Assignment 4

Part I: Text Classifiers

1. 代码组织

文件名	功能
config.py	在 class 中定义模型参数
dataset.py	读取、处理数据，利用 fastnlp 生成 train_data、dev_data、test_data 和 vocabulary，并用 pickle 导出
w2v.py	生成 word2vec 的 embedding 的预训练 weight，并用 pickle 导出
model.py	用 pytorch 定义 CNN、RNN、LSTM、RCNN 模型
utils.py	定义用于计时 Callback 类
train.py	定义 trainer，用于训练、测试模型
visualize.py	从 log 文件中提取出 loss、accuracy，并画出曲线（由于在训练过程中用 tee 命令把控制台输出保存到 log 文件中）

2. 数据处理

（代码请看 dataset.py）

2.1 使用了 20news-bydate_py3.pkz 数据，用 fetch_20newsgroups 取出全部 20 类的数据，其中有 train 和 test 数据

2.2 把原始 train、test 数据导入到 fastNLP 的 DataSet 中，分别用 3 个 apply 来对数据进行以下操作

- 去掉 string,punctuation
- 把 string.whitespace 变成 space
- 把数据全部小写化，并以 space 切词

2.3 把处理好的 train 数据按 4:1 划分为 train_data 和 dev_dev，而处理好的 test 数据就作为 test_data

2.4 从 train_data 中获得 vocabulary，大小为 55253

2.5 按照 train_data 获得的 vocabulary, 分别用 apply 把 trian_data、dev_data 和 test_data 中的 word 变为 index

2.6 用 pickle 导出最终的 trian_data、dev_data、test_data 和 vocabulary，后缀名为.pkl

2.7 数据集参数

20news-bydate_py3.pkz	最终数据	Size	**Categories**
train_set	train_data	9052 (80%) (vocabulary size: 55253)	**20**
	dev_data	2262 (20%)	
test_set	test_data	7532	

3. word2vec 的 embedding 预训练 weight 生成

（代码请看 w2v.py）

- 用 pickle 导入上文生成的 trian_data、dev_data、test_data 和 vocabulary
- 用 vocabulary 把 trian_data、dev_data、test_data 中的 idx 变回 word
- 把 train_data 中的 input 数据取出，存到一个二维数组中
- 用 gensim 定义 word2vec 模型

- 用上述 train_data 的二维数据构建 word2vec 模型的 vocabulary
- 把上述 train_data 的二维数据放到 word2vec 模型中训练
- 从模型中取出每个 index 对应的 embedding，组成 weight 矩阵。其中 index 为 0 的单词是'<pad>'，在 word2vec 模型中不存在（虽然 fastnlp 的 vocabulary 中有'<pad>'，但是数据中没有'<pad>'），因此把 index 为 0 的 embedding 定为全 0
- pickle 保存 weight 矩阵

word2vec 模型参数	**window**	**min_count**	**size**
值	**1 和 64**	**1**	**embed_dim=128**

4. 模型参数

（代码请看 config.py）

- **软编码**：本次实验所涉及的参数全部定义在这个文件里，包括模型参数、文件路径、文件名等。
- **本次实验一共实现了 CNN, CNN_w2v, RNN, LSTM, LSTM_maxpool, RCNN 共 6 个不一样的模型，**以下是这些模型的参数：

