自然语言处理研讨课
（实践课）
第6章 序列标注实践

赵 阳
中国科学院自动化研究所
yang.zhao@nlpr.ia.ac.cn

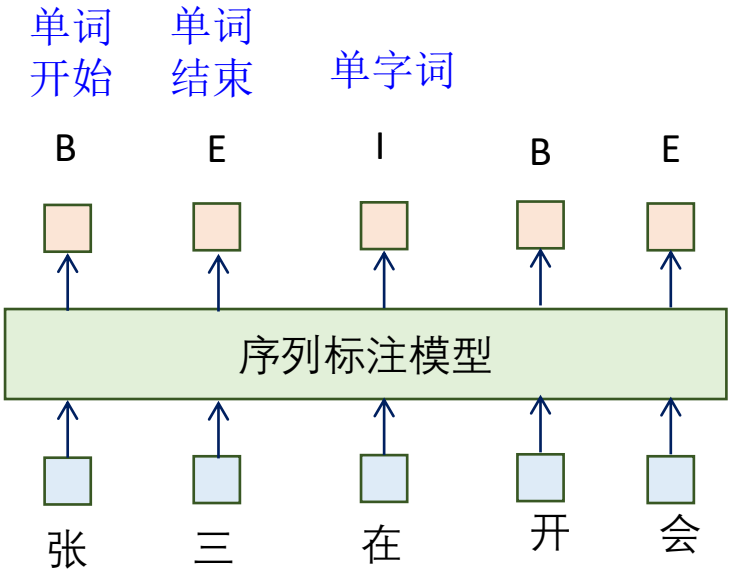
本章内容

1. 序列标注介绍
2. 实践基础介绍
3. BiLSTM+CRF实践介绍
4. 本章实践

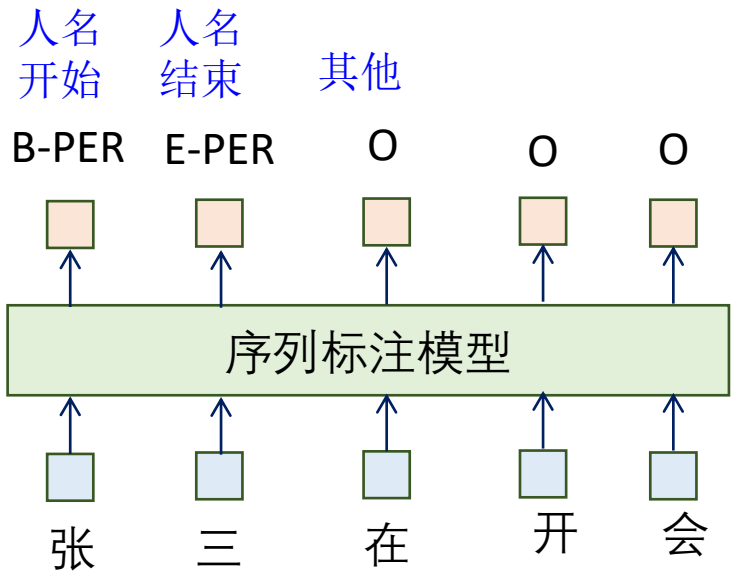
1. 序列标注介绍

序列标注介绍

- 序列标注是自然语言处理的基础任务之一
- 给定输入序列（句子），序列标注任务是将序列中每一个字/词进行分类。
- 常见的序列标注任务：汉语分词、词性标注、命名实体识别、槽位抽取等。



