。论文《End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF》中的解释：

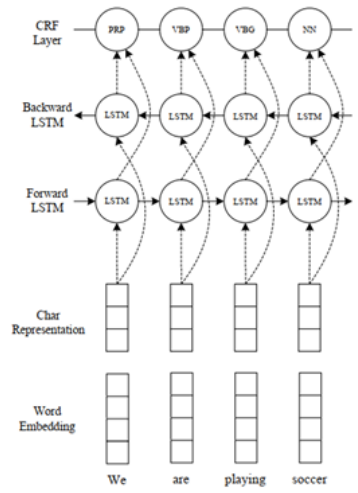


Figure 3: The main architecture of our neural network. The character representation for each word is computed by the CNN in Figure 1. Then the character representation vector is concatenated with the word embedding before feeding into the BLSTM network. Dashed arrows indicate dropout layers applied on both the input and output vectors of BLSTM.

Bi-LSTM + CRF由于引入了双向长短依赖的结构，使其能更好的抽取序列特征，而CRF结构本身可以在句子级别上计算损失，能约束label之间的关系，Bi-LSTM和CRF两者的结合作为序列标注任务的模型，比单独的CRF或LSTM效果都要好。

数据集

- 使用了CONLL 2003 Language-Independent Named Entity Recognition
- CoNLL-2003共享任务数据文件包含四列，由一个空格分隔。每个词都被放在一个单独的行上，每个句子后面都有一个空行。每行的第一项是一个词，第二项是语音部分（POS）标签，第三项是句法块标签，第四项是命名实体标签。大块标签和命名实体标签的格式为I-TYPE，这意味着该词位于TYPE类型的短语中。只有当两个相同类型的短语紧随其后时，第二个短语的第一个词才会有标签B-TYPE，以表明它是一个新短语的开始。带有标签O的单词不是短语的一部分。
- 示例：

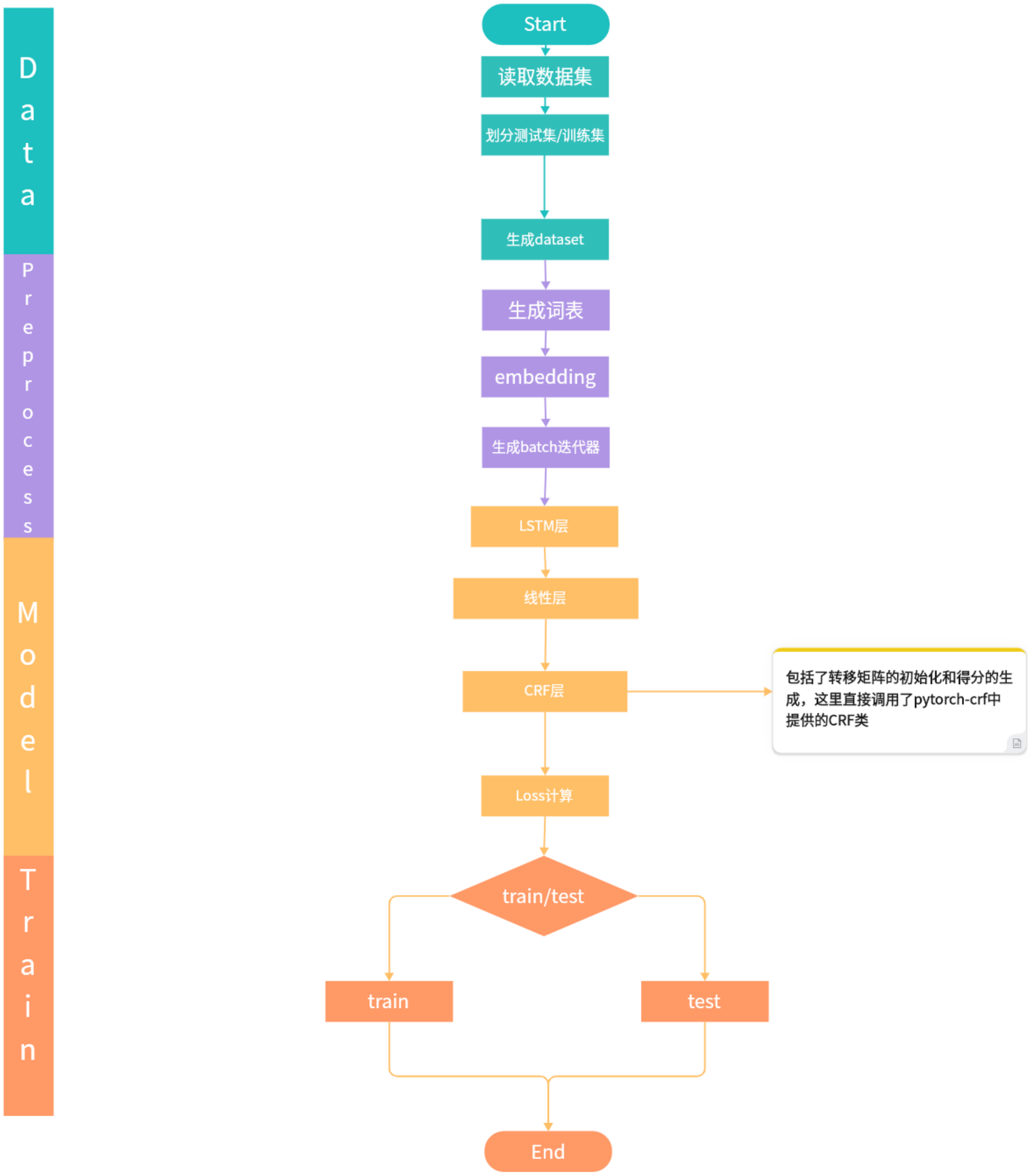
```
U.N.          NNP  I-NP  I-ORG
official      NN   I-NP  O
Ekeus         NNP  I-NP  I-PER
heads         VBZ  I-VP  O
for           IN   I-PP  O
Baghdad       NNP  I-NP  I-LOC
.             .    O     O
```

