NML-Opennmt EN-ZH

- Last updated: 2021-06-21
- Project_files: ubuntu@52.10.208.122/home/ubuntu/caodongnan/work/nmt_opennnmt

Dataset

- Title: UN Parallel Corpus V1.0
- Description: Training English-Chinese nerual machince translation model based on Opennmt framework. This document describes the detailed training precedures.

e.g.

```
RESOLUTION 918 (1994)
Adopted by the Security Council at its 3377th meeting, on 17 May 1994
The Security Council,
Reaffirming all its previous resolutions on the situation in Rwanda, in
particular its resolution 872 (1993) of 5 October 1993 by which it
established the United Nations Assistance Mission for Rwanda (UNAMIR), its
resolution 909 (1994) of 5 April 1994 which extended the mandate of UNAMIR
until 29 July 1994, and its resolution 912 (1994) of 21 April 1994 by
which it adjusted the mandate of UNAMIR,
Recalling the statements made by the President of the Council on 7 April
1994 (S/PRST/1994/16) and 30 April 1994 (S/PRST/1994/21),

第918(1994)号决议
1994年5月17日安全理事会第3377次会议通过
安全理事会，
重申其以往关于卢旺达局势的所有决议，特别是成立联合国卢旺达援助团(联卢援助团)的1993年10月
5日第872(1993)号决议，延长联卢援助团任务期限至1994年7月29日的1994年4月5日第
909(1994)号决议，以及调整联卢援助团的任务规定的1994年4月21日第912(1994)号决议，
回顾安理会主席以安理会名义在1994年4月7日发表的声明(S/PRST/ 1994/16)和在1994年4月30日
发表的声明(S/PRST/1994/21)，
```

Pre-processing

Before feeding the data into the model, we need to preprocess the raw data following these steps listed below.

Tagging

- Add tags

  encode the num/date/amount/price/custom into tags, which can be decoded into standard form later. [code: ./nmt_trans/tags/]

  here we simply use rule based approached to add tags.

e.g.

```
org:
2000/1 2011/5 2011/4/1, 5 December,  14 Dec 2011,  03rd July 2018,  05/6-
07/10, 2015 Apr, Nov 1990.
,2011/4/1，5月3日，6月1号到5日 5月1号至8月1日，2018年7月03号，05/6-07/10, 15年4月

enc:
《d0》 《d3》 《d6》, 《d5》, 《d4》, 《d1》, 05/6-07/10, 2015 Apr, 《d2》.
,《d2》, 《d0》，6月1号到5日 5月1号至8月1日，《d1》，05/6-07/10, 15年4月

dec:
('2000/1 2011/5 2011/4/1, 5 Dec,  14 Dec 2011,  3 Jul 2018,  05/6-07/10,
2015 Apr, Nov 1990.', True)
(',2011/4/1，5月3日，6月1号到5日 5月1号至8月1日，2018年7月3日，05/6-07/10, 15年4
月', True)
```

- Remove duplicated tags

  due to the up-sampling

Data cleaning

- Remove white space

  exclude '\s'

- Punctuation normalization

  apply MosesPunctNormalizer to English sentence, including 'replace unicode puncts'.

  check https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/replace-unicode-punctuation.perl

  e.g. (u", ", u","),(r"。\s*", u". "),(u"、 ", u","),(u"”", u'"'),...

- Filtering

  in case where source and target sentances are in the same language.

Tokenization

- Chinese tokenization

  based on Jieba tokenizer, split the sentance into tokens.