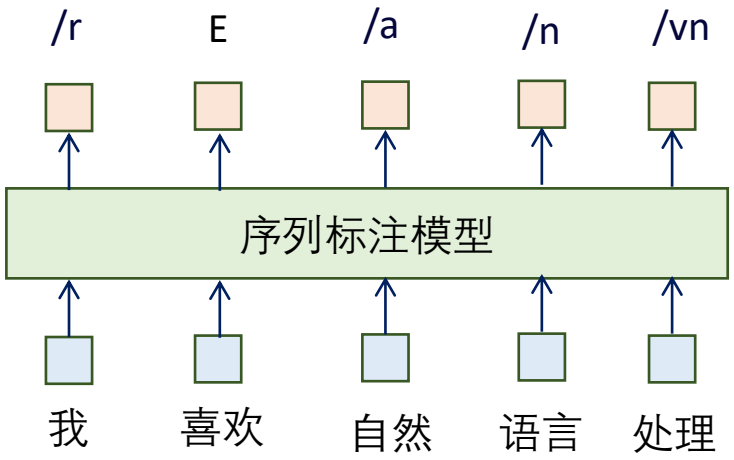
汉语分词



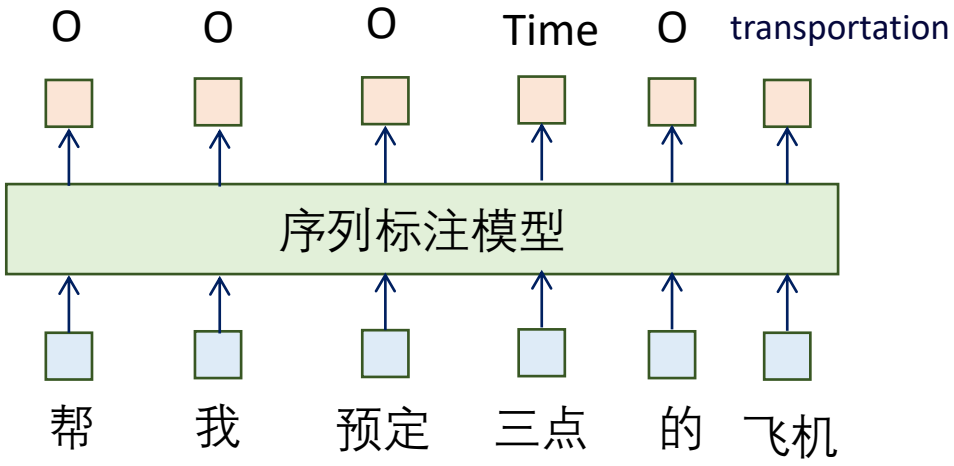
命名实体识别

1. 序列标注介绍

- 序列标注介绍



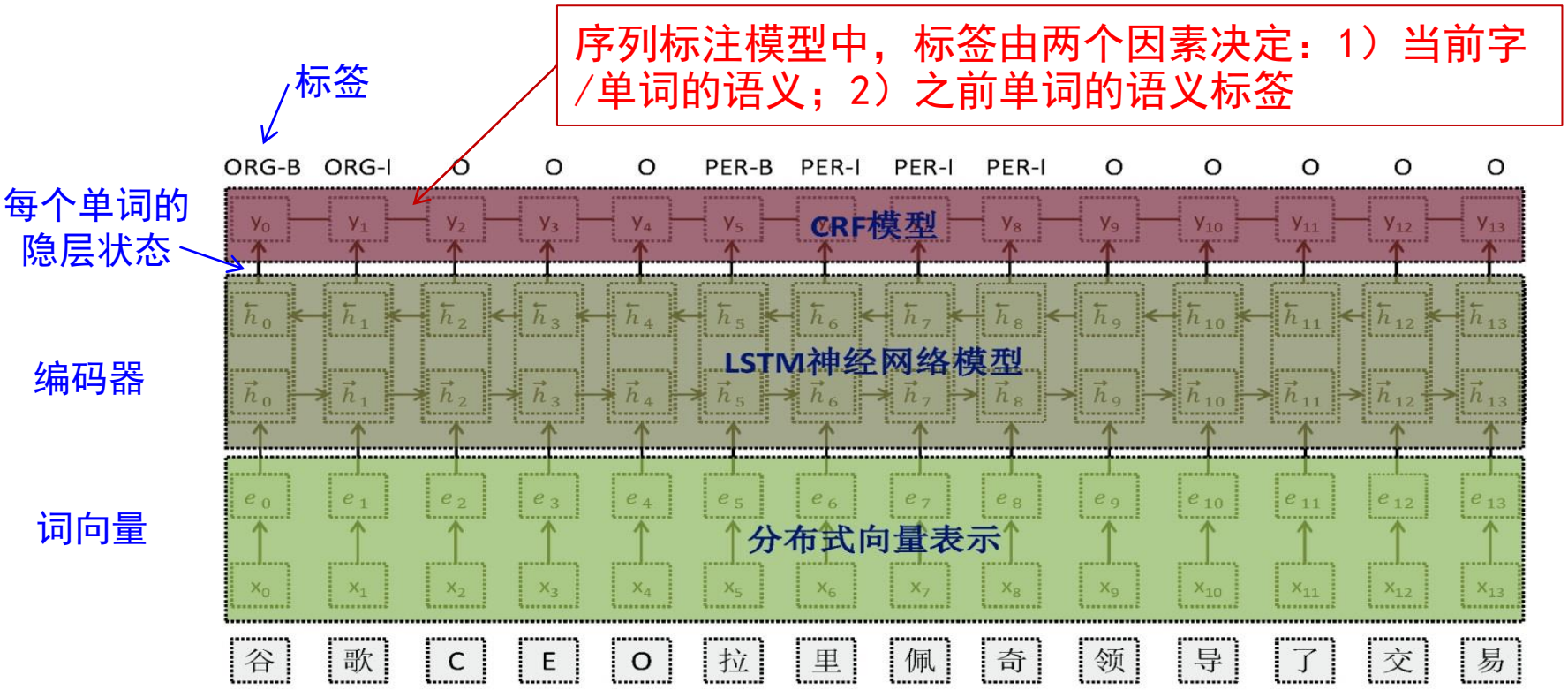
词性标注



槽位识别

1.文本分类介绍

BiLSTM+CRF模型结构



本章内容

1. 序列标注介绍
2. 实践基础介绍
3. BiLSTM+CRF实践介绍
4. 本章实践

2. 实践基础

上节课回顾1：PyTorch的Datasets & DataLoaders

- 使用 PyTorch 的 DataLoader 和 Dataset 处理自己的数据，可以将数据集代码与模型训练代码解耦，而且能够高效处理数据，提升代码的可读性和模块化。
- 加载模块： from torch.utils.data import DataLoader, Dataset

https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

2. 实践基础

基本过程一、构建自定义 Dataset 类

继承 Dataset，实现 __init__、__len__ 和 __getitem__ 三个方法。

- __init__ 方法用于初始化和读取数据集。
- __len__ 方法返回数据集中的样本总数。
- __getitem__ 方法根据给定的索引返回数据集中的样本和标签。

