
环境安装

1. 升级到python3.6

如果已经是3.6就不用升啦

```
sudo yum update
sudo yum install yum-utils
sudo yum groupinstall development
sudo yum install https://centos7.iuscommunity.org/ius-release.rpm
sudo yum install python36u
python3.6 -V
sudo yum install python36u-pip
sudo yum install python36u-devel
```

2. 安装sqlite3 (仅限阿里云)

启动 `tensorboard` 时需要, 否则报错: `No module named _sqlite3`

```
wget https://www.sqlite.org/2017/sqlite-autoconf-3170000.tar.gz --no-check-certificate
tar zxvf sqlite-autoconf-3170000.tar.gz
cd sqlite-autoconf-3170000
./configure --prefix=/usr/local/sqlite3 --disable-static --enable-fts5 --enable-json1 CFLAGS="-g
-O2 -DSQLITE_ENABLE_FTS3=1 -DSQLITE_ENABLE_FTS4=1 -DSQLITE_ENABLE_RTREE=1"
```

重新编译 `python3.6`

```
wget https://www.python.org/ftp/python/3.6.4/Python-3.6.4.tgz (阿里云速度慢, 可以直接windows上下
载, 上传到阿里云上)
cd Python-3.6.4
LD_RUN_PATH=/usr/local/sqlite3/lib ./configure LDFLAGS="-L/usr/local/sqlite3/lib" CPPFLAGS="-I
/usr/local/sqlite3/include"
LD_RUN_PATH=/usr/local/sqlite3/lib make
LD_RUN_PATH=/usr/local/sqlite3/lib sudo make install
```

3. 创建虚拟环境

```
python3.6 -m venv 'your env name'
```

e.g.: `python3.6 -m venv env-cwmt` ,会在当前目录下生成 `env-cwmt` 目录

4. 激活虚拟环境

```
source 'your env name'/bin/activate
```

e.g.: `source env-cwmt/bin/activate` , 此时你的命令行提示符变成了 `(env-cwmt)***`

5. 设置pip源

将镜像源修改至清华

```
mkdir ~/.pip
cd ~/.pip
vi pip.conf
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
[install]
trusted-host=mirrors.aliyun.com
:wq
```

6. 安装tensorflow

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple tensorflow-gpu==1.3.0
```

如果是阿里云服务器，设置清华大学镜像，速度快； `tensorflow` 使用 `1.3.0` 版本

7. 安装其他package

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple sympy
```

目录结构

- bin/

包括：模型训练，解码，打分*

训练脚本： `train.sh`

解码脚本： `decoder.sh` , `translate_dataset.sh`

打分脚本*： `todo`

- data/

使用的数据集，目录结构是：语言方向 / 版本 - 切分方式 - 数据集，e.g. zh2en/v4-bpe32k-cwmt 表示 v4 版本，切分方式是 bpe32k，数据集是 cwmt

train 目录是 tgt-gen.py 生成的 tfRecord 文件，以及 bpe 结果 eval 目录包括校验集和所有的测试集，每一个评价集合文件内，包括：input.bpe (源语输入的 bpe 结果)，input.token (源语输入的分词结果，实际并没有使用)，ref* (单个/多个 references)

- output/

存放生成的模型，目录结构是：语言方向 / 版本 - 切分方式 - 数据集 / tag

如果使用过 ensemble 解码，还会生成存在 ensemble model 的目录，e.g. ensemble15

- tensor2tensor/

核心代码

- doc/

使用文档，或者实验记录

训练流程

配置介绍

编辑 bin/train.sh，进行配置

硬件

- dev 表示使用的 gpu 设备，例如 dev=0,1,2,3
- gpu_fraction 表示每个 gpu 占用的百分比，e.g. gpu_fraction=0.95，通常不用改

数据集

- lang 表示当前翻译方向，例如 lang=zh2en
- datatype 表示使用数据的类型，例如 datatype=v4-bpe32k
- dataset 表示使用的训练数据集，例如 dataset=cwmt

训练参数

- model 表示使用的已注册模型，e.g. model=transformer
- param 表示使用的已注册参数，e.g. param=transformer_base
- train_step 表示当前模型要更新的次数，e.g. train_step=103000，这是使用 cwmt700w 数据更近似 10epoch 的次数
- other_hparams 表示临时改变的训练参数，常见的用法是调整 batch_size 大小，e.g. other_hparams='batch_size=2048'，否则你必须在代码里注册相关的训练参数
- tag 表示当前跑的实验的名字，e.g. tag=baseline-epoch20，这个名字是方便你记录的。每次跑不同的实验都应该改

使用方法

更新设置后，直接运行 `./train.sh`。该脚本会开始训练，并且会在 `cpu` 上进行多卡的自动校验。此时，你打开 `tensorboard`，观察相应曲线即可