e.g.

```
$ head -5
/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../../data/offlin
e_data/raw_para_corpus/all_1kw_tagged_v2/UNv1.0.en-zh.zh
```

```
第918(1994)号决议
1994年5月17日安全理事会第3377次会议通过
安全理事会，
重申其以往关于卢旺达局势的所有决议，特别是成立联合国卢旺达援助团(联卢援助团)的1993年10月5日第872(1993)号决议，延长联卢援助团任务期限至1994年7月29日的1994年4月5日第909(1994)号决议，以及调整联卢援助团的任务规定的1994年4月21日第912(1994)号决议，
回顾安理会主席以安理会名义在1994年4月7日发表的声明(S/PRST/ 1994/16)和在1994年4月30日发表的声明(S/PRST/1994/21)，

$ head -5
/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../../data/offline_data/prep_corpus/prep_1kw_tagged_v2/tmp/UNv1.0.en-zh.tok.zh

第 918 ( 1994 ) 号 决议
1994 年 5 月 17 日 安全 理事会 第 3377 次 会议 通过
安全 理事会 ，
重申 其 以往 关于 卢旺达 局势 的 所有 决议 ， 特别 是 成立 联合国 卢旺达 援助团 ( 联卢援助团 ) 的 1993 年 10 月 5 日 第 872 ( 1993 ) 号 决议 ， 延长 联卢 援助团 任务 期限 至 1994 年 7 月 29 日 的 1994 年 4 月 5 日 第 909 ( 1994 ) 号 决议 ， 以及 调整 联卢 援助团 的 任务 规定 的 1994 年 4 月 21 日 第 912 ( 1994 ) 号 决议 ，
回顾 安理会 主席 以 安理会 名义 在 《d24》 发表 的 声明 ( S / PRST / 1994 / 16 ) 和 在 《d25》 发表 的 声明 ( S / PRST / 1994 / 21 ) ，
```

- English tokenization

  based on Moses toknizer, split the sentance into tokens.

e.g.

```
$ head -5
/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../../data/offline_data/raw_para_corpus/all_1kw_tagged_v2/UNv1.0.en-zh.en

RESOLUTION 918 (1994)
Adopted by the Security Council at its 3377th meeting, on 17 May 1994
The Security Council,
Reaffirming all its previous resolutions on the situation in Rwanda, in particular its resolution 872 (1993) of 《d5》 by which it established the United Nations Assistance Mission for Rwanda (UNAMIR), its resolution 909 (1994) of 《d6》 which extended the mandate of UNAMIR until 29 July 1994, and its resolution 912 (1994) of 《d7》 by which it adjusted the mandate of UNAMIR,
Recalling the statements made by the President of the Council on 《d22》 (S/PRST/1994/16) and 《d23》 (S/PRST/1994/21),

$ head -5
/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../../data/offline_data/prep_corpus/prep_1kw_tagged_v2/tmp/UNv1.0.en-zh.tok.en

RESOLUTION 918 ( 1994 )
Adopted by the Security Council at its 3377th meeting , on 17 May 1994
```

```
The Security Council ,
Reaffirming all its previous resolutions on the situation in Rwanda , in
particular its resolution 872 ( 1993 ) of 5 October 1993 by which it
established the United Nations Assistance Mission for Rwanda ( UNAMIR ) ,
its resolution 909 ( 1994 ) of 5 April 1994 which extended the mandate of
UNAMIR until 29 July 1994 , and its resolution 912 ( 1994 ) of 21 April
1994 by which it adjusted the mandate of UNAMIR ,
Recalling the statements made by the President of the Council on 《d22》 (
S / PRST / 1994 / 16 ) and 《d23》 ( S / PRST / 1994 / 21 ) ,
```

BPE

Here, we use **subword_nmt** to build bpe code on training data and apply to all data.

```
from subword_nmt import learn_bpe, apply_bpe

# learn bpe
BPE_TOKENS = 35000
learn_bpe.learn_bpe(in_, out_, BPE_TOKENS)

# apply bpe
bpe = apply_bpe.BPE(f_code, glossaries=tag_tokens)
res = bpe.process_line(line)
```