2. 实践基础

基本过程二、 DataLoader实例化和迭代

包装一个数据集，使其能够以一种可迭代的方式提供批次（batch）数据。

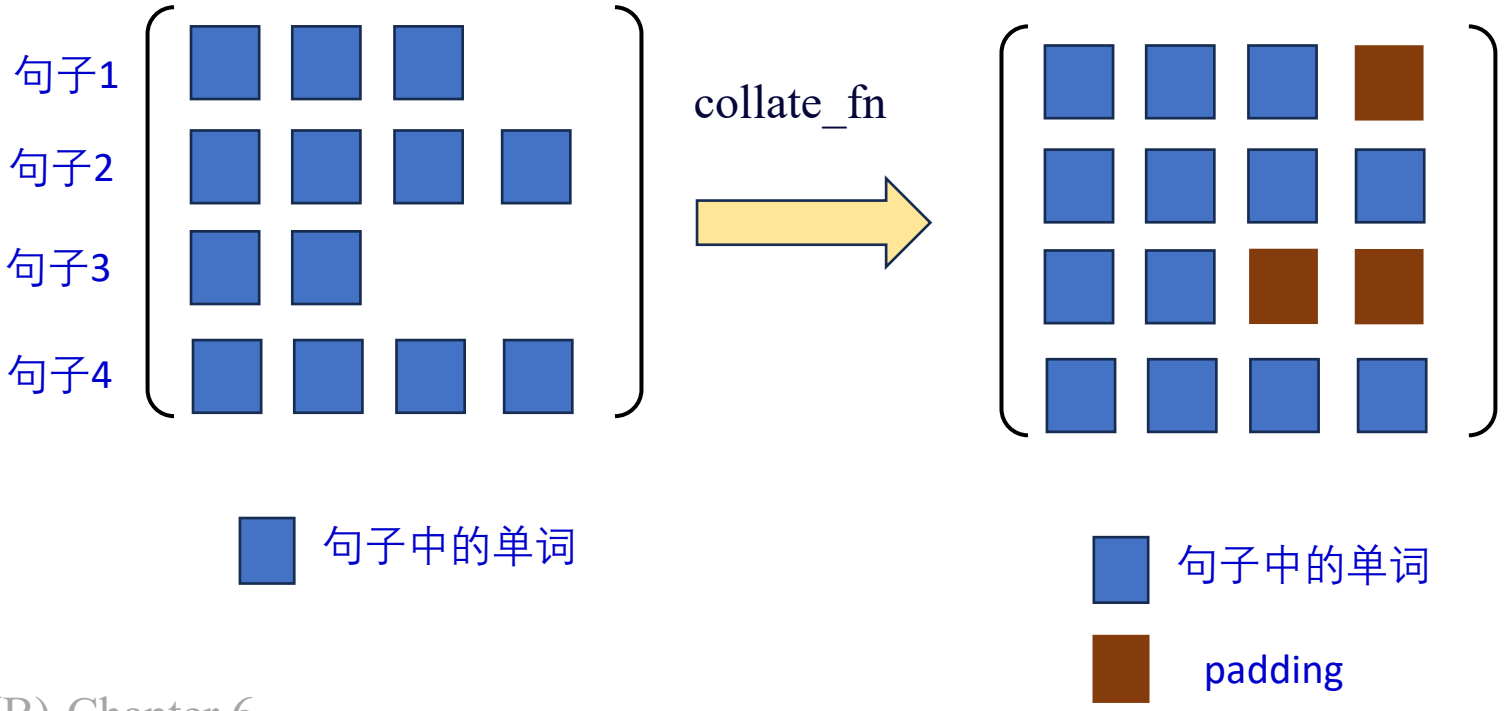
- 基本使用如下：

```
dataset = MyDataset(data_paths)                                    #实例化Dataset
dataloader = DataLoader(dataset, batch_size=8, shuffle=True, num_workers=4)  #实例化DataLoader
for data, labels in dataloader:                                    #迭代DataLoader
    # 在这里进行模型训练等操作
```

2. 实践基础

基本过程二、 DataLoader中的collate_fn

- collate_fn 是 PyTorch 中 DataLoader 的一个重要参数，自定义如何从数据集中获取样本组合成一个批次（batch）
- 如果不指定 collate_fn，DataLoader 会使用默认的方式，它通常将样本直接堆叠，形成一个批次的张量列表。



2. 实践基础

基本过程二、 DataLoader中的collate_fn

Padding的id

初始化text_ids和label_ides

赋值

```
def collate_fn(data):
    pad_idx = 8019
    texts = [d[0] for d in data]
    label = [d[1] for d in data]
    batch_size = len(texts)
    max_length = max([len(t) for t in texts])
    text_ids = torch.ones((batch_size, max_length)).long().fill_(pad_idx)
    label_ids = torch.tensor(label).long()
    for idx, text in enumerate(texts):
        text_ids[idx, :len(text)] = torch.tensor(text)
    return text_ids, label_ids
```