解码流程

配置介绍

硬件

- `device` 表示使用的gpu设备，例如 `device=(0 1 2 3)`
 - 注意，这里的 `device` 是 `shell` 中的数组，不是 `train.sh` 里的字符串，多个值之间用 `空格` 隔开
 - `device` 可以设置多卡，虽然每一个解码程序只能用一个卡，但是我们可以用多卡并行跑不同的实验

评价

- `is_eval` 表示是否进行评价（e.g. 跑 BLEU），`1` 表示解码+评价，`0` 表示只解码
 - 本项目中设置为 `1`
- `eval_tool` 表示使用的评价工具，可选 `multi-bleu` 和 `mteval`
 - 本项目中设置 `multi-bleu`
- `lowercase` 表示评价的时候，是否全小写化，即大小写不敏感，`1` 表示大小写不敏感，`0` 表示大小写敏感
 - 此选项目前只使用于 `multi-bleu`，本项目中设置为 `0`

数据集/词汇表

- `lang` 表示当前翻译方向，例如 `lang=zh2en`
- `datatype` 表示使用数据的类型，例如 `datatype=v4-bpe32k`
- `dataset` 表示使用的训练数据集，例如 `dataset=cwmt`
- `evalset` 表示要被评价的测试集
 - 此选项是 `shell` 的数组，可以设置多个值，用 `空格` 隔开

翻译模型

- `model` 表示使用的已注册模型，e.g. `model=transformer`
- `param` 表示使用的已注册参数，e.g. `param=transformer_base`
- `tag` 表示该模型对应的实验名称，e.g. `tag=baseline`，**注意修改**

解码参数

- `beam_size` 表示 `beam` 大小，通常是 `12`
- `batch_size` 表示解码时一个 `batch` 的大小，通常是 `32`，如果出现 `OOM` 错误，可以降低该值
- `alphas` 表示长度归一化 (Layer Normalization, LN) 的系数

- 此选项是 `shell` 的数组，可以设置多个值，程序能够自动进行 `grid search`，寻找最优 `alpha`
- `ensemble` 表示用来 `checkpoint average` 的数量
 - 如果使用单模型，不要设置这个参数值，空着就行
 - 如果想使用ensemble, 例如最后 `15` 个模型平均，则设置此选项为 `15` 即可

使用方法

解码流程主要包括4个步骤：

- 使用单模型，在 `校验集` 上搜索最好的超参数 `alpha`
- 使用单模型，用最好的 `alpha` 跑所有的 `测试集`
- 使用 `ensemble` 模型，在 `校验集` 上搜索最好的超参数 `alpha`
- 使用 `ensemble` 模型，用最好的 `alpha` 跑所有的 `测试集`

`ensemble` 模型的超参数 `alpha` 不一定每次都需要搜索，或者搜索范围可以小一点（一般跟单模型 `alpha` 差不多，不会相差特别悬殊）

可以根据最后解码报告的 `bp` 和 `ratio` 值来判断是否应该再调整 `alpha`

每次跑的多组解码结果，会在程序结束后汇总在一起输出

假设你刚使用 `4卡(0,1,2,3)` 训练完模型，设置 `数据集` & `翻译模型` 的相关参数（类似 `train.sh`）

1. 单模型，搜索超参数

- 设置解码用的gpu，能多用就多用

```
device=(0 1 2 3)
```

- 设置校验集

```
evalset=(cwmt18-dev)
```

- 设置要grid search的超参数

```
alphas=(0.9 1.0 1.1 1.2 1.3)
```

这里使用了5个值，可以比gpu的卡数多，程序会自动调度

- 设置ensemble为空

```
ensemble=
```

2. 单模型，跑测试集

- 设置解码用的gpu，能多用就多用

```
device=(0 1 2 3)
```

- 设置测试集，一共8个

```
evalset=(cwmt17-dev wmt17-test mt06 mt08 mt12-nd mt12-nw mt12-wb exact2k)
```

- 设置超参数，根据上一步的最优值，假设是 1.2

```
alphas=(1.2)
```

- 设置ensemble为空

```
ensemble=
```

3. ensemble模型，搜索超参数

- 设置解码用的gpu，能多用就多用

```
device=(0 1 2 3)
```

- 设置校验集

```
evalset=(cwmt18-dev)
```

- 设置要grid search的超参数，不用像单模型那么多，在单模型最优值附近搜索就行

```
alphas=(1.1 1.2 1.3)
```

- 设置ensemble，假设使用 15 个模型来平均

```
ensemble=15
```

4. ensemble模型，跑测试集

- 设置解码用的gpu，能多用就多用

```
device=(0 1 2 3)
```

- 设置测试集，一共8个

```
evalset=(cwmt17-dev wmt17-test mt06 mt08 mt12-nd mt12-nw mt12-wb exact2k)
```

- 设置超参数，根据上一步的最优值，假设是 1.2

```
alphas=(1.2)
```

- 设置ensemble

```
ensemble=15
```