Project Report

We did two lines of works:

1) **Engineering Oriented**: wrote a minimal python version of JET, the core is the Tipster architecture

2) **Research Oriented**: try using Seq2Seq model to build a constituency parser

We briefly described the two lines of work respectively below.

The links to the repos of the two (clickable):

1) Jettie

2) Seq2Seq parser

---

1) Jettie

a light weight python information extraction framework, a descendant of NYU's JET IE system

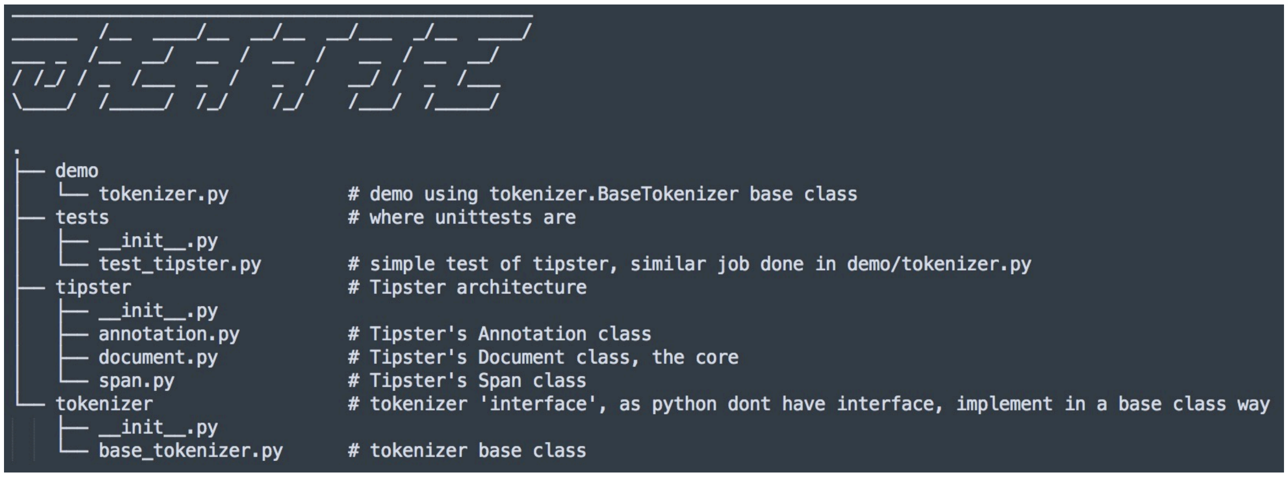
Motivation

Tipster architecture good for

- pipelining
- systematic experiment on IE tasks
- alignment of annotations which might facilitates future visualization work.

In this project, we consider a minimal architecture of Tipster.

Code Structure



```
_________________________________________________
______  /__  ____/__  __/__  __/___  _/__  ____/
___ _  /__  __/  __  /  __  /    __  / __  __/
/ /_/ / _  /___  _  /   _  /    __/ /  _  /___
\____/  /_____/  /_/    /_/     /___/  /_____/

.
├── demo
│   └── tokenizer.py          # demo using tokenizer.BaseTokenizer base class
├── tests                     # where unittests are
│   ├── __init__.py
│   └── test_tipster.py       # simple test of tipster, similar job done in demo/tokenizer.py
├── tipster                   # Tipster architecture
│   ├── __init__.py
│   ├── annotation.py         # Tipster's Annotation class
│   ├── document.py           # Tipster's Document class, the core
│   └── span.py               # Tipster's Span class
└── tokenizer                 # tokenizer 'interface', as python dont have interface, implement in a base class way
    ├── __init__.py
    └── base_tokenizer.py     # tokenizer base class
```

Basic Design

Primary architecture: Tipster

Tipster is implemented in the subpackage of `jettie`, which is the core of ensuring annotations are well aligned. While python is dynamically typed language, much effort can be saved from writing getter and setter methods.