输入 文本　　输出 标签

2. 实践基础

实践基础1：构建mask_ids

- 序列标注中，处理变长问题，需要额外构建mask_ids，标注序列中某个单词是否是正常单词还是填充(padding)
- 通过mask_ids，消除和降低填充的影响，尤其是在RNN和Transformer网络中

读数据和标签

张量形状

初始化

赋值

```
def collate_fn(data):
    words = [item[0] for item in data]
    tags = [item[1] for item in data]
    max_seq_len = max([t.shape[0] for t in words])
    batch_size = len(words)
    word_ids = torch.zeros(batch_size, max_seq_len).long()
    mask = torch.zeros(batch_size, max_seq_len).long()
    tag_ids = torch.zeros(batch_size, max_seq_len).long()
    for idx, (word, tag) in enumerate(zip(words, tags)):
        word_ids[idx, :word.shape[0]] = word
        tag_ids[idx, :tag.shape[0]] = tag
        mask[idx, :word.shape[0]] = 1
    return word_ids, mask.bool(), tag_ids
```

输入 文本　　掩码 标记　　输出 标签

2. 实践基础

实践基础1：构建mask_ids

例子

```
Word IDs:
 tensor([ [1, 2, 3],
          [4, 5, 6],
          [7, 8, 0]])  # 0用于填充
Mask:
tensor([ [ True,  True,  True],
          [ True,  True,  True],
          [ True,  True, False]])  # False表示填充
Tag IDs:
tensor([ [1, 2, 3],
          [1, 2, 3],
          [1, 2, 0]]) # 0用于填充
```

2. 实践基础

实践基础2：条件随机场

- 给定观察序列X，输出标识序列Y，通过计算$P(Y|X)$求解最优标注序列。

$$p(Y|X)=\frac{1}{Z(X)}\exp\left(\sum_j \lambda_j t_j(y_{i-1},y_i,X,i)+\sum_k \mu_k s_k(y_i,X,i)\right)$$

标签的转移概率　　标签的发射概率

- 归一化

$$Z(X)=\sum_Y \exp\left(\sum_{i,j} \lambda_j t_j(y_{i-1},y_i,X,i)+\sum_{i,k} \mu_k s_k(y_i,X,i)\right)$$

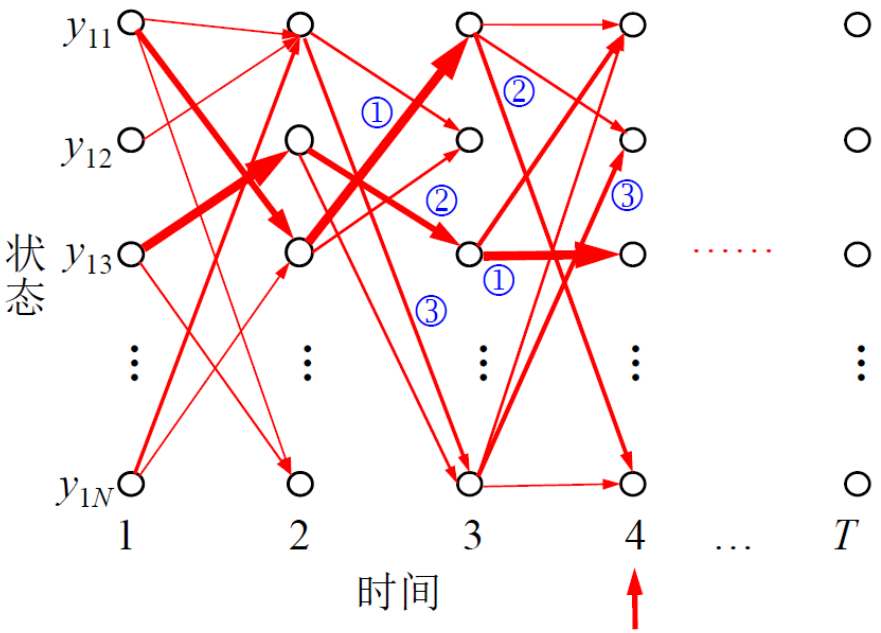
所有序列的分数

2. 实践基础

实践基础2：条件随机场

- 解码：维特比搜索

图解Viterbi搜索过程



剪枝策略：

❶ $\delta_t(j) \geq \Delta$

❷NPath $\leq \sigma$ (3)