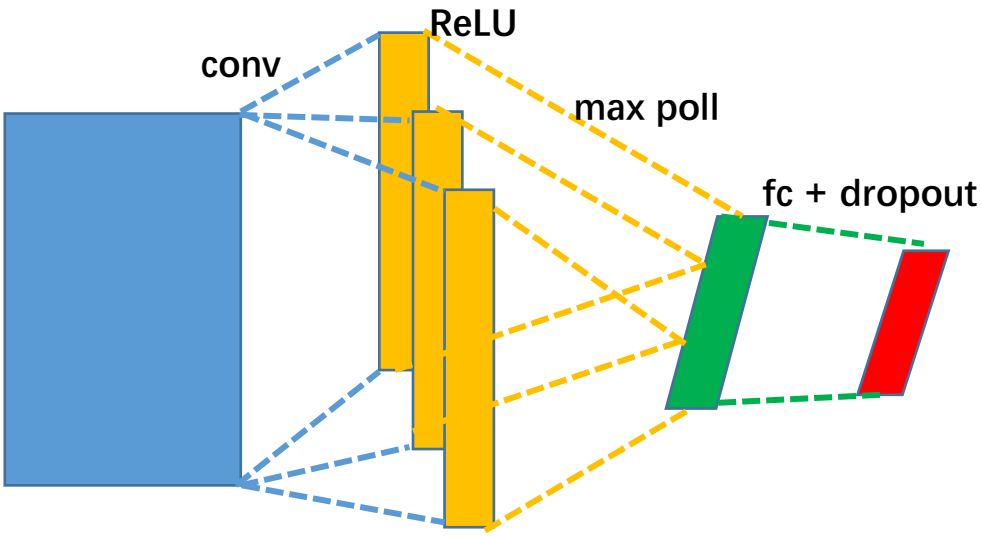
model	**CNN**	**CNN_w2v**	**RNN**	**LSTM**	**LSTM_maxpool**	**RCNN**
embed_dim	128					
kernel_sizes	(3, 4, 5)					
kernel_num	100					
in_channels	1					
dropout	0.5					
word2vec	False	True	False			
num_layers			1	2	2	1 和 2
bidirectional			true			
hidden_dim			256			
optimizer	Adam (lr=1e-3, weight_decay=0)					
patience	100 和 10	20	20	100 和 20	20	20
max_epoch	128					
batch_size	64		8			
print_every	10					
validate_every	100					

5. 模型结构（用 pytorch 实现）

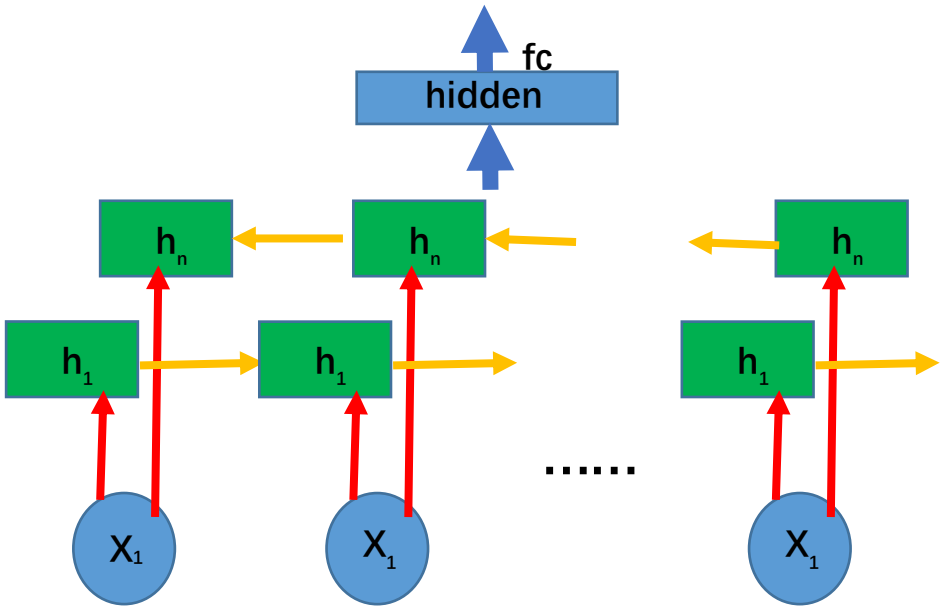
（代码请看 model.py）

一共实现了 CNN, CNN_w2v, RNN, LSTM, LSTM_maxpool, RCNN 共 6 个不一样的模型。

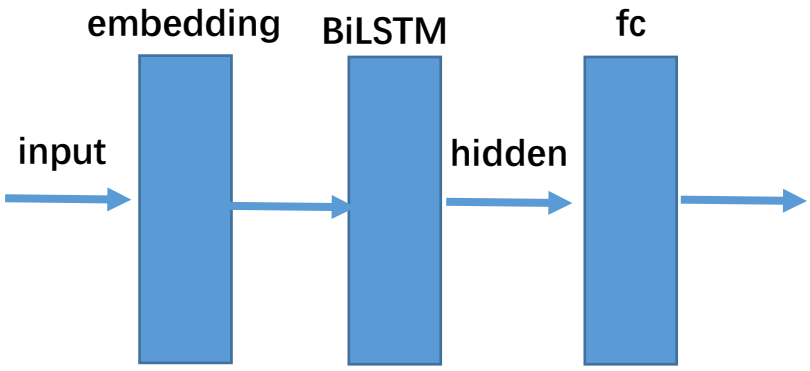
- **模型一 CNN：**input 先过一个 embedding 层，再过一个卷积层。卷积核大小为(3, 4, 5)，有 100 个。过了卷积层后用 ReLU 激活，然后 max pool。再把 3 个卷积核的结果连起来，然后 dropout，最后过全连接层，然后输出



