Assignment 2

Files

data/

data_utils.py: Helper functions or classes used in data processing.

process.py: Process a raw dataset into a sample file.

model/

config.py: Define configuration parameters.

dataset.py: Define the format of samples used in the model.

evaluate.py: Evaluate the loss in the dev set.

model.py: Define the model.

predict.py: Generate a summary.

train.py: Train the model.

utils.py: Helper functions or classes used for the model.

vocal.py: Define the vocabulary object.

saved_model/

baseline/

Save baseline model.

pgn/

Save PGN model.

cov_pgn/

Save PGN model with coverage mechanism.

ft_pgn/

Save fine-tuned PGN model.

files/

Place data files here.

runs/

Save logs here for TensorboardX.

TO-DO list:

必备资料

为了完成以下任务，我们需要逐步熟悉、掌握Pytorch框架，所以请大家在完成每一个模块时先查阅一下Pytorch的API文档，弄清楚要使用的模块是做什么的以及如何使用。此外，模型的具体实现，需要参见论文 *Get To The Point: Summarization with Pointer-Generator Networks*。

作业2简介

这次我们要在作业1的基础上，实现一个PGN模型。为了达到这个目的，我们只需要在model/config.py这个文件下面，定义一个变量`pointer`，并且在model/model.py中实现PGN的代码后通过这个变量来控制我们的模型是baseline或者PGN；为了实现coverage机制，我们可以用类似的方式，定义一个变量`coverage`来控制是否加入coverage机制；此外，由于论文中作者建议coverage机制在训练好一个收敛的PGN模型加入之后效果更好，所以我们为了实现fine-tune机制，可以定义一个变量`fine-tune`。

总结一下，这一个作业我们需要训练三个模型：

1. PGN，令`pointer = True`即可。
2. PGN (with coverage)，令`pointer = True`以及`coverage = True`
3. PGN (fine-tuned with coverage)，令`pointer = True`，`coverage = True`以及`fine_tune = True`

在saved_model/ 下已经建好相应的文件夹来存放对应的模型。

此外，作业2还需要大家对Beam search实现几种优化，大家在理解了几种normalization的作用之后，可以通过调节超参数来调整效果。

模块1: OOV tokens处理

model/utils.py:

任务1: 完成abstract2ids函数。

由于PGN可以生成在source里面出现过的OOV tokens，所以这次我们对reference的token ids需要换一种映射方式，即将在source里出现过的OOV tokens也记录下来并给一个临时的id，而不是直接替换为"<UNK>" token，以便在训练阶段准确的计算损失。

model/dataset.py:

任务2: 完成SampleDataset类中的__getitem__函数。

用任务1完成的abstract2ids函数来对Y进行处理。

模块2: 实现PGN和coverage

model/model.py:

任务1: 完成Decoder。

1. 定义一个线性层w_gen。
2. 实现p_gen的计算，详见公式(8)。

   你可能用到的Pytorch模块：

   torch.cat

   nn.functional.sigmoid

任务2: 完成Attention。

1. 定义一个线性层w_c。
2. 定义前向传导。

   a. 计算attention weights时加入coverage vector，参考公式(11)。

   b. 对coverage vector进行更新，参考公式(10)。

任务3：实现一个get_final_distribution函数。

所谓的 pointer 本质是根据 attention 的分布 (source 中每个 token 的概率分布)来挑选输出的词，是从 source 中 挑选最佳的 token 作为输出;所谓的 generator 的本质是根据 decoder 计算得到的字典概率分布 P_vocab 来挑选输出的词，是从字典中挑 选最佳的 token 作为输出。所以大家应该能发现:Attention 的分布和 P_vocab 的分布的长度和对应位置代表的 token 是不一样的，所以在计算 final distribution 的时候应该如何对应上呢?

这里推荐的方式是，先对 P_vocab 进行扩展，将 source 中的 oov 添 加到 P_vocab 的尾部，得到 P_vocab_extend 这样 attention weights 中的每一个 token 都能在 P_vocab_extend 中找到对应的位置，然后 将对应的 attention weights 叠加到扩展后的 P_vocab_extend 中的对 应位置，得到 final distribution。

为了做到将 attention weights 这个 tensor 中的值添加到 P_vocab_extend 中对应的位置，你需要使用到 torch.Tensor.scatter_add 这个函数， P_vocab_extend 作为添加值的目标 tensor，attention_weights 作为 添加值的来源 tensor，index 化后的 source 可以作为 attention_weights 的添加依据。

这部分建议大家仔细阅读paper，并理解公式(9)。

你可能用到的Pytorch模块：

torch.cat

torch.Tensor.scatter_add_

任务4: 完成整个model的前向传导。

这里的关键是要实现coverage loss，详见公式(12)(13)。

你可能用到的Pytorch模块：

torch.sum

torch.min

模块3: Beam Search优化

具体实现请参考这个链接：https://opennmt.net/OpenNMT/translation/beam_search/。

model/utils.py

任务1: 实现length normalization和coverage normalization。

这一部分请在Beam类下的seq_score函数中实现。

model/predict.py

任务2: 实现EOS token normalization，并选择一些禁用词汇。

这一部分请在best_k函数中实现。

测试

测试方法与上一个作业一致，这里贴出我们训练的PGN的评估结果，大家可以以此为baseline尝试调参优化。

```
rouge-1
f: 25.701718780532023
p: 29.876133219920998
r: 23.063210272233505
rouge-2
f: 4.522395718902289
p: 5.243118006152585
r: 4.076644687831752
rouge-l
f: 15.571672290083436
p: 20.180543379971503
r: 13.036895653347946
```

模块4(optional)：词汇重复问题

根据你对论文中coverage mechanism的理解，以及实际实验的结果，你觉得这篇论文提出的coverage机制是否能完全解决生成重复词汇的问题呢？如果不能的话，你能解释原因吗？你有没有什么其他的方法，能更好的解决词汇重复的问题呢（模型设计方面或者输出控制方面）？

具体任务描述：写一篇文章，包括一下内容：

1. 将你对论文中coverage机制的理解进行阐述（原理、实际应用）

2. 将你的改进方案进行阐述（最好动手实现并验证结果）

####