- 代码见课程网站 CRF.py

本章内容

1. 序列标注介绍
2. 实践基础介绍
3. BiLSTM+CRF实践介绍
4. 本章实践

2. BiLSTM+CRF实践介绍

- 数据情况

```
1   O
.   O
规  O
定  O
与  O
香  B-LOC
港  I-LOC
特  I-LOC
别  I-LOC
行  I-LOC
政  I-LOC
区  I-LOC
有  O
关  O
的  O
外  O
交  O
事  O
务  O
的  O
法  O
律  O
，  O
如  O
与  O
在  O
香  B-LOC
港  I-LOC
特  I-LOC
别  I-LOC
行  I-LOC
政  I-LOC
区  I-LOC
实  O
```

{"当": 0, "希": 1, "望": 2, "工": 3, "程": 4, "救": 5, "助": 6, "的": 7, "百": 8, "
14, "来": 15, ", ": 16, "科": 17, "教": 18, "兴": 19, "国": 20, "蔚": 21, "然": 22,
: 28, "藏": 29, "价": 30, "值": 31, "书": 32, "你": 33, "没": 34, "买": 35, "明": 3
"初": 42, "！": 43, "本": 44, "是": 45, "所": 46, "传": 47, "统": 48, "门": 49, "类
, "只": 56, "我": 57, "们": 58, "结": 59, "束": 60, "温": 61, "饱": 62, "间": 63, "
69, "关": 70, "寇": 71, "在": 72, "京": 73, "掠": 74, "夺": 75, "文": 76, "物": 77,

chr_vocab.json

{"O": 0, "B-LOC": 1, "I-LOC": 2, "B-ORG": 3, "I-ORG": 4, "B-PER": 5, "I-PER": 6}

tag_vocab.json

数据集样式（train.txt val.txt text.txt）

2. BiLSTM+CRF实践介绍

- BiLSTM+CRF实现思路

导入的模块

```
import torch
import torch.autograd as autograd
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import torch.optim as optim
import json
from CRF import CRF
import argparse
from collections import namedtuple
```

2. BiLSTM+CRF实践介绍

■ BiLSTM+CRF实现思路

- BiLSTM_CRF类（1）

```
class BiLSTM_CRF(nn.Module):

    def __init__(self, config):
        super(BiLSTM_CRF, self).__init__()
        self.target_size = config.tag_size
        self.hidden_dim = config.hidden_dim
        self.dropout = config.dropout
        self.embedding_dim = config.embedding_dim
        self.embedding = nn.Embedding(config.vocab_size, config.embedding_dim)
        self.drop = nn.Dropout(self.dropout)
        self.lstm = nn.LSTM(self.embedding_dim, self.hidden_dim, num_layers=2,
batch_first=True, bidirectional=True)
        self.hidden2tag = nn.Linear(self.hidden_dim * 2, self.target_size + 2)
        self.crf = CRF(self.target_size, config.gpu)
```

