ディープラーニングのフレームワークを使った自然言語処理研究の実装の知見

寺西 裕紀
奈良先端科学技術大学院大学　自然言語処理学研究室

2017年9月1日(金)
第15回SVM勉強会

目次

- 自己紹介
- 機械学習は難しい
- 実装の工夫
  - Logging
  - Application
  - Preprocessing
  - Dataset
  - Loader
  - Model
  - Training
- 実例
- おまけ

- 内容
  - 個人的なノウハウ・提案・チュートリアル的な内容
  - DL+NLPをやり始めたときに知っていたかったこと
- 資料：https://github.com/chantera/svm2017

自己紹介

- Background
  - 大学(商学部) → 社会人(Web/アプリエンジニア)
    → NAIST松本研 (M2)
  - 機械学習・深層学習 / NLP 歴：ちょうど1年ぐらい
- 研究内容
  - 並列句解析: https://www.slideshare.net/HirokiTeranishi1/ss-78087806
- 会社
  - TeraBytes inc. https://terabytes.jp/
    - 事業内容: 受託開発, コンサルティング
- 趣味
  - ボクシング

機械学習は難しい

- 学習しない！→ 原因の切り分けが難しい
  - データが悪い
    - データの量が悪い・質が悪い
    - 前処理に問題がある
  - モデルが悪い
    - バグがある（ある程度動いてしまう）
    - モデルそのものが悪い（表現力 ↔ 汎化性能）
  - 学習方法が悪い
    - 最適化手法の選択が悪い, 学習率などの設定が悪い
  - 参考: https://www.slideshare.net/takahirokubo7792/ss-65413290
- 対応方法：Goodfellow本 (Chapter 11) が詳しい
  - URL：http://www.deeplearningbook.org/contents/guidelines.html

実装で回避

- 前提・目的：adhocな実装を避ける
  - adhocな実装はバグを生みやすい
    → モデル実装以外のバグで悩みたくない, バグを減らしたい
  - adhocな実装は学習がうまくいかない原因を探るのが大変
    → 原因：設定(データ, モデル, 学習方法), 実装(前処理, モデル実装, その他)
    - 研究的には設定の部分に時間を注ぎ込みたい
- 仕組み化・ライブラリ化
  - ディープラーニング / NLPに使える汎用的な仕組みを作る
  - 再利用なコードにまとめる
    - デバッグ用の処理を入れる, テストを書く