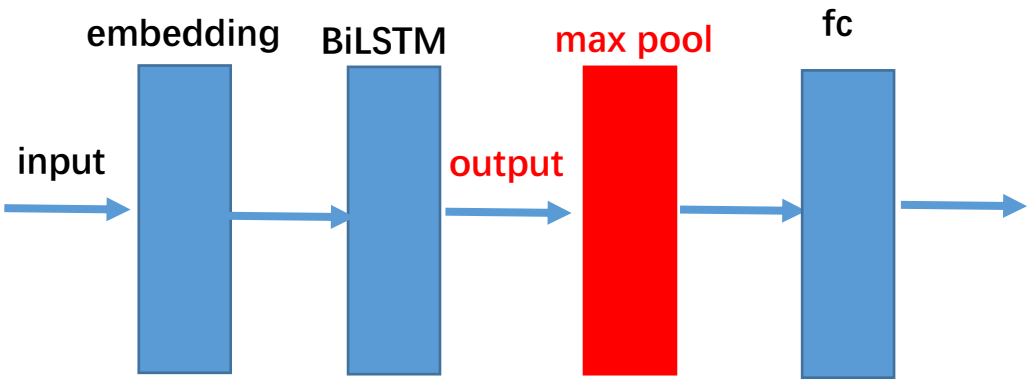
- **模型二 CNN_w2v：和 CNN 的唯一区别是，在 embedding 层导入 word2vec 的 embedding 预训练 weight（初始化），然后再进行训练**
- **模型三 RNN：**1 个 embedding 层+1 层双向 RNN+1 个全连接层



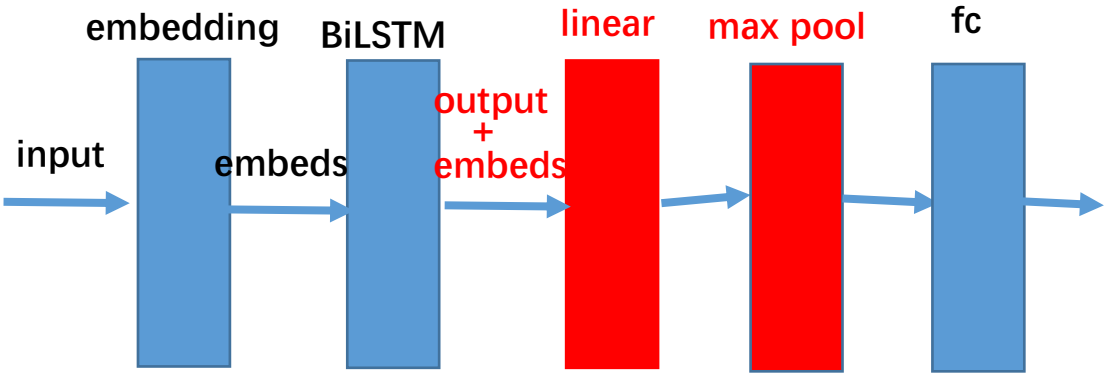
- **模型四 LSTM：**1 个 embedding 层+2 层双向 LSTM+dropout+1 个全连接层



- **模型五 LSTM_maxpool：**1 个 embedding 层+2 层双向 LSTM+max pool+dropout+1 个全连接层（这个模型介于 LSTM 和 RCNN 之间，只是比 LSTM 多了一层 max pool）