为对象各属性赋值

Embedding层

BiLSTM层

线性层

CRF层

正向和反向层的隐层状态

增加开始和结束两个标签

2. BiLSTM+CRF实践介绍

■ BiLSTM+CRF实现思路

- BiLSTM_CRF类（2）

发射概率

正向计算损失
（标准答案的分数）

解码得到分数最高
的序列
（维特比解码）

```
def get_tags(self, word_input):
    embedded = self.embedding(word_input)
    hidden = None
    lstm_feats, hidden = self.lstm(embedded, hidden)
    feats = self.drop(lstm_feats)
    feats = self.hidden2tag(feats)
    return feats

def forward(self, word_input, mask, tags):
    feats = self.get_tags(word_input)
    loss = self.crf(feats, mask, tags)
    return loss

def decode(self, word_input, mask):
    feats = self.get_tags(word_input)
    tag_seq = self.crf.decode(feats, mask)
    return tag_seq
```

2. BiLSTM+CRF实践介绍

BiLSTM+CRF实现思路

- NER_dataset（1）

__init__()
读取字典
读取数据

```
class NER_dataset(Dataset):
    def __init__(self, data_path, chr_vocab_path, tag_vocab_path):
        self.data = self.load_data(data_path)
        self.chr_vocab = json.load(open(chr_vocab_path, 'r', encoding='utf-8'))
        self.tag_vocab = json.load(open(tag_vocab_path, 'r', encoding='utf-8'))
        print(len(self.data))

    def load_data(self, data_path):
        data = []
        with open(data_path, 'r', encoding='utf-8') as f:
            sentence = []
            tags = []
            for line in f:
                if line == '\n':
                    data.append([sentence, tags])
                    sentence = []
                    tags = []
                else:
                    word, tag = line.strip().split('\t')
                    sentence.append(word)
                    tags.append(tag)
        if sentence:
            data.append([sentence, tags])
        return data
```

2. BiLSTM+CRF实践介绍

BiLSTM+CRF实现思路

- NER_dataset（2）

__len__()
返回样本个数

__getitem__()
根据index，返回text和tag

```
def __len__(self):
    return len(self.data)

def __getitem__(self, index):
    sentence, tags = self.data[index]
    char_ids = [self.chr_vocab[char] for char in sentence]
    tag_ids = [self.tag_vocab[tag] for tag in tags]
    return torch.tensor(char_ids), torch.tensor(tag_ids)
```

2. BiLSTM+CRF实践介绍

BiLSTM+CRF实现思路

- 其他函数

参考之前的内容

```
def collate_fn(data):
```

根据标签，抽取tag、开始位置和结束位置

```
def extract_BIO(seq):
    """
    Extracts the BIO tag from a sequence.
    Returns a list of tuples (tag, chunk_start, chunk_end)
    """

return res
```

需要

计算F1值

```
def cal_F1(pred_seq, golden_seq):


return precision, recall, f1
```

2. BiLSTM+CRF实践介绍

BiLSTM+CRF实现思路

- Train()

训练模式

常规操作

```
def train(model, config, train_dataset, eval_dataset, tag_vocab):
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    model.train()
    global_step = 0
    best_f1 = 0
    for epoch in range(config.num_epoch):
        for data in train_dataset:
            loss = model(*data)
            loss.backward()
            optimizer.step()
            model.zero_grad()
            global_step += 1
            if global_step % config.eval_interval == 0:
                print('epoch: {}, global_step: {}, loss: {:.4f}'.format(epoch, global_step, loss.item()))
                f1 = evaluate(model, eval_dataset, tag_vocab)
                if f1 > best_f1:
                    best_f1 = f1
                    torch.save(model.state_dict(), config.save_path)
    return
```

2. BiLSTM+CRF实践介绍

BiLSTM+CRF实现思路

- evaluate（）

评价和测试模式

```
def evaluate(model, dataset, tag_vocab):
    model.eval()
    pred = []
    golden = []
    with torch.no_grad():
        for data in dataset:
            pred.extend(model.decode(*data[:-1]).reshape(1, -1).squeeze().int().tolist())
            golden.extend(data[-1].reshape(1, -1).squeeze().int().tolist())
    pred = [tag_vocab[i] for i in pred]
    golden = [tag_vocab[i] for i in golden]
    precision, recall, f1 = cal_F1(pred, golden)
    print('precision: {:.4f}, recall: {:.4f}, f1: {:.4f}'.format(precision, recall, f1))
    return f1
```