  → バグの減少, 問題の特定が容易, 開発の効率化
- 設計の見直し ← 今日の話のメイン

Example: BiLSTM POS tagger

```python
#/usr/bin/env python
# -*- coding: utf-8 -*-

from argparse import ArgumentParser
from collections import defaultdict
import logging

import chainer
from chainer import functions as F, links as L
import numpy as np


class Model(chainer.Chain):
    ...


def read_conll(file):
    ...


def main(train_file,
         test_file,
         n_epoch=20,
         batch_size=10,
         lr=0.01,
         gpu=-1):
    train_sentences = read_conll(train_file)
    n_train_samples = len(train_sentences)
    test_sentences = read_conll(test_file)
    n_test_samples = len(test_sentences)

    word2id = defaultdict(lambda: len(word2id))
    char2id = defaultdict(lambda: len(char2id))
    tag2id = defaultdict(lambda: len(tag2id))

    unknown = "<UNKNOWN>"
    unknown_word_id = word2id[unknown]
    unknown_char_id = char2id[unknown]

    train_words = []
    train_chars = []
    train_tags = []
    for tokens in train_sentences:
        words = []
        chars = []
        tags = []
        for token in tokens:
            words.append(word2id[token['form'].lower()])
            chars.append(np.array([char2id[char]
                                   for char in token['form']], np.int32))
            tags.append(tag2id[token['postag'].lower()])
        train_words.append(np.array(words, np.int32))
        train_chars.append(chars)
        train_tags.append(np.array(tags, np.int32))

    test_words = []
    test_chars = []
    test_tags = []
    for tokens in test_sentences:
        words = []
        chars = []
        tags = []
        for token in tokens:
            words.append(word2id.get(token['form'].lower(),
                                     unknown_word_id))
            chars.append(np.array([char2id.get(char, unknown_char_id)
                                   for char in token['form']], np.int32))
            tags.append(tag2id[token['postag'].lower()])
        test_words.append(np.array(words, np.int32))
        test_chars.append(chars)
        test_tags.append(np.array(tags, np.int32))

    model = Model(word_vocab_size=len(word2id),
                  word_embed_size=100,
                  char_vocab_size=len(char2id),
                  char_embed_size=30,
                  n_labels=len(tag2id))
    model(train_words, train_chars)
    if gpu > -1:
        chainer.cuda.get_device_from_id(gpu).use()
        model.to_gpu()

    optimizer = chainer.optimizers.Adam(lr)
    optimizer.setup(model)
    optimizer.add_hook(chainer.optimizer.GradientClipping(5.0))

    indices = np.arange(n_train_samples)
    for epoch in range(n_epoch):
        iteration, epoch_loss, epoch_accuracy = 0, 0.0, 0.0
        chainer.config.train = True
        chainer.config.enable_backprop = True
        np.random.shuffle(indices)
        for i in range(0, n_train_samples, batch_size):
            model.cleargrads()
            words = np.take(train_words, indices[i: i + batch_size])
            chars = np.take(train_chars, indices[i: i + batch_size])
            tags = np.take(train_tags, indices[i: i + batch_size])
            ys = model(words, chars)
            ys = F.concat(ys, axis=0)
            ts = F.concat(tags, axis=0)
            loss = F.softmax_cross_entropy(ys, ts, ignore_label=-1)
            loss.backward()
            optimizer.update()
            accuracy = F.accuracy(ys, ts, ignore_label=-1)
            epoch_loss += loss.data
            epoch_accuracy += accuracy.data
            iteration += 1
        logging.info('[train epoch {:02d}] loss: {:.6f}, accuracy: {:.6f}'
                     .format(epoch + 1,
                             epoch_loss / iteration,
                             epoch_accuracy / iteration))

        model.cleargrads()
        iteration, epoch_loss, epoch_accuracy = 0, 0.0, 0.0
        chainer.config.train = False
        chainer.config.enable_backprop = False
        for i in range(0, n_test_samples, batch_size):
            words = test_words[i: i + batch_size]
            chars = test_chars[i: i + batch_size]
            tags = test_tags[i: i + batch_size]
            ys = model(words, chars)
            ys = F.concat(ys, axis=0)
            ts = F.concat(tags, axis=0)
            loss = F.softmax_cross_entropy(ys, ts, ignore_label=-1)
            accuracy = F.accuracy(ys, ts, ignore_label=-1)
            epoch_loss += loss.data
            epoch_accuracy += accuracy.data
            iteration += 1
        logging.info('[test  epoch {:02d}] loss: {:.6f}, accuracy: {:.6f}'
                     .format(epoch + 1,
                             epoch_loss / iteration,
                             epoch_accuracy / iteration))


if __name__ == "__main__":
    logging.basicConfig(level=logging.DEBUG)
    chainer.config.debug = False
    chainer.config.type_check = False
    parser = ArgumentParser()
    parser.add_argument('--trainfile', type=str, required=True)
    parser.add_argument('--testfile', type=str)
    parser.add_argument('--epoch', type=int, default=20)
    parser.add_argument('--batchsize', type=int, default=10)
    parser.add_argument('--lr', type=float, default=0.01)
    parser.add_argument('--gpu', type=int, default=-1)
    args = parser.parse_args()
    main(args.trainfile, args.testfile, args.epoch, args.batchsize, args.lr)
```

Logging

- 原因特定のためには最も重要
- 機能の作り込み
  - ファイルと標準（エラー）出力への書き込み
    - 異なるエラーレベルを設定可能にする
  - ログファイルのユーティリティ
- フォーマットの見直し
  - 日付+タイムゾーン, プロセスごとの識別子, メッセージ
    - ミリ秒まで出す：正確な時間の把握, 簡易な時間計測
    - 識別子：同一ファイルに同時に書き込んだ場合のメッセージの識別
- 実験設定を残す
  - OS・実行環境, コマンドライン引数, モデルアーキテクチャ, ハイパーパラメータ

Logging

※Pythonの場合

- 標準ライブラリのロガーを拡張
- モデルのアーキテクチャは __repr__ を実装して表現
- コマンドライン引数やmain()への引数は薄いラッパーを使って呼び出し側でロギングすると楽
- main()はtry-exceptでラップして例外はログへ

```
DATE_FORMAT = "%Y-%m-%d %H:%M:%S.%f %Z"

class AppLogger(Logger):
    FORMAT = "%(asctime)-15s\t%(accessid)s\t[%(levelname)s]\t%(message)s"
    _config = {
        'level': INFO,
        'verbosity': TRACE,
        'filelog': True,
        'logdir': None,
        'filename': "%Y%m%d.log",
        'filemode': 'a',
        'fileprefix': '',
        'filesuffix': '',
        'fmt': FORMAT,
        'datefmt': DATE_FORMAT,
        'mkdir': False,
    }

    @classmethod
    def configure(cls, **kwargs):
        cls._config.update(kwargs)

    def initialize(self):
        super(AppLogger, self).initialize()
        config = AppLogger._config
        now = datetime.now(tzlocal())
        self._accessid = uuid.uuid4().hex[:6]
        self._uniqueid = "UNIQID"
        self._accesssec = now
        self._accesstime = now.strftime(config['datefmt'])

        if len(self.handlers) == 0:
            if config['filelog']:
                self._add_file_handler(config)

            stream_handler = logging.StreamHandler()
            stream_handler.setLevel(config['verbosity'])
            stream_handler.setFormatter(
                ColoredFormatter(config['fmt'], config['datefmt']))
            self.addHandler(stream_handler)

        message = "LOG Start with ACCESSID=[%s] UNIQUEID=[%s] ACCESSTIME=[%s]"
        self.info(message % (self._accessid, self._uniqueid, self._accesstime))
```