- **模型六 RCNN：**1 个 embedding 层+1 层双向 LSTM+1 个线性层+max pool+1 个全连接层



6. 计时 Callback 实现

（代码请看 utils.py）

- 基于 fastNLP 的 Callback，实现了 on_epoch_end 的计时功能，可以用于比较不同模型

的运行时间。

7. Trainer 和 Tester 实现

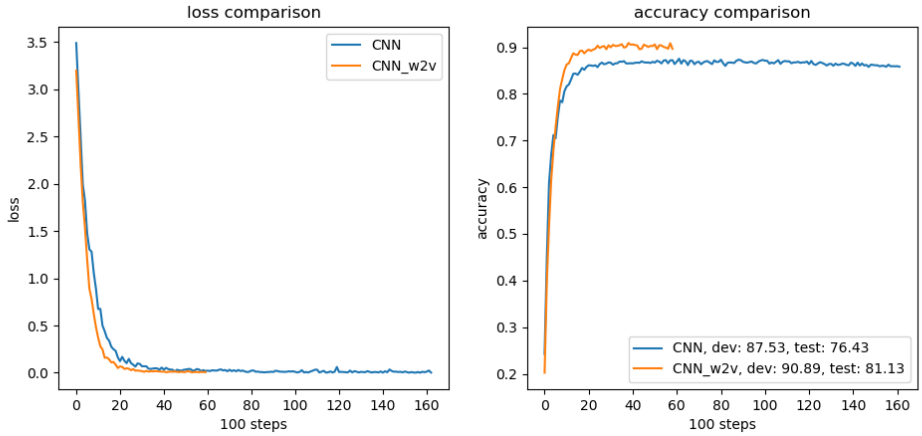
（代码请看 train.py）

- 用 pickle 导入 trian_data、dev_data、test_data 和 vocabulary，以及 word2vec 模型的 embedding 预训 weights
- 根据 config 中的 task_name 参数来定义对应的模型，并导入对应模型所需的参数
- **定义 Adam 的 optimizer**
- 定义计时 Callback、EarlyStop 的 Callback
- 定义 Metric 为 AccuracyMetric
- 然后定义 Trainer，进行训练
- 最后定义 Tester，并在 test_data 上进行测试

8. 模型结果和对比

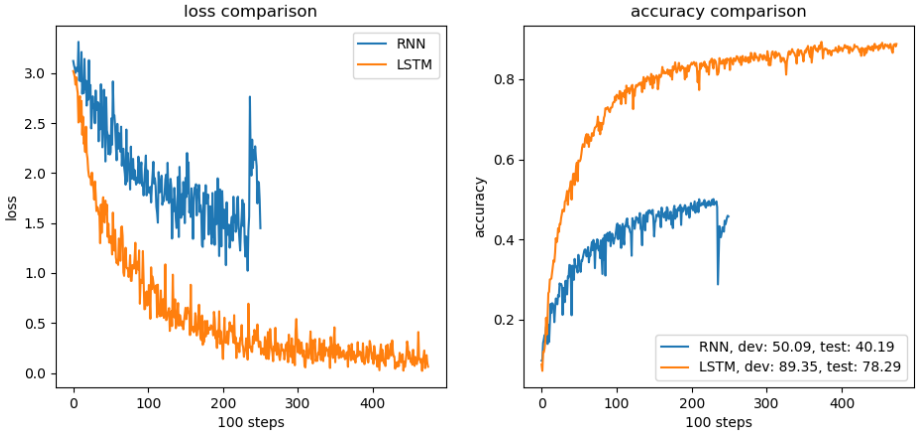
（以下出现的图有统一的格式：左图为 loss 对比，右图为 dev 上的 accuracy 对比，右图的 label 显示了 dev 上的最高 accuracy、test 上的最终 accuracy）

- **对比一：CNN 和 CNN_w2v**



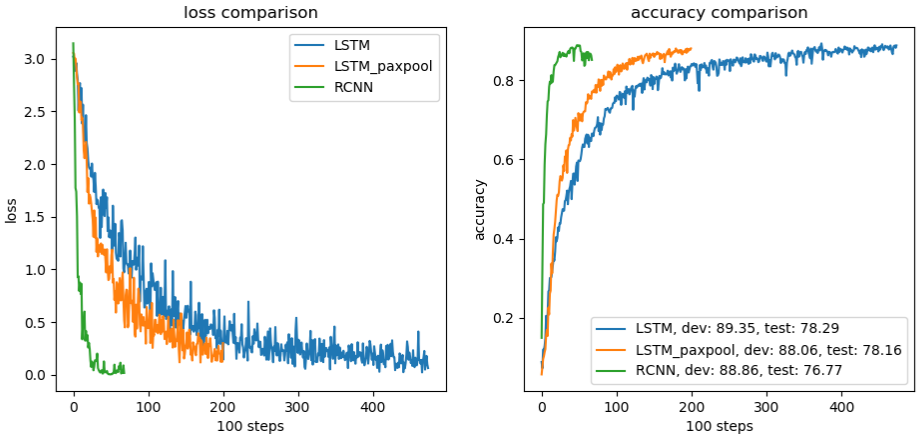
模型	CNN	CNN_w2v
end_epoch	115	43
sum time	3851s	3049s
best dev acc	87.53	90.89
test acc	76.43	81.13
对比	• 用了 word2vec 的 embedding 的 weight 初始化后，模型收敛速度更快，而且最终结果提高了~5%	