e.g.

```
$ head -5
/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../../data/offlin
e_data/prep_corpus/prep_1kw_tagged_v2/train.en

Reaffirming all its previous resolutions on the situation in Rwanda , in
particular its resolution 8@@ 72 ( 1993 ) of 5 October 1993 by which it
established the United Nations Assistance Mission for Rwanda ( UNAMI@@ R )
, its resolution 90@@ 9 ( 1994 ) of 5 April 1994 which extended the
mandate of UNAMI@@ R until 29 July 1994 , and its resolution 9@@ 12 ( 1994
) of 21 April 1994 by which it adjusted the mandate of UNAMI@@ R ,
Recalling the statements made by the President of the Council on 《d24》 (
S / PRST / 1994 / 16 ) and 《d25》 ( S / PRST / 1994 / 21 ) ,
Having considered the report of the Secretary-General dated 《d24》 ( S /
1994 / 565 ) ,
Reaffirming its resolution 8@@ 68 ( 1993 ) of 29 September 1993 on the
security of United Nations operations ,
Strongly condemning the ongoing violence in Rwanda and particularly
condemning the very numerous killings of civilians which have taken place
in Rwanda and the impunity with which armed individuals have been able to
operate and continue operating therein ,
```

```
$ head
/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../../data/offlin
e_data/prep_corpus/prep_1kw_tagged_v2/train.zh

重申 其 以往 关于 卢旺达 局势 的 所有 决议 ， 特别 是 成立 联合国 卢旺达 援助团 （ 联卢
援助团 ） 的 1993 年 10 月 5 日 第 872 （ 1993 ） 号 决议 ， 延长 联卢 援助团 任务 期
限 至 1994 年 7 月 29 日 的 1994 年 4 月 5 日 第 909 （ 1994 ） 号 决议 ， 以及 调
整 联卢 援助团 的 任务 规定 的 1994 年 4 月 21 日 第 912 （ 1994 ） 号 决议 ，
回顾 安理会 主席 以 安理会 名义 在 《d24》 发表 的 声明 （ S / PRST / 1994 / 16 ）
和 在 《d25》 发表 的 声明 （ S / PRST / 1994 / 21 ） ，
审议 了 《d24》 秘书长 的 报告 （ S / 1994 / 565 ） ，
重申 其 1993 年 9 月 29 日 关于 联合国 行动 安全 的 第 868 （ 1993 ） 号 决议 ，
强烈 谴责 卢旺达 境内 目前 发生 的 暴乱 ， 特别 是 谴责 在 卢旺达 境内 发生 大量 残杀
平民 ， 以及 武装 个人 在 国内 一直 横@@ 行 并 继续 横@@ 行 而 不受 惩罚 的 情况 ，
```

process input and vocab

This is the last part before training, we need to process the input '.pt' and build the vocab fields.

```
cd nmt_trans/preporcess
python process_for_nmt.py ../conf/en_zh_config.json
```

the main function is located at ./onmt/bin/preprocess

```
def preprocess(opt):
    ArgumentParser.validate_preprocess_args(opt)
    torch.manual_seed(opt.seed)

    init_logger(opt.log_file)

    logger.info("Extracting features...")

    src_nfeats = 0
    tgt_nfeats = 0
    for src, tgt in zip(opt.train_src, opt.train_tgt):
        src_nfeats += count_features(src) if opt.data_type == 'text' \
            else 0
        tgt_nfeats += count_features(tgt)  # tgt always text so far
    logger.info(" * number of source features: %d." % src_nfeats)
    logger.info(" * number of target features: %d." % tgt_nfeats)

    logger.info("Building `Fields` object...")
    fields = inputters.get_fields(
        opt.data_type,
        src_nfeats,
        tgt_nfeats,
        dynamic_dict=opt.dynamic_dict,
        with_align=opt.train_align[0] is not None,
```

```
        src_truncate=opt.src_seq_length_trunc,
        tgt_truncate=opt.tgt_seq_length_trunc)

    src_reader = inputters.str2reader[opt.data_type].from_opt(opt)
    tgt_reader = inputters.str2reader["text"].from_opt(opt)
    align_reader = inputters.str2reader["text"].from_opt(opt)

    # corpus_type must be in ['train', 'valid']
    logger.info("Building & saving training data...")
    build_save_dataset(
        'train', fields, src_reader, tgt_reader, align_reader, opt)

    if opt.valid_src and opt.valid_tgt:
        logger.info("Building & saving validation data...")
        build_save_dataset(
            'valid', fields, src_reader, tgt_reader, align_reader, opt)
```

the main function in `build_save_dataset` is `shard_iterator` and `process_one_shard`, later use pool method to build dataset in parallel

```
def shard_iterator(srcs, tgts, ids, aligns, existing_shards,
                       existing_fields, corpus_type, opt):
        """
        Builds a single iterator yielding every shard of every corpus.
        """
        for src, tgt, maybe_id, maybe_align in zip(srcs, tgts, ids,
aligns):
            if maybe_id in existing_shards:
                if opt.overwrite:
                    logger.warning("Overwrite shards for corpus {}"
                                   .format(maybe_id))
                else:
                    if corpus_type == "train":
                        assert existing_fields is not None,\
                            ("A 'vocab.pt' file should be passed to "
                             "`-src_vocab` when adding a corpus to "
                             "a set of already existing shards.")
                    logger.warning("Ignore corpus {} because "
                                   "shards already exist"
                                   .format(maybe_id))
                    continue
            if ((corpus_type == "train" or opt.filter_valid)
                    and tgt is not None):
                filter_pred = partial(
                    inputters.filter_example,
                    use_src_len=opt.data_type == "text",
                    max_src_len=opt.src_seq_length,
                    max_tgt_len=opt.tgt_seq_length)
            else:
                filter_pred = None
            src_shards = split_corpus(src, opt.shard_size)
```

```
            tgt_shards = split_corpus(tgt, opt.shard_size)
            align_shards = split_corpus(maybe_align, opt.shard_size)
            for i, (ss, ts, a_s) in enumerate(
                    zip(src_shards, tgt_shards, align_shards)):
                yield (i, (ss, ts, a_s, maybe_id, filter_pred))
```