- 本task需要使用的是第一列和第四列，即实体名称和命名实体标签

任务思路

- dataset的生成，preprocess步骤等大致与之前task相同，不再赘述
- 关于dataset需要注意的一点是原数据集中的文件需要对格式进行一下处理
- `crf`的自行实现是比较困难的，其难点在于
    - 如何从`LSTM`层输出的得分矩阵以及给定的标签生成总可能得分，从而计算loss
    - 如何生成预测序列
- 计算所有序列的得分时，如果使用暴力方法进行遍历，那么时间复杂度是$<$ 标签数 $>^{<\text{序列长度}>}$,是不可接受的，因此网上的方法大多使用`DP`，使用的最多的应当是`viterbi`算法~~(然后本人复现的时候写挂了)~~
- 最后决定直接使用`pytorch-crf`提供的`CRF`类，跳过了上述问题~~(套轮子万岁)~~
- LSTM层基本结构与`task2`中相仿

- 流程图如下：



实验设置

- `batch_size` = 250
- `epochs` = 20
- `learning_rate` = 0.001

实验结果

baseline

- baseline没有用CONLL 2003 的数据集，而是自己使用了一个中文数据集，同时分类数也要大于5
- 因此baseline的参考价值不大
- ~~更遗憾的是baseline跑不通，我也懒得帮它debug了~~

my model

- 最开始调用pytorch-crf的时候，使用了

```
1 from torchcrf import crf
```

然后显示没有 `torchcrf` 这个模块，上网查了一下然后教程说要import `torchctf==0.4.0` 才行，按照教程做了之后又发现它的CRF类缺了一个叫 `batch_first` 的参数，最后手动下载包解压到本地才解决

~~（P.S.首先包名和顶层模块名不同名就已经挺恶心了，然后在包里把import关系写挂了我是没想到的，感觉我自己写pypi包的时候都比这强）~~

- 具体的实验结果和论文中比较相近，或许是因为LSTM和CRF都是直接套轮子的原因
- 20个epoch之后准确率达到了 `85%` ，感觉再多跑一些个epoch还能够有一些提高