训练中使用，每隔一定训练步数，输出验证集结果

2. BiLSTM+CRF实践介绍

■ BiLSTM+CRF实现思路

- 主函数（1）

创建解析器对象

添加参数

解析参数

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--save_path', default='save_model/best.pt')
    parser.add_argument('--train', default='../train.txt')
    parser.add_argument('--test', default='../test.txt')
    parser.add_argument('--val', default='../val.txt')
    parser.add_argument('--chr_vocab', default='../LSTM+CRF/chr_vocab.json')
    parser.add_argument('--tag_vocab', default='../LSTM+CRF/tag_vocab.json')
    parser.add_argument('--num_epoch', default=25, type=int)
    parser.add_argument('--lr', default=0.0015, type=float)
    parser.add_argument('--dropout', default=0.3, type=float)
    parser.add_argument('--hidden_dim', default=300, type=int)
    parser.add_argument('--batch_size', default=64, type=int)
    parser.add_argument('--eval_interval', default=100, type=int)
    arg = parser.parse_args()
```

2. BiLSTM+CRF实践介绍

■ BiLSTM+CRF实现思路

- 主函数（2）

实例化Dataset

实例化DataLoader

config字典和元组

实例化模型训练

```
train_dataset = NER_dataset(arg.train, arg.chr_vocab, arg.tag_vocab)
val_dataset = NER_dataset(arg.val, arg.chr_vocab, arg.tag_vocab)
test_dataset = NER_dataset(arg.test, arg.chr_vocab, arg.tag_vocab)
train_loader = DataLoader(train_dataset, batch_size=arg.batch_size, collate_fn=collate_fn)
val_loader = DataLoader(val_dataset, batch_size=arg.batch_size, collate_fn=collate_fn)
test_loader = DataLoader(test_dataset, batch_size=1, collate_fn=collate_fn)

chr_vocab = json.load(open(arg.chr_vocab, 'r', encoding = 'utf-8'))
tag_vocab = json.load(open(arg.tag_vocab, 'r', encoding = 'utf-8'))
tag_vocab = {v:k for k,v in tag_vocab.items()}
config = {
    'vocab_size': len(chr_vocab),
    'hidden_dim': arg.hidden_dim,
    'dropout': arg.dropout,
    'embedding_dim': arg.hidden_dim,
    'tag_size': len(tag_vocab),
    'gpu':False
}
config = namedtuple('config', config.keys())(**config)
model = BiLSTM_CRF(config)
train(model, arg, train_loader, val_loader, tag_vocab)
```

本章内容

1. 序列标注介绍
2. 实践基础介绍
3. BiLSTM+CRF实践介绍
4. 本章实践

本节实践

■ 序列标注实践

- 利用提供的数据集，实现BiLSTM+CRF 或者BiGRU+CRF（二选一）。
- 完成未提供的函数，完成测试过程。
- 在提供的数据集上训练实现的模型，并汇报最终的性能指标。

> 注意：
>
> 可以按照PPT上的思路实现，也可以不按照思路实现。

本节实践

■ **要求和评分准则：**

- 报告需要详细说明使用的模型结构，超参数，实验结果等；
- 不需要把程序大段复制上去，如果有需要，只需要截取和复制关键程序即可；报告页数不需要太长。
- 如果满足了基本要求就可以得到B（80-85分）
- 剩余的15分， 看大家的自己拓展分析和实验

本节实践

■ **要求和评分准则：**

- 第二次作业需要大家从（文本分类、序列标注和机器翻译）三项任务中选择**至少一项**进行实践；
- 等三项内容介绍完成后，一起提交（大约在11月下旬）。
- 完成任务越多，对应的分数也越高。

谢谢！

Thanks!