for each shard, do processing:

```
def process_one_shard(corpus_params, params):
    corpus_type, fields, src_reader, tgt_reader, align_reader, opt,\
         existing_fields, src_vocab, tgt_vocab = corpus_params
    i, (src_shard, tgt_shard, align_shard, maybe_id, filter_pred) = params
    # create one counter per shard
    sub_sub_counter = defaultdict(Counter)
    assert len(src_shard) == len(tgt_shard)
    logger.info("Building shard %d." % i)

    src_data = {"reader": src_reader, "data": src_shard, "dir":
opt.src_dir}
    tgt_data = {"reader": tgt_reader, "data": tgt_shard, "dir": None}
    align_data = {"reader": align_reader, "data": align_shard, "dir":
None}
    _readers, _data, _dir = inputters.Dataset.config(
        [('src', src_data), ('tgt', tgt_data), ('align', align_data)])

    dataset = inputters.Dataset(
        fields, readers=_readers, data=_data, dirs=_dir,
        sort_key=inputters.str2sortkey[opt.data_type],
        filter_pred=filter_pred,
        corpus_id=maybe_id
    )
    if corpus_type == "train" and existing_fields is None:
        for ex in dataset.examples:
            sub_sub_counter['corpus_id'].update(
                ["train" if maybe_id is None else maybe_id])
            for name, field in fields.items():
                if ((opt.data_type == "audio") and (name == "src")):
                    continue
                try:
                    f_iter = iter(field)
                except TypeError:
                    f_iter = [(name, field)]
                    all_data = [getattr(ex, name, None)]
                else:
                    all_data = getattr(ex, name)
                for (sub_n, sub_f), fd in zip(
                        f_iter, all_data):
                    has_vocab = (sub_n == 'src' and
                                 src_vocab is not None) or \
                                (sub_n == 'tgt' and
                                 tgt_vocab is not None)
```

```
                    if (hasattr(sub_f, 'sequential')
                            and sub_f.sequential and not has_vocab):
                        val = fd
                        sub_sub_counter[sub_n].update(val)
    if maybe_id:
        shard_base = corpus_type + "_" + maybe_id
    else:
        shard_base = corpus_type
    data_path = "{:s}.{:s}.{:d}.pt".\
        format(opt.save_data, shard_base, i)

    logger.info(" * saving %sth %s data shard to %s."
                % (i, shard_base, data_path))

    dataset.save(data_path)

    del dataset.examples
    gc.collect()
    del dataset
    gc.collect()

    return sub_sub_counter
```

When processing each shard, Counters are updated, later the vocab fields will be build if not exits.

```
# onmt/bin/preprocess.py
if corpus_type == "train":
        vocab_path = opt.save_data + '.vocab.pt'
        if existing_fields is None:
            fields = _build_fields_vocab(
                fields, counters, opt.data_type,
                opt.share_vocab, opt.vocab_size_multiple,
                opt.src_vocab_size, opt.src_words_min_frequency,
                opt.tgt_vocab_size, opt.tgt_words_min_frequency,
                subword_prefix=opt.subword_prefix,
                subword_prefix_is_joiner=opt.subword_prefix_is_joiner)
        else:
            fields = existing_fields
        torch.save(fields, vocab_path)
```

Training

In this session, we will explain how to use **Opennmt** to build our nmt model.