- **对比二：RNN 和 LSTM**



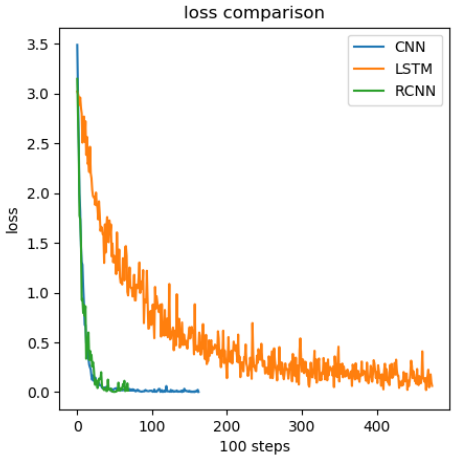
模型	RNN	LSTM
num_layers	1	2
end_epoch	23	42
sum time	6790s	31128s
best dev acc	50.09	89.35
test acc	40.19	78.29
对比	• 层数增加了之后，运行时间显著增加 • 应用了 LSTM 的结构后，accuracy 显著提高，因此如今 LSTM 一般会被作为 vanilla RNN	

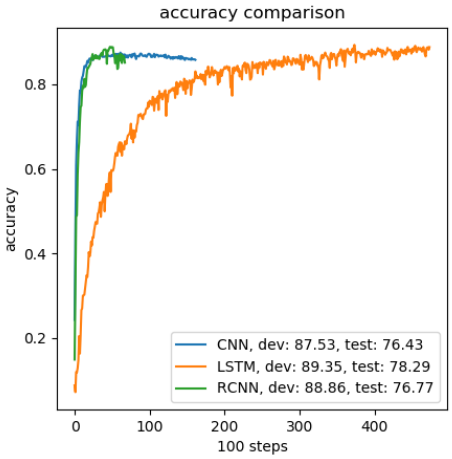
- **对比三： LSTM、LSTM_maxpool 和 RCNN**



模型	LSTM	LSTM_maxpool	RCNN
end_epoch	42	18	7
sum time	31128s	26154s	2887s
best dev acc	89.35	88.06	88.86
test acc	78.29	78.16	76.77
对比	• 应用了 max pool 结构后，收敛更快了，最终 accuracy 也差不多 • 而应用了完整的 RCNN 结构后，收敛还能更快，而且 performance 也能保持在一个水平		

- **对比四： CNN、LSTM 和 RCNN**

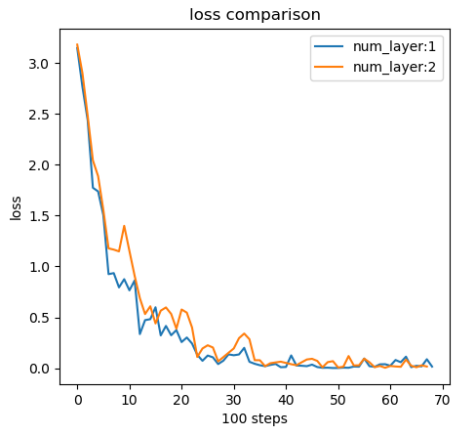


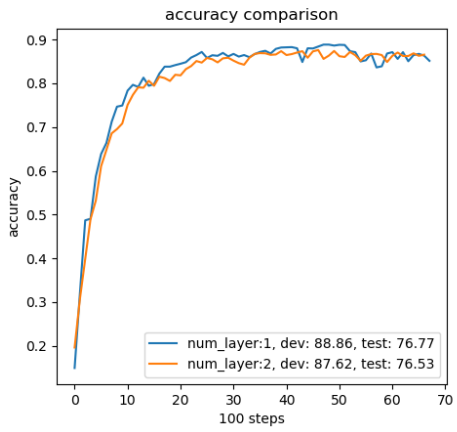


模型	CNN	LSTM	RCNN
end_epoch	115	42	7
sum time	3851s	31128s	2887s
best dev acc	87.53	89.35	88.86
test acc	76.43	78.29	76.77
对比	• CNN 结构的运行时间更快，而具有时序结构的 LSTM 和 RCNN（基于 LSTM）运行时间长了很多 • 三者最终的 accuracy 都差不多 • RCNN 既具有了 LSTM 的特点，也具有了 CNN 的收敛速度，是两者的综合。		