NLP component interface

In order for NLP task components to comply with Tipster, a natural way to do is require the customized components to provide methods which annotate and align text in a Tipster way. While in Java, such requirements can be implemented through interface, but in python, we do it in a base class and ask the customized component to inherent from the base class, and implement the methods defined in the base class. To see an example, see the `BaseTokenizer` class defined in `./tokenizer/base_tokenizer.py`, and its inherent subclass `SimpleTokenizer` defined in `./demo/tokenizer.py`.

Tests

We performed unittest with automatic unit test tool `nose`, for an unittest case example please see `/tests/test_tipster.py`. Since we would still develop and use the project code in our own future work, writting automatic tests can make our life a little easier and neater.

One can run all the unittests at the `./jettie/` directorie by issueing `$ nosetests .`, and ideal output indicating test cases passed should look like `./doc/unittest.png`.

Distribution and install

We have distributed `jettie` with `pip`, to install it, run:

`$ pip install jettie`

to verify it installed successfully, open a python shell, do the following:

```
>>> import jettie
>>> jettie.woof()
I am a Shiba, and I woof! Woof!
>>>
```

---

2) A Seq2Seq Constituency Parser

Based on Google's work, Grammar as a Foreign Language. By the title of the paper, it carries the intuition that just linearize the constituency parse tree, we can treat parsing as task aims to learning a function:

$f : string \mapsto string$

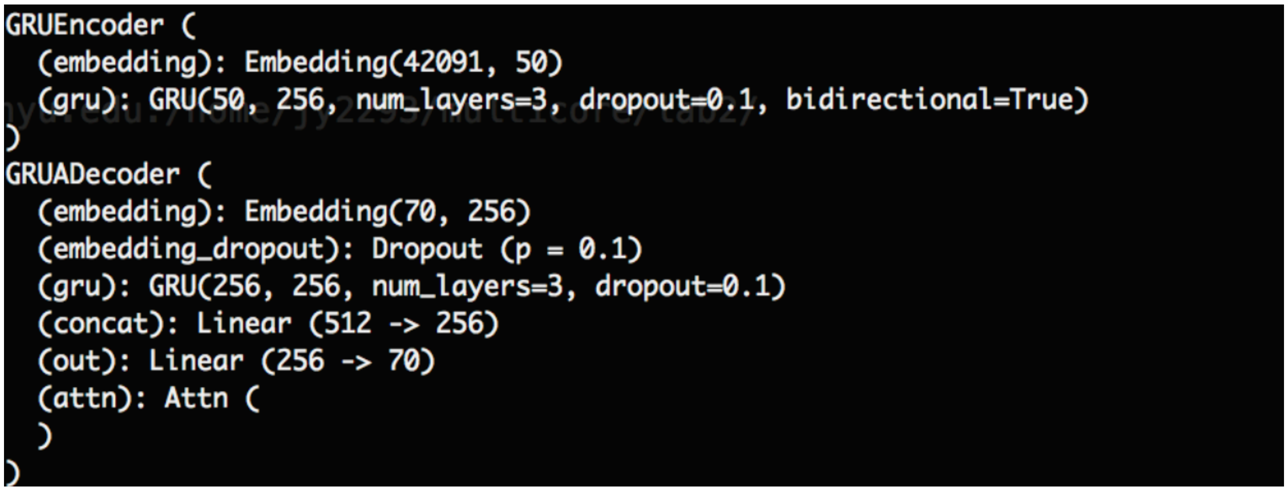
so that Seq2Seq method might work.

Dataset

We used a portion of OntoNote dataset with are 75187 sentences in total.