The simplest way to train opennmt model is preparing the data in a proper format and defining the training config, which identifis all the needed hyperparameters and data/model paths.

```
# import opennmt
from onmt.bin import train
```

```
from nmt_trans.utils import conf_parser

# opennmt parser
parser = train._get_parser()
# the concrete config 'args_dict' is listed below
# convert it into opennmt args
args_arr = conf_parser.dict2args(args_dict)
args, _ = parser.parse_known_args(args_arr)

# start training
train.train(args)
```

this is the trainging config for transformer-base:

```
{
train_info:{
    "train_fmt_path": "data/offline_data/nmt_1kw_v2/tagged_en",
    "src_seq_length": 200,
    "tgt_seq_length": 200,
    "layers": 6,
    "rnn_size": 512,
    "word_vec_size": 512,
    "transformer_ff": 2048,
    "heads": 8,
    "encoder_type": "transformer",
    "decoder_type": "transformer",
    "position_encoding": null,
    "train_steps": 80000,
    "max_generator_batches": 2,
    "dropout": 0.1,
    "batch_size": 4096,
    "valid_batch_size": 32,
    "batch_type": "tokens",
    "normalization": "tokens",
    "accum_count": 8,
    "optim": "adam",
    "adam_bta2": 0.998,
    "decay_method": "noam",
    "warmup_steps": 8000,
    "learing_rate": 2,
    "max_grad_norm": 0,
    "param_init": 0,
    "param_init_glorot": null,
    "label_smoothing": 0.1,
    "keep_checkpoint": 4000,
    "valid_steps": 4000,
    "save_checkpoint_steps": 4000,
    "world_size": 1,
    "gpu_ranks": [0]

  "model_path": "data/offline_data/model_bin/nmt_en_zh/cht",
  "eval_info": {
```

```
    "model_suffix": "_avg.pt",
    "quantization": "float16",
    "replace_unk": null,
    "verbose": null,
    "output": "data/offline_data/val_pred.zh"
  },
  "pred_info": {
    "c_model_path": "data/online_data/nmt_en_zh.pt",
    "c_translate_thread": 8,
    "model_spec": "TransformerBase"
  }
}
```

Now let's training our model:

```
cd nmt_trans/train
nohup python train_nmt.py ../conf/en_zh_config.json > train_log.txt &
```

So far, we have learned how to use opennmt in the simplest way. However, if we want to change the encoder-decoder architecture for different tasks, we have to modify the code. You can find the opennmt code here and the corresponding doc.

One thing needed to be noticed is that the Opennmt-py has been updated to 2.0 version while we are using 1.0 version in this project.

Now let's dive into the onmt code.

.....

Config parsing

```
from onmt.bin import train

# this is the main code for training
# args_arr is the list of the training config
"""
arges_arr = ['--data',
'/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../../data/offli
ne_data/nmt_1kw_v2/tagged_en', '--save_model',
'/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../../data/offli
ne_data/model_bin/nmt_en_zh/cht', '--train_fmt_path',
'data/offline_data/nmt_1kw_v2/tagged_en', '--src_seq_length', '200', '--
tgt_seq_length', '200', '--layers', '6', '--rnn_size', '512', '--
word_vec_size', '512', '--transformer_ff', '2048', '--heads', '8', '--
encoder_type', 'transformer', '--decoder_type', 'transformer', '--
position_encoding', '--train_steps', '80000', '--max_generator_batches',
'2', '--dropout', '0.1', '--batch_size', '4096', '--valid_batch_size',
'32', '--batch_type', 'tokens', '--normalization', 'tokens', '--
accum_count', '8', '--optim', 'adam', '--adam_bta2', '0.998', '--
```

```
decay_method', 'noam', '--warmup_steps', '8000', '--learing_rate', '2', '-
-max_grad_norm', '0', '--param_init', '0', '--param_init_glorot', '--
label_smoothing', '0.1', '--keep_checkpoint', '20', '--valid_steps',
'4000', '--save_checkpoint_steps', '4000', '--world_size', '1', '--
gpu_ranks', '0']
"""

parser = train._get_parser()
args, _ = parser.parse_known_args(args_arr)
train.train(args)
```