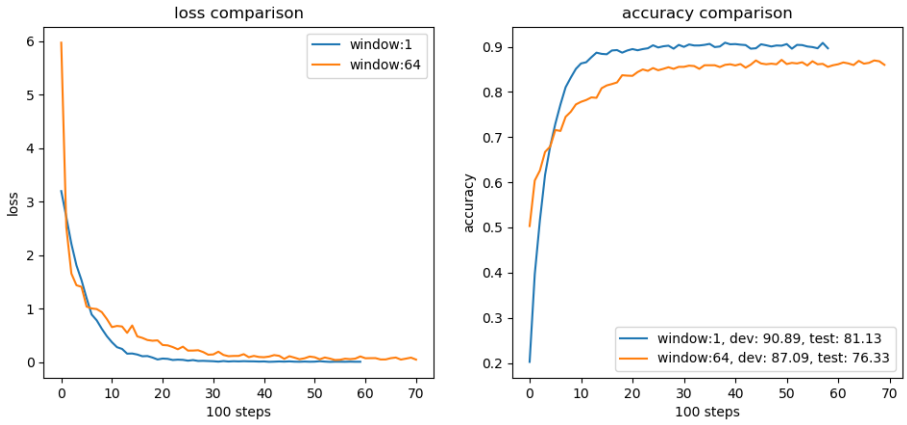
9. 部分模型参数的简单对比

- **对比一： 基于 RCNN 的 num_layers 对比（增加 BiLSTM 层数并没有显著影响）**

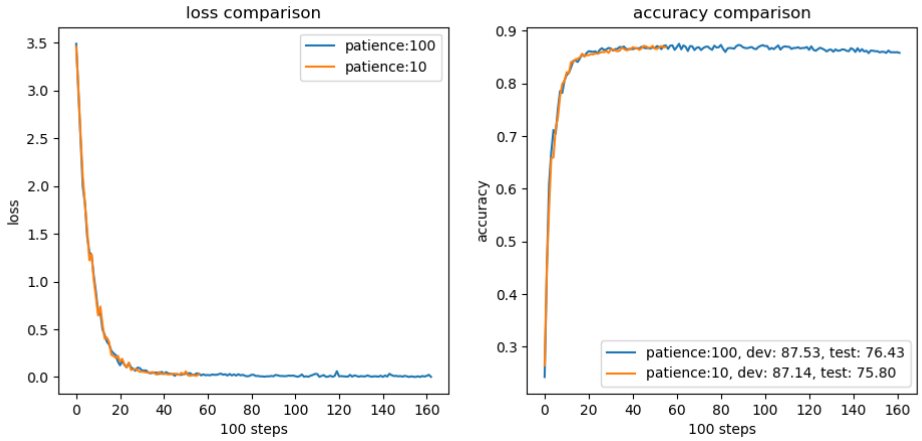




- **对比二： 基于 CNN_w2v 的 window 对比（小的 window 效果更好）**



- **对比三： 基于 CNN 的 patience 对比（patience 不需要设置的很大，不仅不能他提高 accuracy，反而使用了更长的训练时间）**



Part II: Suggestion

1. 使用感觉

- 这样的封装感觉还是很好的，写代码可以更快捷、简洁

2. 建议

- Vocabulary 在初始化的时候可以添加 end_of_sentence='<EOS>' 和 start_of_sentence='<START>'
- 可以导入 dataset 直接生成 Vocabulary
- Trainer 只能传进 1 个 batch_size，但是 train 和 dev 数据其实可以用不同的 batch_size
- 似乎不支持继续训练模型，希望可以支持继续训练模型。
- print_every 输入的是 step 间隔，可以增加一个输入 epoch 的间隔
- 文档里有部分的函数或类是没有使用样例的，如果能像 pytorch 的文档一样有使用样例和输出样例的话，能帮助理解 fastNLP 的使用