Logging

```
:hiroki $ python example/tagging-final.py --trainfile ~/Desktop/NLP/data/ptb-sd3.3.0/dep/wsj_train.conll --testfile ~/Desktop/NLP/data/ptb-sd3.3.0/dep/wsj_test.conll --epoch 50 --batchsize 32
2017-08-31 10:44:12.206 JST     2ecee2  [info]  LOG Start with ACCESSID=[2ecee2] UNIQUEID=[UNIQID] ACCESSTIME=[2017-08-31 10:44:12.206496 JST]
2017-08-31 10:44:12.207 JST     2ecee2  [trace] posix.uname_result(sysname='Darwin', nodename='Hirokis-MacBook-Air.local', release='15.6.0', version='Darwin Kernel Version 15.6.0: Sun Jun  4 21:43:07 PDT 2017; root:xnu-3248.70.3~1/RELEASE_X86_64', machine='x86_64')
2017-08-31 10:44:12.207 JST     2ecee2  [trace] sys.argv: ['example/tagging-final.py', '--trainfile', '/Users/hiroki/Desktop/NLP/data/ptb-sd3.3.0/dep/wsj_train.conll', '--testfile', '/Users/hiroki/Desktop/NLP/data/ptb-sd3.3.0/dep/wsj_test.conll', '--epoch', '50', '--batchsize', '32']
2017-08-31 10:44:12.207 JST     2ecee2  [trace] app._config: {'debug': False, 'logdir': '/Users/hiroki/Desktop/svm2017/example/logs', 'loglevel': 'info', 'logoption': 'a', 'quiet': False}
2017-08-31 10:44:12.207 JST     2ecee2  [info]  *** [START] ***
2017-08-31 10:44:12.207 JST     2ecee2  [debug] App._process(self) called - command: <function train at 0x10e5d48c8>, args: {'batch_size': 32, 'n_epoch': 50, 'gpu': -1, 'lr': 0.01, 'test_file': '/Users/hiroki/Desktop/NLP/data/ptb-sd3.3.0/dep/wsj_test.conll', 'train_file': '/Users/hiroki/Desktop/NLP/data/ptb-sd3.3.0/dep/wsj_train.conll'}
2017-08-31 10:44:12.512 JST     2ecee2  [info]  initialized model: {'dropout': 0.5, 'lstm_hidden_size': 200, 'n_blstm_layers': 1, 'char_filter_size': 30, 'char_window_size': 3, 'n_labels': 45, 'char_embed_size': 30, 'char_vocab_size': 78, 'word_embed_size': 100, 'word_vocab_size': 7305, 'self': {...}, '__class__': <class '__main__.Model'>, 'links': (<chainer.links.connection.embed_id.EmbedID object at 0x10fc44dd8>, {'dropout': 0.5, 'window_size': 3, 'out_size': 30, 'embed_size': 30, 'vocab_size': 78, 'self': {...}, '__class__': <class '__main__.CharCNN'>}, <chainer.links.connection.n_step_lstm.NStepBiLSTM object at 0x10fc44e10>, <chainer.links.connection.linear.Linear object at 0x10fca6a20>)}
2017-08-31 10:46:40.329 JST     2ecee2  [info]  [train epoch 01] loss: 1.399560, accuracy: 0.594988
2017-08-31 10:46:42.383 JST     2ecee2  [info]  [test  epoch 01] loss: 0.772760, accuracy: 0.778271
2017-08-31 10:48:15.499 JST     2ecee2  [info]  [train epoch 02] loss: 0.713188, accuracy: 0.784788
2017-08-31 10:48:17.590 JST     2ecee2  [info]  [test  epoch 02] loss: 0.490854, accuracy: 0.869005
2017-08-31 10:49:47.006 JST     2ecee2  [info]  [train epoch 03] loss: 0.515316, accuracy: 0.844857
2017-08-31 10:49:48.949 JST     2ecee2  [info]  [test  epoch 03] loss: 0.416239, accuracy: 0.889443
2017-08-31 10:51:22.413 JST     2ecee2  [info]  [train epoch 04] loss: 0.410470, accuracy: 0.876689
2017-08-31 10:51:24.432 JST     2ecee2  [info]  [test  epoch 04] loss: 0.386856, accuracy: 0.902937
^C2017-08-31 10:51:26.110 JST   2ecee2  [warn]  Signal(2) received: The program /Users/hiroki/Desktop/teras/teras/app/__init__.py will be closed
2017-08-31 10:51:26.206 JST     2ecee2  [info]  LOG End with ACCESSID=[2ecee2] UNIQUEID=[UNIQID] ACCESSTIME=[2017-08-31 10:44:12.206496 JST] PROCESSTIME=[433.997841000]

:hiroki $                                                    [~/Desktop/svm2017|2017-08-31 10:51:26]
```