Model Architecture

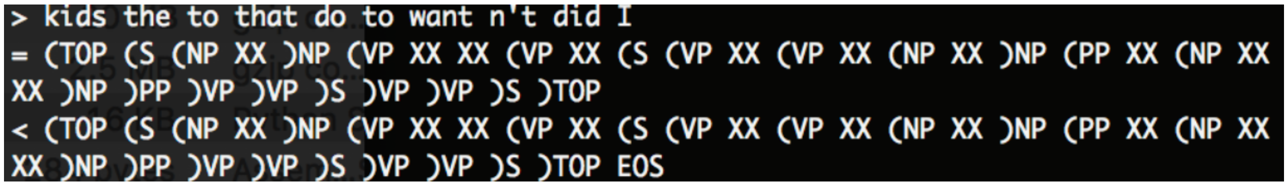
We trained a sequence-to-sequence model with attention mechanism, using 3 GRU layers as encoder and 3 GRU layers as decoder. We used a pre-trained 50 dimension embedding model for encoder and a random initialized embedding model for decoder. The model was trained using Pytorch framework with CUDA.



```
GRUEncoder (
  (embedding): Embedding(42091, 50)
  (gru): GRU(50, 256, num_layers=3, dropout=0.1, bidirectional=True)
)
GRUADecoder (
  (embedding): Embedding(70, 256)
  (embedding_dropout): Dropout (p = 0.1)
  (gru): GRU(256, 256, num_layers=3, dropout=0.1)
  (concat): Linear (512 -> 256)
  (out): Linear (256 -> 70)
  (attn): Attn (
  )
)
```

Results

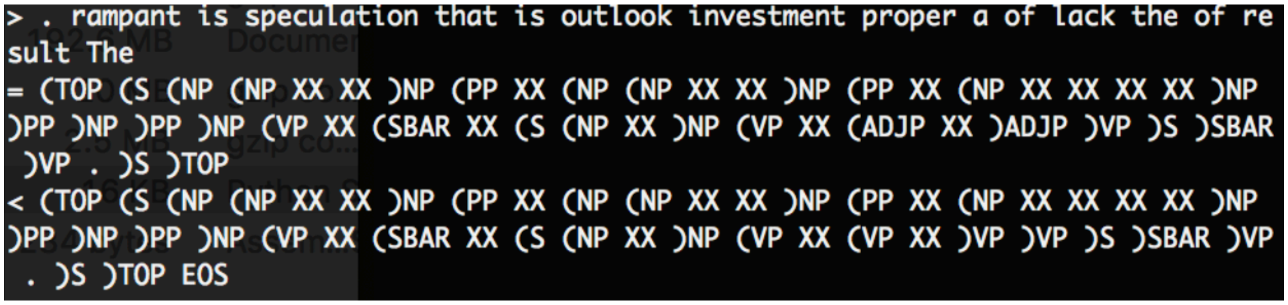
The model we trained showed acceptable performance on the short sentences. After half of an epoch of training, the result for short sentences are quite accurate.



```
> kids the to that do to want n't did I
= (TOP (S (NP XX )NP (VP XX XX (VP XX (S (VP XX (VP XX (NP XX )NP (PP XX (NP XX
XX )NP )PP )VP )VP )S )VP )VP )S )TOP
< (TOP (S (NP XX )NP (VP XX XX (VP XX (S (VP XX (VP XX (NP XX )NP (PP XX (NP XX
XX )NP )PP )VP )VP )S )VP )VP )S )TOP EOS
```

'>' followed by input token, '=' followed by true parser tree and '<' followed by the prediction from trained model, the 'EOS' is added manually not counted as a prediction

However on longer sentences things are not very satisfied up to the time we run 1.6 epoches:



```
> . rampant is speculation that is outlook investment proper a of lack the of re
sult The
= (TOP (S (NP (NP XX XX )NP (PP XX (NP (NP XX XX )NP (PP XX (NP XX XX XX XX )NP
)PP )NP )PP )NP (VP XX (SBAR XX (S (NP XX )NP (VP XX (ADJP XX )ADJP )VP )S )SBAR
 )VP . )S )TOP
< (TOP (S (NP (NP XX XX )NP (PP XX (NP (NP XX XX )NP (PP XX (NP XX XX XX XX )NP
)PP )NP )PP )NP (VP XX (SBAR XX (S (NP XX )NP (VP XX (VP XX )VP )VP )S )SBAR )VP
 . )S )TOP EOS
```

after 1.6 epoches of training, which was as far as we went, long sentences are getting better

Due to the racing conditions on CIMS GPU servers, we could not run our program very fast, but the trend observed was clear that as training phase goes longer, results should be better.