How is the onmt parser initialized and how does the parser work?

```
# train._get_parser()
# ArgumentParser is a class that can check/update model options
def _get_parser():
    parser = ArgumentParser(description='train.py')

    opts.config_opts(parser)
    opts.model_opts(parser)
    opts.train_opts(parser)
    return parser

# then it will call the following methods to parse the config
# you can find all the predefined options in ./onmt/bin/opts.py to modify
your config file
def config_opts(parser):
    parser.add('-config', '--config', required=False,
               is_config_file_arg=True, help='config file path')
    parser.add('-save_config', '--save_config', required=False,
               is_write_out_config_file_arg=True,
               help='config file save path')

def model_opts(parser):
    """
    These options are passed to the construction of the model.
    Be careful with these as they will be used during translation.
    """

    # Embedding Options
    # Encoder-Decoder Options
    # Attention options
    # Alignement options
    # Generator and loss options

def train_opts(parser):
    """ Training and saving options """
    """
    --data
    --data-ids
    --data_weights
```

```
    --data_to_noise
    --save_model
    --save_checkpoint_steps
    --keep_checkpoint
    # GPU
    # Init options
    # Pretrained word vectors
    # Optimization options
    # learning rate
    # logging
    # Use Tensorboard for visualization during training
    # Options most relevant to speech
    # Option most relevant to image input
    """
```

When getting the parser, parse the arguments array, which will return the args for training.

```
arges,_ = parser.parse_known_args(args_arr)

"""
args = Namespace(aan_useffn=False, accum_count=[8], accum_steps=[0],
adagrad_accumulator_init=0, adam_beta1=0.9, adam_beta2=0.999,
alignment_heads=0, alignment_layer=-3, apex_opt_level='O1',
attention_dropout=[0.1], audio_enc_pooling='1', average_decay=0,
average_every=1, batch_size=4096, batch_type='tokens', bridge=False,
brnn=None, cnn_kernel_width=3, config=None, context_gate=None,
copy_attn=False, copy_attn_force=False, copy_attn_type=None,
copy_loss_by_seqlength=False, coverage_attn=False,
data='/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../../data/
offline_data/nmt_1kw_v2/tagged_en', data_ids=[None], data_to_noise=[],
data_weights=[1], dec_layers=2, dec_rnn_size=500, decay_method='noam',
decay_steps=10000, decoder_type='transformer', dropout=[0.1],
dropout_steps=[0], early_stopping=0, early_stopping_criteria=None,
enc_layers=2, enc_rnn_size=500, encoder_type='transformer', epochs=0,
exp='', exp_host='', feat_merge='concat', feat_vec_exponent=0.7,
feat_vec_size=-1, fix_word_vecs_dec=False, fix_word_vecs_enc=False,
full_context_alignment=False, generator_function='softmax',
global_attention='general', global_attention_function='softmax',
gpu_backend='nccl', gpu_ranks=[0], gpu_verbose_level=0, gpuid=[], heads=8,
image_channel_size=3, input_feed=1, keep_checkpoint=20,
label_smoothing=0.1, lambda_align=0.0, lambda_coverage=0.0, layers=6,
learning_rate=1.0, learning_rate_decay=0.5, log_file='',
log_file_level='0', loss_scale=0, master_ip='localhost',
master_port=10000, max_generator_batches=2, max_grad_norm=0.0,
max_relative_positions=0, model_dtype='fp32', model_type='text',
normalization='tokens', optim='adam', param_init=0.0,
param_init_glorot=True, pool_factor=8192, position_encoding=True,
pre_word_vecs_dec=None, pre_word_vecs_enc=None, queue_size=40,
report_every=50, reset_optim='none', reuse_copy_attn=False, rnn_size=512,
rnn_type='LSTM', sample_rate=16000, save_checkpoint_steps=4000,
```

```
save_config=None,
save_model='/home/ubuntu/caodongnan/work/nmt_opennmt/nmt_trans/utils/../..
/data/offline_data/model_bin/nmt_en_zh/cht', seed=-1,
self_attn_type='scaled-dot', share_decoder_embeddings=False,
share_embeddings=False, single_pass=False, src_noise=[], src_noise_prob=
[], src_word_vec_size=500, start_decay_steps=50000, tensorboard=False,
tensorboard_log_dir='runs/onmt', tgt_word_vec_size=500, train_from='',
train_steps=80000, transformer_ff=2048, truncated_decoder=0,
valid_batch_size=32, valid_steps=4000, warmup_steps=8000,
window_size=0.02, word_vec_size=512, world_size=1)
"""
```