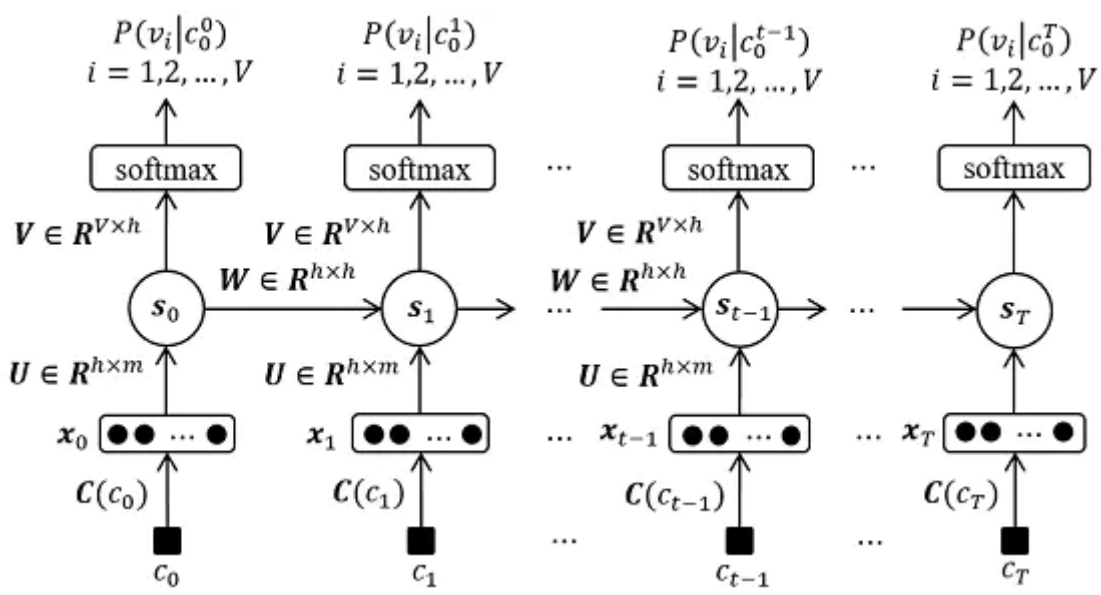
task5： 基于神经网络的语言模型

任务

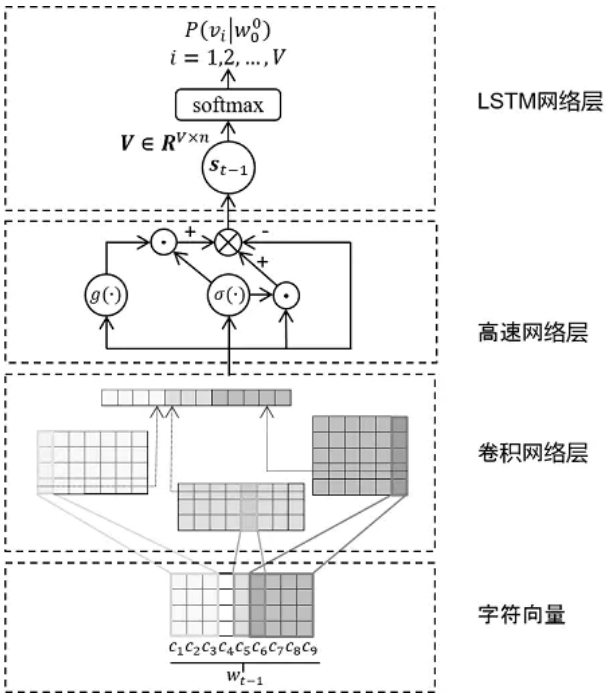
用 `LSTM` 、 `GRU` 来训练字符级的语言模型，计算困惑度

预备知识

- **字符级模型**：语言建模一直以分词为最小单位，即词级语言模型。后来研究者尝试在字符级别进行语言建模，提出了数种字符级的语言模型，其中最为成功是Y. Kim and et. al. (2015)提出的模型。字符级语言模型的优势在于能够解决未登录词的问题，并且能够将词语的形态信息引入语言模型，从而提升模型的性能。
  - **纯粹字符级模型**：建立字符级语言模型的直接想法就是将文本作为字符序列来处理，即将词级别的语言模型中的词替换为字符，而模型的输入序列由词序列换为字符序列，这也是字符级模型研究的最初构思，其模型结构如下图。不过，字符级语言模型的效果并不理想，字符级语言模型生成的文本虽然看起来似乎遵循一定的语法，但并不是很有说服力。



  - **字符级输入模型**：针对纯粹的字符级语言模型的研究结果并不理想，研究者就改变了思路，放弃了字符级语言模型的部分优点，选择了折中的方案。Y. Kim and et. al. (2015)提出了Character-aware的字符级语言模型，即只在模型的输入端采用字符级输入，而输出输出端仍然沿用词级模型，整个模型的结构如下图。 在Y. Kim and et. al. (2015)提出模型中，输入端将每个词当作字符序列进行处理，而后采用卷积网络(Convolutional Neural Network，CNN) 在字符序列上进行一维卷积，从而得到分词的词向量。



数据集

数据集为nlp-beginner仓库中提供的163首唐诗，其中有部分脏数据

任务思路

- 任务要求是训练字符级的语言模型，这里使用字符级输入模型。
- 理论上来讲，使用字符级的输入模型应当能够简化模型（降维），从而提高效率。然而实际上，由于数据集过小，所以本人认为选用词模型其实并不会有太大影响。
