```
# __author__ = 'Dongjie Fan'
# __email__ = 'df1676@nyu.edu'
```

(For English Words or Chinese Hanzi)

Outline:

- Word_Embedding
  - word level or Hanzi (Chinese char) level
  - char level (only for English)
- Bi-LSTM
- Conditional Random Filed (CRF)

---

1. Word Embedding

Input: English words (eg. 'Play')

Embedding vector: word2vec + char2vec

- word2vec:
  - pre-trained dictionary (1 x p_1)
  - **OR** initial with random values (1 x p_1)
- char2vec:
  - initial with random numbers for each possible char (n)
  - padding -> the same length for every word (m)
  - ConvNet (n x m -> 1 x p_2)
  - hidden_output_layer (for each word in char level) (1 x p_2)
- word2vec + char2vec
  - dim: 1 x p = 1 x ($p_1 + p_2$) (for every word)
  - end2end training - (*hyper-param:* freeze - update word2vec ?)

---

2. Bi-LSTM

Pre-Process:

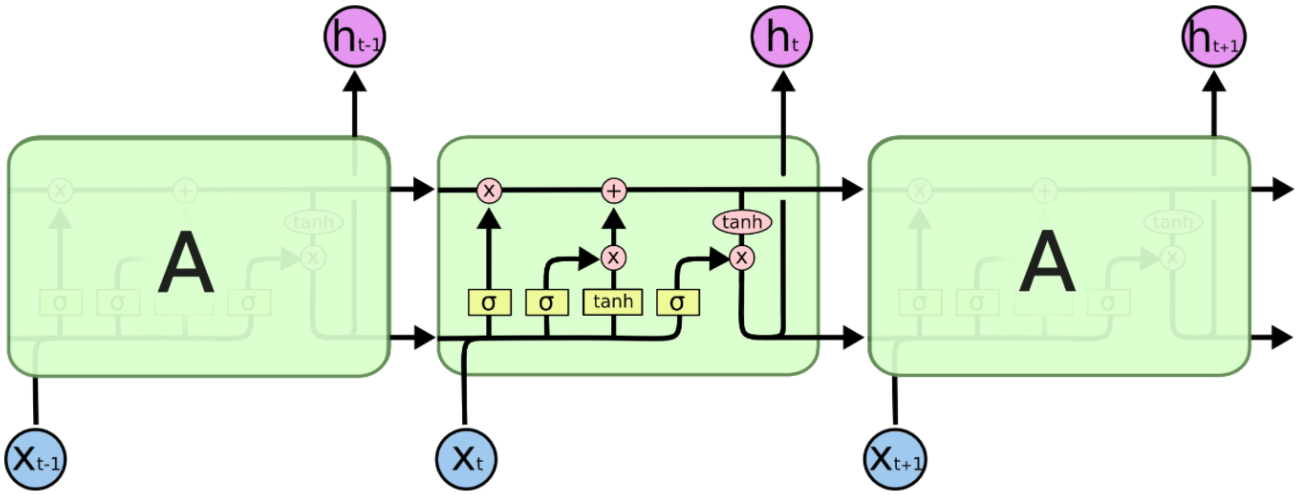
- Split into 'sentence' level (In our case, each medical record could be treated as a 'sentence')
- $s = len(sentence)$

Batch Input:

- Batch == 1 (current code): Shuffle all sentence and input them *one by one*
  - batch_size (1) x s x p
- Alternative solution:

- sort all sentences by length
- aggregate sentences with the same (or similar) length into one batch
- padding (try to avoid sparse input after padding)

LSTM:



The repeating module in an LSTM contains four interacting layers.

```
forget:  ft = σ(Wxf*xt + Whf*ht−1 + Wcf*ct−1 + bf )
input:   it = σ(Wxi*xt + Whi*ht−1 + Wci*ct−1 + bi)
direction: dt = tanh(Wxc*xt + Whc*ht−1 + bc)
update1: cell_state (memory): ct = ft*ct−1 + it*dt
output:  ot = σ(Wxo*xt + Who*ht−1 + Wco*ct + bo)
update2: hidden state (real output vector): ht = ot*tanh(ct)  --- (1 x h)
...
Next: FC layer
```

Bi-LSTM

- output dim: *batch_size* x s x $2h$
- (similar) task: sequential labeling (stack the hidden state (bi-lstm: 1 x $2h$) at each step together)

(For one sentence: s x $2h$)

FC:

- t = target_size -> How many classes?
- dim: s x $2h$ -> s x t (emission score) (For one sentence: s x $2h$)

---

3. CRF

- initial:
  - **transition matrix (M)**: $(t+2)$x$(t+2)$ -> { target size (t), [Start], [End] } -> P(ij) - $M_{i,j}$: from tag_i to tag_j
  - randomly initial values
    - and manually set $M_{[End],.}$ $M_{.,[Start]}$

    - **M** Back propagation <- update [end2end training]
- Score Function
    - (for one training sentence α, with a fixed known list of tags)

$$P(w_0, w_1, w_2, \ldots, w_k, w_{k+1}, [Start], T_1, T_2, \ldots, T_k, [End]) = \prod_{k=1}^{s} P(T_k|w_k) P(Tag_0 = [Start]) \prod_{i=1}^{s+1} P(T_i|T_{i-1})$$