How does the training work? Opennmt Trainer is defined at ./onmt/train.py

```
from onmt.train_single import main as single_main

# input: the args you got before
def train(opt):
    ArgumentParser.validate_train_opts(opt)
    ArgumentParser.update_model_opts(opt)
    ArgumentParser.validate_model_opts(opt)

    set_random_seed(opt.seed, False)

    # Load checkpoint if we resume from a previous training.
    if opt.train_from:
        logger.info('Loading checkpoint from %s' % opt.train_from)
        checkpoint = torch.load(opt.train_from,
                                map_location=lambda storage, loc: storage)
        logger.info('Loading vocab from checkpoint at %s.' %
opt.train_from)
        vocab = checkpoint['vocab']
    else:
        vocab = torch.load(opt.data + '.vocab.pt')

    # check for code where vocab is saved instead of fields
    # (in the future this will be done in a smarter way)
    # vocab: a list of (field name, torchtext.vocab.Vocab) pairs. This is
the
    #        format formerly saved in *.vocab.pt files. Or, text data
    #        not using a :class:`TextMultiField`.
    # load_old_vocab will constructure fileds from Vocab, ['src',
'tgt','indices','corpus_id']
    if old_style_vocab(vocab):
        fields = load_old_vocab(
            vocab, opt.model_type, dynamic_dict=opt.copy_attn)
    else:
        fields = vocab

    if len(opt.data_ids) > 1:
        train_shards = []
        for train_id in opt.data_ids:
```

```
            shard_base = "train_" + train_id
            train_shards.append(shard_base)
        train_iter = build_dataset_iter_multiple(train_shards, fields,
opt)
    else:
        if opt.data_ids[0] is not None:
            shard_base = "train_" + opt.data_ids[0]
        else:
            shard_base = "train"
        train_iter = build_dataset_iter(shard_base, fields, opt)

    nb_gpu = len(opt.gpu_ranks)

    if opt.world_size > 1:
        queues = []
        mp = torch.multiprocessing.get_context('spawn')
        semaphore = mp.Semaphore(opt.world_size * opt.queue_size)
        # Create a thread to listen for errors in the child processes.
        error_queue = mp.SimpleQueue()
        error_handler = ErrorHandler(error_queue)
        # Train with multiprocessing.
        procs = []
        for device_id in range(nb_gpu):
            q = mp.Queue(opt.queue_size)
            queues += [q]
            procs.append(mp.Process(target=run, args=(
                opt, device_id, error_queue, q, semaphore), daemon=True))
            procs[device_id].start()
            logger.info(" Starting process pid: %d  " %
procs[device_id].pid)
            error_handler.add_child(procs[device_id].pid)
        producer = mp.Process(target=batch_producer,
                              args=(train_iter, queues, semaphore, opt,),
                              daemon=True)
        producer.start()
        error_handler.add_child(producer.pid)

        for p in procs:
            p.join()
        producer.terminate()

    elif nb_gpu == 1:  # case 1 GPU only
        single_main(opt, 0)
    else:   # case only CPU
        single_main(opt, -1)


    # build model(seq2seq or lm)
    # available encoders:
    # str2enc = {"ggnn": GGNNEncoder, "rnn": RNNEncoder, "brnn":
RNNEncoder,
    #      "cnn": CNNEncoder, "transformer": TransformerEncoder,
    #      "mean": MeanEncoder}
    # available decoders:
```

```
    #str2dec = {"rnn": StdRNNDecoder, "ifrnn": InputFeedRNNDecoder,
    #      "cnn": CNNDecoder, "transformer": TransformerDecoder,
    #      "transformer_lm": TransformerLMDecoder}
```

Evaluation

When training is finished, we can average the checkpoints or simply choose the best. In the project, we use opennmt function onmt_average_models to handle this. You should sepcify the models(ckp) that you want to average and the output path. Besides, it has another arg called `-fp32` that can cast params to float32 otherwise fp16.

```
onmt_average_models -models MODEL -output OUTPUT
```

Detail:

```
for i, model_file in enumerate(model_files):
        m = torch.load(model_file, map_location='cpu')
        model_weights = m['model']
        generator_weights = m['generator']

        if fp32:
            for k, v in model_weights.items():
                model_weights[k] = v.float()
            for k, v in generator_weights.items():
                generator_weights[k] = v.float()

        if i == 0:
            vocab, opt = m['vocab'], m['opt']
            avg_model = model_weights
            avg_generator = generator_weights
        else:
            for (k, v) in avg_model.items():
                avg_model[k].mul_(i).add_(model_weights[k]).div_(i + 1)

            for (k, v) in avg_generator.items():
                avg_generator[k].mul_(i).add_(generator_weights[k]).div_(i
+ 1)

    final = {"vocab": vocab, "opt": opt, "optim": None,
             "generator": avg_generator, "model": avg_model}
```