- ログから実験の再現ができそう

Application

- 統合的な役割
  - コマンドライン引数の処理
  - 設定ファイルの読み書き
  - ロガーの初期化
  - 環境設定の切り替え
  - デバッグモードの切り替え
  - 命令の実行
  - 例外ハンドリング
  - シグナルハンドリング

```
class App(AppBase):
    DEFAULT_APP_NAME = "teras.app"
    DEFAULT_CONFIG_FILE = "~/.teras.conf"
    app_name = ''
    _argparser = ConfigArgParser(DEFAULT_CONFIG_FILE)
    verbose = True

    ...

    @classmethod
    def run(cls, command=None):
        cls.configure()
        command, command_args, common_args \
            = cls._parse_args(command)
        try:
            def handler(signum, frame):
                raise SystemExit("Signal(%d) received: "
                                 "The program %s will be closed"
                                 % (signum, __file__))
            signal.signal(signal.SIGINT, handler)
            signal.signal(signal.SIGTERM, handler)
            os.umask(0)

            self = cls._get_instance()
            self._initialize(command, command_args, common_args)
            self._preprocess()
            self._process()
            self._postprocess()
            self._finalize()
        except Exception:
            logging.e("Exception occurred during execution:",
                      exc_info=True, stack_info=cls.debug)
        except SystemExit as e:
            logging.w(e)
        finally:
            logging.getLogger().finalize()
            sys.exit(0)
```

Application

```
239
240 if __name__ == "__main__":
241     logging.AppLogger.configure(mkdir=True)
242     App.add_command('train', train, {
243         'train_file': arg('--trainfile', type=str, required=True),
244         'test_file': arg('--testfile', type=str),
245         'n_epoch': arg('--epoch', type=int, default=20),
246         'batch_size': arg('--batchsize', type=int, default=10),
247         'lr': arg('--lr', type=float, default=0.01),
248         'gpu': arg('--gpu', type=int, default=-1),
249     })
250     App.run()
```

- 実行したい関数をラップするだけ

Preprocessing

- 汎用的な前処理
  - 表層形の処理
    - lowercase化, 数字のトークン化
  - 単語のID化, 未知語処理
  - IDから単語のlookup
  - パディング
  - pretrained embeddingの読み込み
    - 2種類のファイル形式に対応
      - 辞書・ベクトル同一ファイル
      - 辞書・ベクトル別ファイル
    - ヘッダー有無：(単語数, 次元数)
  - データの正規化

```
318
319 class EmbeddingPreprocessor(Preprocessor):
320
321     def __init__(self,
322                  embed_file=None,
323                  embed_size=50,
324                  unknown="<UNK>",
325                  pad=None,
326                  tokenizer=split,
327                  initializer=standard_normal,
328                  preprocess=lower,
329                  embed_dtype=np.float32):
330         self.embed_dtype = embed_dtype
331         self._init_embeddings(embed_file, embed_size)
332         super(EmbeddingPreprocessor, self).__init__(
333             unknown, pad, tokenizer, preprocess,
334             _get_int_type(embed_dtype))
335         self._initializer = initializer
336         if not self.use_pretrained and self._pad_id >= 0:
337             self.get_embeddings()[self._pad_id] = 0
338
339     def _init_embeddings(self, embed_file, embed_size):
340         if embed_file is not None:
341             vocab_file = None
342             if isinstance(embed_file, (list, tuple)):
343                 embed_file, vocab_file = embed_file
344             vocabulary, embeddings = \
345                 load_embeddings(embed_file,
346                                 vocab_file,
347                                 self.embed_dtype)
348             self.use_pretrained = True
349         elif embed_size is not None:
350             ...
351
```