        - $P(Tag_0 = [Start]) = 1$
        - Log Space: $Score_\alpha = logP(w_0, w_1, w_2, \ldots, w_k, w_{k+1}, [Start], T_1, T_2, \ldots, T_k, [End]) = \sum_{k=1}^{s} P(T_k|w_k) + \sum_{i=1}^{s+1} P(T_i|T_{i-1})$
        - The first items are from Bi-LSTM emission score; The second items are base on transition matrix.
    - For all possible series of tags: **Partition Probability** $Score_{Total}$
        - Forward Algorithm (similar in HMM, easier to be understood)
        - Backward Algorithm (similar in HMM) (see function in original code)
- Loss Function (based on Score Function)
    - (for one training sentence α, with a fixed known list of tags) $S = Score_{Total} - Score_\alpha$
        - when α = ground-truth list of tags, $Score_\alpha$ is the largest, thus S is the smallest.
    - Alternative Solution: Lablewise Loss (see function in original code)
    - End2End training
- Prediction Method
    - Vertibi Algotithm
- Evaluation Metric:
    - accuracy
    - precision
    - recall
    - FB1

---

4. Future Work & Ideas

1. Chinese
    1. Split characters into Chinese-word level. Then we have char&word level like English material and we may do prediction in Chinese-word level rather than current Chinese-char level.
    2. better and more specific word/char embedding methods for Chinese
    3. Like English-char, use CNN to do additional feature engineering from the raw data
2. Bi-LSTM
    1. We may use more than one input, like $input_{t-1}$, $input_t$ (t : current timestamp) to train and predict
3. CRF
    1. I think current CRF method is the simplest way to build seq2seq model. we only use linear chain CRF, not nested CRF like graph. And actually current version is quite like the pure HMM model. So More

complicated CRF models are worthwhile to be trained and tested.
2. Right now, the transition matrix only concerns the transition probability from previous tag t-1 to current tag t. How about initializing a much more complexed one including the probability from t-2, t-1 (or t-n, ... t-1) to t.

4. Run time
    1. quite slow now, because the batch_size ==1. improve this step using methods mentioned above.
5. Add Attention model ?

---

5. Code

```
.
├── CRF.py
├── data
│   ├── conll2000.test.txt
│   ├── conll2000.train.txt
│   ├── test.txt
│   └── train.txt
├── embedding
│   └── emb.txt
├── loader.py
├── LstmCrfModel.py
├── LstmModel.py
├── model
│   ├── dictionaries.dic
│   ├── model.mdl
│   └── parameters.json
├── README.md
├── tmp
│   ├── conlleval
│   ├── evaluate.txt
│   ├── score.txt
│   └── sen_score.txt
├── train.py
└── utils.py
```

```
1. Data
data/test.txt
data/train.txt

2. setting
model/parameters.json

3. pre-trained
model/dictionaries.dic: built dict for training/test set, like indexing and sort of
things
model/model.mdl: pre-trained model
embedding/emb.txt: Chinese-char word2vec embedding (pre-trained or stored after
training by the code owner)

4. training
- train.py (Pipeline)
        - CRF.py
                - [core] (copy from one of the official pytorch tutorials)
        - LstmCrfModel.py (LstmModel.py: no need anymore)
                - combine CRF and LSTM together
- other
        - loader.py
        - utils.py

5. result
tmp/*
```

6. Reference

1. fangwater/**Medical-named-entity-recognition-for-ccks2017**
    1. main.pdf
2. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF

    > State-of-the-art sequence labeling systems traditionally require large amounts of task-specific knowledge in the form of hand-crafted features and data pre-processing. In this paper, we introduce a novel neutral network architecture that benefits from both word- and character-level representations automatically, by using combination of bidirectional LSTM, CNN and CRF. Our system is truly end-to-end, requiring no feature engineering or data pre-processing, thus making it applicable to a wide range of sequence labeling tasks. We evaluate our system on two data sets for two sequence labeling tasks --- Penn Treebank WSJ corpus for part-of-speech (POS) tagging and CoNLL 2003 corpus for named entity recognition (NER). We obtain state-of-the-art performance on both the two data --- 97.55\% accuracy for POS tagging and 91.21\% F1 for NER

3. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method

```
The word2vec software of Tomas Mikolov and colleagues ([this https URL]
(https://code.google.com/p/word2vec/) ) has gained a lot of traction lately, and
provides state-of-the-art word embeddings. The learning models behind the software are
described in two research papers. We found the description of the models in these
papers to be somewhat cryptic and hard to follow. While the motivations and
presentation may be obvious to the neural-networks language-modeling crowd, we had to
struggle quite a bit to figure out the rationale behind the equations. This note is an
attempt to explain equation (4) (negative sampling) in "Distributed Representations of
Words and Phrases and their Compositionality" by Tomas Mikolov, Ilya Sutskever, Kai
Chen, Greg Corrado and Jeffrey Dean.
```

- in the papers above, the training data is in English, not Chinese. And `fangwater` did a great job to customize the method to load, transform and train Chinese material. And check **train.txt** or **test.txt** to understand the real input data format!!!