- LSTM层思路仍与之前的task相同
- 模型的主要不同之处在于模型入口处接受字符向量输入，并且通过一层卷积层转化成LSTM模型接受的输入
- LSTM层也可以由GRU层代替

实验结果

Baseline

- 尽管baseline中使用了稍大一些的数据集，但是效果仍然不理想（梯度爆炸了，始终不收敛）
- baseline中出现了一首名诗：`春不...不,江...不,...`，即生成藏头诗时除了指定的字之外其余全都用常用字代替

My model

- 最开始用的是给定的数据集，尝试了`LSTM`和`GRU`，过拟合很严重
- 于是使用了baseline使用的数据集，再次尝试
- 还是过拟合，具体表现在只要生成藏头诗的时候指定的字在训练集中有，模型就倾向于直接把这句诗拿出来用
- 模型跑着跑着Pycharm崩了，一看硬盘空间还剩两个G，于是不进行进一步优化了~~（拿起手机京东下单硬盘ing）~~
- 可以通过数据增强等手段来提高准确率~~（但是咕咕子）~~

```
春風落月色，落月落花枝。
江上千年月，風光落月開。
花來千里月，月落月光開。
月色無人在，風光入海中。
夜來千里月，風落月光開。
凉氣開金樹，香光入玉衣。
如君不可得，不得不知君。
水上無人事，無人不可知。


春日月中山，秋風落月中。
江山不可見，一月不知君。
花色無人處，風風落月開。
月中春月落，秋月落花聲。
夜落春風落，春風落月開。
凉風千里色，風落月中風。
如此無人事，何人不可知。
```

```
18  水中無處處，不見一花聲。
```

总结

- 开始一周的时候在看nlp-beginner推荐的教材神经网络与深度学习，看了一周感觉看懂了
- 过了一周开始写task，只能说这教材对于写task的帮助不能说为0吧只能说是个高阶无穷小量
- 然后一脸懵地从Pytorch框架开始学起，最开始不太着急，结果3天写了2个task，感觉要写不完了，最后两天开始疯狂套轮子女娲补天了
- 最开始还在注意代码规范，各种参数都放在单独的类里，后来所有文件在根目录一字排开
- 教训是任务驱动式学习yyds，套轮子yyds