Dataset

- 汎用的なミニバッチの処理
  - samplesのバッチイテレーション
  - samplesのシャッフル
  - sampleの属性ごとのバッチ化
    - [(word, pos, label), ...]
      で初期化したデータを
      (words, chars, labels)に変換
  - なるべく余計な処理はさせない

```
class Dataset(Sequence):
    """Immutable Class"""

    def __init__(self, *samples):
        self._n_cols = len(samples)
        if self._n_cols == 0:
            self._columns = ([],)
            self._n_cols = 1
        elif self._n_cols == 1:
            if isinstance(samples[0], (list, tuple)):
                columns = tuple(
                    np.array(column)
                    if isinstance(column[0], np.ndarray) else column
                    for column in map(tuple, zip(*samples[0])))
                self._columns = columns
                self._n_cols = len(columns)
            else:
                self._columns = (samples[0],)
        elif self._n_cols > 1:
            self._columns = samples
        self._samples = list(map(tuple, zip(*self._columns)))
        self._len = len(self._samples)
        self._indices = np.arange(self._len)

    def __iter__(self):
        return (sample for sample in self._samples)

    def batch(self, size, shuffle=False, colwise=True):
        if shuffle:
            np.random.shuffle(self._indices)
        if colwise:
            return self._get_col_iterator(size)
        else:
            return self._get_row_iterator(size)
```

Loader

- Loader
  = FileReader
  + Preprocessing
  + Dataset

- 実装・機能
  - readerからitemsを
    イテレート
  - filterとmapを継承して
    itemを変換
    (preprocessorを呼ぶ)

```
12
13 class CorpusLoader(Loader):
14
15     @abstractmethod
16     def map(self, item):
17         words, labels = zip(*[(token['form'],
18                                self.tag_map.add(token['postag']))
19                               for token in item])
20         sample = (self._word_transform(words, as_one=True),
21                   self.get_processor('char')
22                   .fit_transform(words, as_one=False),
23                   np.array(labels, dtype=np.int32))
24         return sample
25
26     def filter(self, item):
27         return True
28
29     def load(self, file, train=False, size=None):
30         self._train = train
31         self._reader.set_file(file)
32         if size is None:
33             samples = [self.map(item) for item in self._reader
34                        if self.filter(item)]
35         else:
36             samples = []
37             for item in self._reader:
38                 if len(samples) >= size:
39                     break
40                 if not self.filter(item):
41                     continue
42                 samples.append(self.map(item))
43
44         return Dataset(samples)
```

Model

- よくある実装

```
55
56 class Model:
57
58     def forward(xs, ts):
59         ...
60         ys = self.linear(hs)
61         loss = F.softmax_cross_entropy(ys, ts)
62         return ys, loss
63
64
65 ...
66 ys, loss = model.forward(xs, ts)
67 accuracy = F.accuracy(ys, ts)
68 loss.backward()
69
```

Model

- モデルのforward計算でlossを計算させない
  - predict時にはlossの計算は必要ない
  - 本来predict時には正解データは必要ない
    → forwardメソッドの引数に正解データを入れたくない
- モデルと学習の分離
  - Modelの外でloss関数を設定
  - loss関数は タスク依存 > モデル依存
    - 例) 多クラス分類ならモデルはラベルのスコア(logit)を返せばよい
      → 呼び出し側でクロスエントロピーなりヒンジなりを選ぶ
- モデルに学習/予測の処理をなるべく意識させない
  - せいぜいdropoutぐらい

Model

- よくある実装

```
1
2  class MLP(chainer.Chain):
3
4      def __init__(self, n_units, n_out, dropout=0.5):
5          super(MLP, self).__init__()
6          with self.init_scope():
7              self.l1 = L.Linear(None, n_units)
8              self.l2 = L.Linear(None, n_units)
9              self.l3 = L.Linear(None, n_out)
10         self.dropout_ratio = dropout
11
12     def __call__(self, x):
13         h1 = F.dropout(F.relu(self.l1(x)), self.dropout_ratio)
14         h2 = F.dropout(F.relu(self.l2(h1)), self.dropout_ratio)
15         return self.l3(h2)
16
```

  - 先行研究と同じモデルなのに再現できない、なぜ？

Model

- 重みの初期化を明示的に指定しているか？
  - 使っているクラスの重みの初期化を正確に理解しているか
    - normal, uniform, zeros, lecun, glorot (xavier), he
  - デフォルトの初期化が適さないケースもある
    - activation関数によって適した初期化が違う，LSTMのforget gateのbiasを1にするかどうか
- 関数の実装は把握しているか？
  - dropoutのpredict時のscalingは？, LSTMのpeepholeは？
- フレームワークのソースコードは必ず読む
  - 先行研究を再現する場合は公開されているコードで使っているフレームワークと自分が使うフレームワークの両方
- 研究の場合はなるべく Wx + b レベルで自分で書く