Before evaluation, an optional operation is convert the averaged model to **C++**, which can speed up the inferance. And opennmt already provides the converters function.

```
```

```
cd nmt_trans/train
python convert_c_trans.py ../conf/en_zh_config.json
```

```
from ctranslate2.converters import opennmt_py
from ctranslate2.specs import transformer_spec
from nmt_trans.utils import file_helper, conf_parser

###
def run(self):
    model_spec = transformer_spec.TransformerSpec(self.num_layers,
self.num_heads)
    converter = opennmt_py.OpenNMTPyConverter(self.model_path)
    out_path = file_helper.get_abs_path(self.conf.pred_info.c_model_path)
    file_helper.mk_folder_for_file(out_path)
    converter.convert(out_path, model_spec, quantization=self.quant,
force=True)
###
```

Then, a C++ model is generated at the "out_path".

Now let's do evaluation using averaged checkpoint, also, you should input the config file and the val_data path:

```
"model_path": "data/offline_data/model_bin/nmt_en_zh/cht",
"eval_info": {
   "model_suffix": "_avg.pt",
   "quantization": "float16",
   "replace_unk": null,
   "verbose": null,
   "output": "data/offline_data/val_pred_large.cht"
```

check eval_en2zh.py and run it

```
#sh eval.sh
python eval_en2zh.py ../conf/en_zh_config.json PATH_TO_VAL_DATA
```

The main function will invoke ../predictor.py

Again, processing the eval data:

- split sentences according to rules(e.g. punctuations)
- check en-zh word-map [opetional]
- tagging
- bpe encoding

```
# translate using C++ model
translator = ctranslate2.Translator(model_path,device="auto")
res_list = translator.translate_batch(input_list, beam_size=4,
num_hypotheses=1, max_decoding_length=200)
```

post-processing:

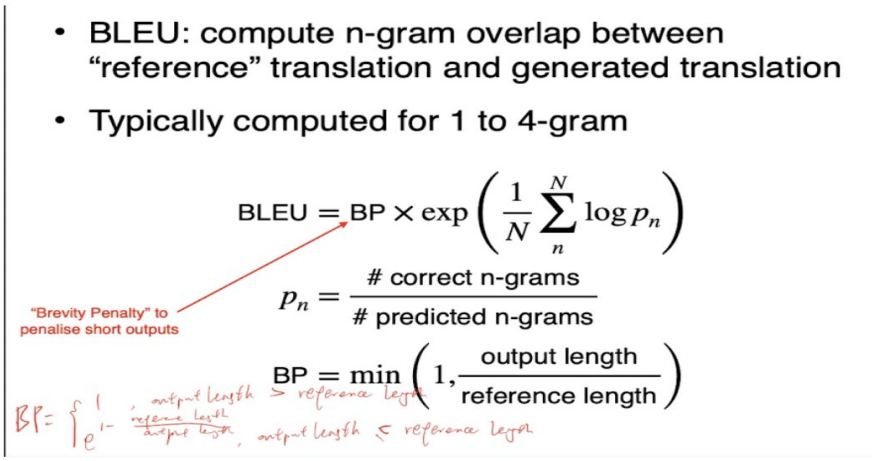
- remove repeated words
- bpe decoding
- merge sentences

Metrics

The metrics considered here are PPL and BLEU, which are normal choices for NMT task. I just paste the definition below, and you may click the links for detail.

$$PPL(W) = P(w_1w_2...w_n)^{-\frac{1}{N}}$$

▶ BLEU(Bilingual evaluation understudy)



training ppl:

2.6-2.9

eval bleu:

```
# base
BLEU = 45.84 72.4/53.4/41.2/32.8 (BP = 0.959 ratio = 0.960 hyp_len =
19651849 ref_len = 20470747)
# large
BLEU = 47.85 74.2/55.1/42.4/33.7 (BP = 0.974 ratio = 0.974 hyp_len =
19936887 ref_len = 20470747)
```

Deployment

Onnx

pass

TensorRT

pass

...