

BiLSTM + CRF文档

模型运行起始位置

整个模型的运行是从 ./run.py 文件的 main 函数开始运行的

BiLSTM-CRF 中文分词模型原理

使用 BiLSTM 模型来得到每个字对应 4 种标签的概率，然后使用 CRF 算法结合上下文来最终确定每个字符所对应的标签，然后根据标签进行中文分词。

BiLSTM 模型和 LSTM 模型原理类似，故只需要理解 LSTM 模型即可理解 BiLSTM 模型。

数据集处理

- 数据来源

./data/output.txt 中给出了已经完成分词的数据集，这些数据集将被用作训练集和验证集

- 数据预处理

```
simple_run()
```

该函数用于进行数据预处理。在读取已经分词的数据集 A 后，会标注其中每个字所对应的标签，并将标签保存在一个新的列表中（即 getlist 函数的返回值）。接下来将会把 A 和 B 这两个数据集按照 9:1 的比例划分为训练集和验证集。训练机用于训练模型，验证集会用于验证模型参数的有效性。

变量 vocab 是一个词表，建立词表是为了统计每一个字出现的次数，并且为每一个字设置一个唯一的 id，这样以后就可以通过 id 来访问字。

模型训练 & 模型验证

run()

这是模型训练最重要的函数，模型训练、验证都是在这个函数中进行。

在该函数中，我们首先创建了两个迭代器 `train_loader` 和 `dev_loader` ,分别为训练集数据迭代器和验证集数据迭代器。

我们会建立一个模型 `model` ，该模型的参数是我们在`config.py`文件中提前设置好的参数。然后我们把模型放入设备中，因为笔者电脑没有GPU，所以我们会将模型放入CPU中。

接下来我们建立了一个优化器。优化器可以保存当前模型参数，并需要根据网络反向传播的梯度信息来更新网络的参数，以起到降低 `loss` 计算值的作用。优化器具体的用法可以参考下面的链接，写得很精炼直白。

<https://blog.csdn.net/lbelievesunshine/article/details/99624645>

在优化器中我们会设置学习率，该值越高模型参数的变化幅度就会越大，因此我们需要设置一个函数来不断降低优化器的学习率，从而使得模型更加容易收敛。 `StepLR(_LRScheduler)` 这个函数就是用来降低优化器的学习率的，该函数的具体用法在代码中有着详尽的注释，直接搬运过来

```
1 class StepLR(_LRScheduler):
2     """Decays the learning rate of each parameter group by gamma every
3     step_size epochs. Notice that such decay can happen simultaneously with
4     other changes to the learning rate from outside this scheduler. When
5     last_epoch=-1, sets initial lr as lr.
6
7     Args:
8         optimizer (Optimizer): Wrapped optimizer.
9         step_size (int): Period of learning rate decay.
10        gamma (float): Multiplicative factor of learning rate decay.
11            Default: 0.1.
12        last_epoch (int): The index of last epoch. Default: -1.
13        verbose (bool): If ``True``, prints a message to stdout for
14            each update. Default: ``False``.
15
16    Example:
17        >>> # Assuming optimizer uses lr = 0.05 for all groups
18        >>> # lr = 0.05    if epoch < 30
19        >>> # lr = 0.005   if 30 <= epoch < 60
20        >>> # lr = 0.0005  if 60 <= epoch < 90
21        >>> # ...
22        >>> scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
23        >>> for epoch in range(100):
24            >>>     train(...)
25            >>>     validate(...)
26            >>>     scheduler.step()
```


下面回到 `train` 函数中。我们已经通过 `epoch_train` 函数完成了本轮的模型参数训练，接下来我们需要判断该参数是否在以前的基础上dedaole优化。验证的具体过程在函数 `dev` 中。验证的思路是基于验证集的文字集合，使用模型训练出来的参数来预测其每个字所对应的标签，然后将其与实际标签进行比较，统计预测的正确率作为模型参数的有效性。具体 `dev` 验证的逻辑详见代码中的注释。

下面回到 `train` 函数中，接下来我们需要决定是否要更新我们保存的最佳模型参数。我们需要根据本次训练得到的模型评价指标来进行判断，如果发现当前参数优于我们保存的参数，则更新参数；否则，继续进行下一次训练。当我们连续5次训练都没有得到更好的参数时，我们可以认为当前的模型已经收敛，当前保存的参数就是最佳参数。