Training

- 毎回書いていないか？
  - epoch / batch のイテレーション
  - trainフラグの管理
  - backwardの呼び出し, 重みの更新
  - loss, accuracyの計測/出力

```
    indices = np.arange(n_train_samples)
    for epoch in range(n_epoch):
        iteration, epoch_loss, epoch_accuracy = 0, 0.0, 0.0
        chainer.config.train = True
        chainer.config.enable_backprop = True
        np.random.shuffle(indices)
        for i in range(0, n_train_samples, batch_size):
            model.cleargrads()
            words = np.take(train_words, indices[i: i + batch_size])
            chars = np.take(train_chars, indices[i: i + batch_size])
            tags = np.take(train_tags, indices[i: i + batch_size])
            ys = model(words, chars)
            ys = F.concat(ys, axis=0)
            ts = F.concat(tags, axis=0)
            if gpu >= 0:
                ts.to_gpu()
            loss = F.softmax_cross_entropy(ys, ts, ignore_label=-1)
            loss.backward()
            optimizer.update()
            accuracy = F.accuracy(ys, ts, ignore_label=-1)
            epoch_loss += loss.data
            epoch_accuracy += accuracy.data
            iteration += 1
        logging.info('[train epoch {:02d}] loss: {:.6f}, accuracy: {:.6f}'
                     .format(epoch + 1,
                             epoch_loss / iteration,
                             epoch_accuracy / iteration))
```

Training

- 設計方針：モデルと学習の分離
  - Trainerクラスにはforward関数, loss関数, accuracy関数(必要であれば)を渡す
    - trainerは学習時にforward → loss (& accuracy) → backward & update, テスト時はforward → accuracy を呼び出す
- タスク・実験設定に固有な処理をループに入れたい
  - 例) テスト時のdecoding + evaluation
  - Trainerクラスをオブザーバーパターンで書く (Keras参考)
    - ループ内でイベント処理：self.notify(BATCH_END, data)
- forward, backward以外の処理が重いと話にならない
  - ループ部分はC, C++, Cythonで書く

Refactoring: Model

```
class Model(chainer.Chain):

    def __init__(self,
                 word_embedding,
                 char_embedding,
                 n_labels,
                 char_window_size=3,
                 char_filter_size=30,
                 n_blstm_layers=1,
                 lstm_hidden_size=200,
                 dropout=0.5):
        """
        @TODO: logging parameters initialization
        """
        super().__init__()
        if isinstance(word_embedding, np.ndarray):
            word_vocab_size, word_embed_size = word_embedding.shape
        else:
            word_vocab_size, word_embed_size = word_embedding
            word_embedding = None
        self.dropout = dropout
        with self.init_scope():
            self.word_embed = L.EmbedID(word_vocab_size, word_embed_size,
                                        word_embedding)
            self.char_cnn = CharCNN(char_embedding,
                                    char_filter_size, char_window_size,
                                    dropout)
            # @TODO: should specify parameters initialization
            self.blstm = L.NStepBiLSTM(n_blstm_layers,
                                       word_embed_size + char_filter_size,
                                       lstm_hidden_size, dropout)
            self.linear = L.Linear(lstm_hidden_size * 2, n_labels,
                                   initialW=initializers.GlorotUniform(),
                                   initial_bias=0)
        self._desc = {}  # model description
```

```
    def __call__(self, word_seqs, char_seqs):
        indices = [0]
        xs = []
        for word_seq, char_seq in zip(word_seqs, char_seqs):
            x_word = self.word_embed(self.xp.array(word_seq))
            x_char = self.char_cnn(char_seq)
            x = F.concat((x_word, x_char))
            indices.append(indices[-1] + x.shape[0])
            xs.append(F.dropout(x, self.dropout))
        hy, cy, ys = self.blstm(hx=None, cx=None, xs=xs)
        ys = F.concat(ys, axis=0)
        ys = self.linear(F.dropout(ys, self.dropout))
        ys = F.split_axis(ys, indices[1:-1], axis=0)
        return ys

    def __repr__(self):
        return repr(self._desc)
```

Refactoring: Loader

```
135
136 class DataLoader(dataset.loader.CorpusLoader):
137
138     def __init__(self,
139                  word_embed_size=100,
140                  char_embed_size=30,
141                  word_embed_file=None,
142                  word_preprocess=lambda x: x.lower(),
143                  word_unknown="<UNKNOWN>",
144                  embed_dtype='float32'):
145         super(DataLoader, self).__init__(reader=io.reader.ConllReader())
146         self.use_pretrained = word_embed_file is not None
147         self.add_processor(
148             'word', embed_file=word_embed_file, embed_size=word_embed_size,
149             embed_dtype=embed_dtype,
150             initializer=Uniform(scale=np.sqrt(3 / word_embed_size)),
151             preprocess=word_preprocess, unknown=word_unknown)
152         self.add_processor(
153             'char', embed_file=None, embed_size=char_embed_size,
154             embed_dtype=embed_dtype,
155             initializer=Uniform(scale=np.sqrt(3 / char_embed_size)),
156             preprocess=lambda x: x.lower())
157         self.tag_map = preprocessing.text.Vocab()
158
159     def map(self, item):
160         # item -> (words, chars, labels)
161         words, labels = zip(*[(token['form'],
162                                self.tag_map.add(token['postag']))
163                               for token in item[1:]])  # skip root token
164         sample = (self._word_transform(words, as_one=True),
165                   self.get_processor('char')
166                   .fit_transform(words, as_one=False),
167                   np.array(labels, dtype=np.int32))
168         return sample
169
170     def load(self, file, train=False, size=None):
171         if train and not self.use_pretrained:
172             # assign an index if the given word is not in vocabulary
173             self._word_transform = self.get_processor('word').fit_transform
174         else:
175             # return the unknown word index if the word is not in vocabulary
176             self._word_transform = self.get_processor('word').transform
177         return super(DataLoader, self).load(file, train, size)
178
```

Refactoring: Training

```
179
180 def train(
181         train_file,
182         test_file,
183         word_embed_file=None,
184         word_embed_size=100,
185         char_embed_size=30,
186         n_epoch=20,
187         batch_size=10,
188         lr=0.01):
189     logging.info('initialize DataLoader with word_embed_file={}, '
190                  'word_embed_size={}, and char_embed_size={}'
191                  .format(word_embed_file, word_embed_size, char_embed_size))
192     loader = DataLoader(word_embed_size, char_embed_size, word_embed_file)
193     logging.info('load train dataset from {}'.format(train_file))
194     train_dataset = loader.load(train_file, train=True)
195     logging.info('load test dataset from {}'.format(test_file))
196     test_dataset = loader.load(test_file, train=False)
197
198     model = Model(word_embedding=loader.get_embeddings('word'),
199                   char_embedding=loader.get_embeddings('char'),
200                   n_labels=len(loader.tag_map))
201     logging.info('initialized model: {}'.format(model))
202
203     optimizer = chainer.optimizers.Adam(lr)
204     optimizer.setup(model)
205     optimizer.add_hook(chainer.optimizer.GradientClipping(5.0))
206
207     trainer = training.Trainer(optimizer, model,
208                                loss_func=lambda ys, ts:
209                                F.softmax_cross_entropy(
210                                    F.concat(ys, axis=0), F.concat(ts, axis=0)),
211                                accuracy_func=lambda ys, ts:
212                                F.accuracy(
213                                    F.concat(ys, axis=0), F.concat(ts, axis=0)))
214     trainer.configure(framework_utils.config)
215
216     trainer.fit(train_dataset, None, batch_size=batch_size,
217                 epochs=n_epoch, validation_data=test_dataset,
218                 verbose=App.verbose)
219
```

Example: BiaffineParser

- BiaffineParser (Dozat+, 2016)
  - Chainer / PyTorch実装

    chantera/biaffineparser
  - コマンドライン引数でbackendを‘chainer’か‘pytorch’を指定
- 提案した設計の利点
  - フレームワークに依存しない

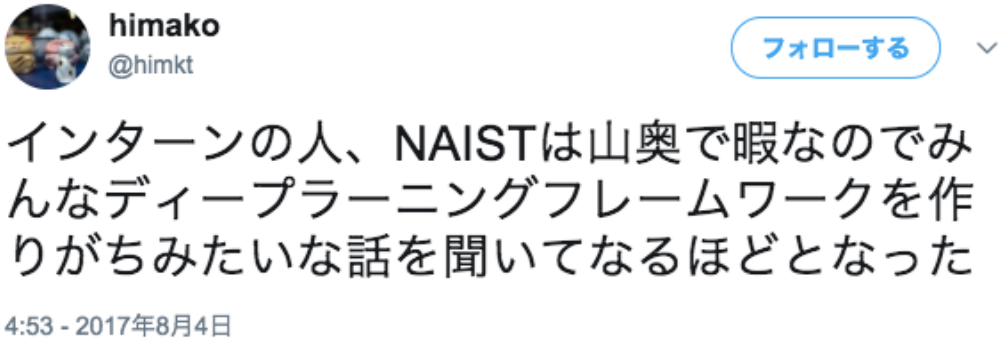
  → Modelクラスだけフレームワークで実装してtrainerにmodelとoptimizerと口ス関数を渡すだけ

```
63
64      model = model_cls(
65          embeddings=(loader.get_embeddings('word'),
66                      loader.get_embeddings('pos')),
67          n_labels=len(loader.label_map),
68          **model_params,
69      )
70      if gpu >= 0:
71          framework_utils.set_model_to_device(model, device_id=gpu)
72
73      if backend == 'chainer':
74          optimizer = chainer.optimizers.Adam(
75              alpha=lr, beta1=0.9, beta2=0.9, eps=1e-08)
76          optimizer.setup(model)
77          optimizer.add_hook(chainer.optimizer.GradientClipping(5.0))
78      elif backend == 'pytorch':
79          optimizer = torch.optim.Adam(model.parameters(),
80                                       lr=lr, betas=(0.9, 0.9), eps=1e-08)
81          torch.nn.utils.clip_grad_norm(model.parameters(), max_norm=5.0)
82
83      Log.i('optimizer: Adam(alpha={}, beta1=0.9, '
84            'beta2=0.9, eps=1e-08), grad_clip=5.0'.format(lr))
85
86      parser = models.BiaffineParser(model)
87
88      trainer = Trainer(optimizer, parser, loss_func=parser.compute_loss,
89                        accuracy_func=parser.compute_accuracy)
90      trainer.configure(framework_utils.config)
91      if backend == 'pytorch':
92          trainer.add_hook(Event.EPOCH_TRAIN_BEGIN, lambda data: model.train())
93          trainer.add_hook(Event.EPOCH_VALIDATE_BEGIN, lambda data: model.eval())
94      if test_dataset:
95          trainer.attach_callback(
96              utils.Evaluator(parser,
97                              pos_map=loader.get_processor('pos').vocabulary,
98                              ignore_punct=True))
99
100     trainer.fit(train_dataset, None,
101                 batch_size=batch_size,
102                 epochs=n_epoch,
103                 validation_data=test_dataset,
104                 verbose=App.verbose)
105
```

実装力を高めるために...

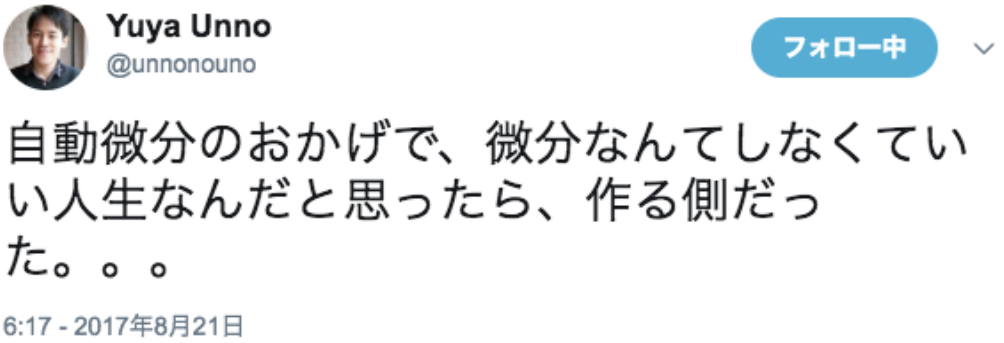
- ソースコードを読む
  - Chainer, Keras, DyNet
- きれいなコードを書く
  - コーディング規約を読む
    - Google C++ Style Guide
    - PEP8
  - linterを使ってコード解析/整形する：cpplint, flake8
- 勉強
  - リーダブルコード
  - デザインパターン
  - PofEAA
  - Effective C++ / Effective Java

おまけ



- 研究するなら結局はフレームワークのソースコードを読んで理解しなければならない → 仕様が変わるし面倒
- オレオレフレームワークを作ると幸せになれる
- フレームワーク実装の難しさの8割はtensor計算
  - tensor計算をライブラリに任せるならすぐ作れる
  - CPU/GPUの演算が同じインタフェースでできればもっと楽
    - numpy/cupy, arrayfire, etc...

おまけ



- フレームワーク実装の面倒くささの8割は微分の実装
- 自分の研究で使うときに必要な関数だけ書けば楽
  - みんなに使ってもらうフレームワークは目指さない
- フレームワーク雑感
  - Chainer: プロトタイピング
  - DyNet: アルゴリズムをゴリゴリ書くときに使う
  - Tensorflow: プロダクション
  - PyTorch: 洗練されてないのでコードが汚くなる, 今後に期待

Appendix

- 発表資料：https://github.com/chantera/svm2017
- ライブラリ：https://github.com/chantera/teras
  - 自分の研究用なのでAPIはちょくちょく変わります, forkして使ってください
  - 開発計画：Cython実装, Transition Systemの汎用的なクラス(Stateなど)