POS Tagging

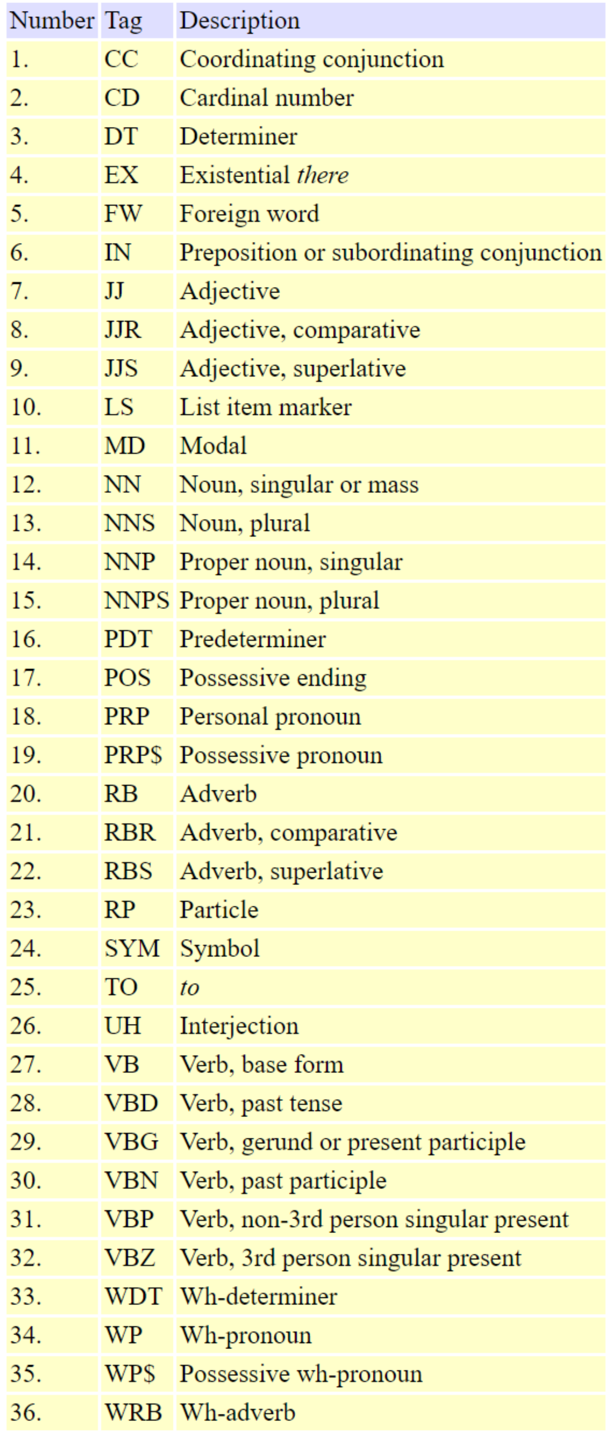
Dong Liu & Yunlin Peng

Introduction - Problem

- Definition:
  - *In corpus linguistics, part-of-speech tagging (POS tagging or POS tagging or POST), also called grammatical tagging is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc. – Wikipedia*
- Motivation
  - *POS tagging is the fundamental of many NLP tasks, for example, machine translation, QA, etc. We decided to realize the POS tagging in this project.*
- Proposal:
  - *Our project is to implement part-of-speech tagging task by using Bi-RNN Tagger, Bi-LSTM Tagger on TreeBank Tag dataset.*

Methodology-Dataset

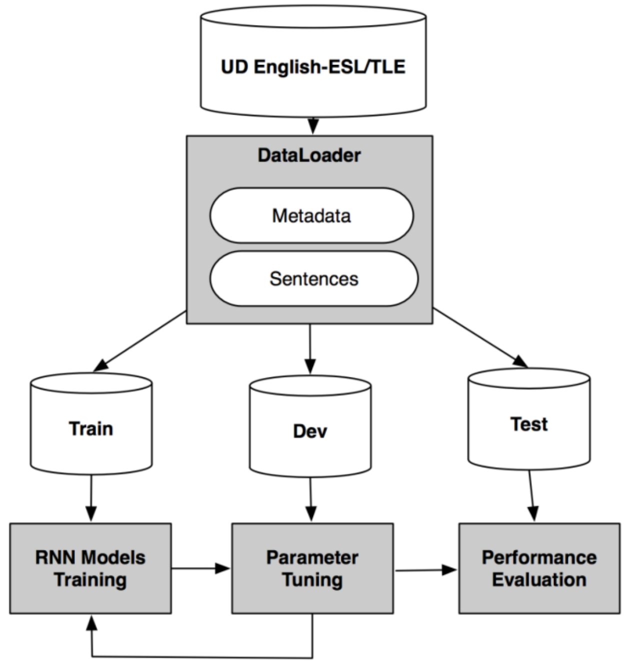
- *A treebank or parsed corpus is a text corpus in which each sentence has been parsed, i.e. annotated with syntactic structure. Syntactic structure is commonly represented as a tree structure, hence the name Treebank. The term Parsed Corpus is often used interchangeably with Treebank: with the emphasis on the primacy of sentences rather than trees.*
- *Treebanks are often created on top of a corpus that has already been annotated with part-of-speech tags. In turn, treebanks are sometimes enhanced with semantic or other linguistic information.*
- *Treebanks can be created completely manually, where linguists annotate each sentence with syntactic structure, or semi-automatically, where a parser assigns some syntactic structure which linguists then check and, if necessary, correct. In practice, fully checking and completing the parsing of natural language corpora is a labour intensive project that can take teams of graduate linguists several years. The level of annotation detail and the breadth of the linguistic sample determine the difficulty of the task and the length of time required to build a treebank.*
- *Some treebanks follow a specific linguistic theory in their syntactic annotation (e.g. the BulTreeBank follows HPSG) but most try to be less theory-specific. However, two main groups can be distinguished: treebanks that annotate phrase structure (for example the Penn Treebank or ICE-GB) and those that annotate dependency structure (for example the Prague Dependency Treebank or the Quranic Arabic Dependency Treebank).*

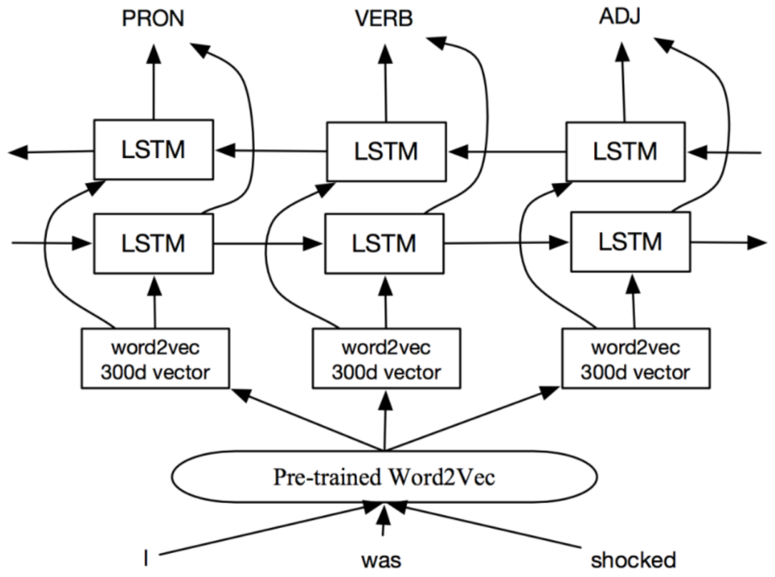


Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

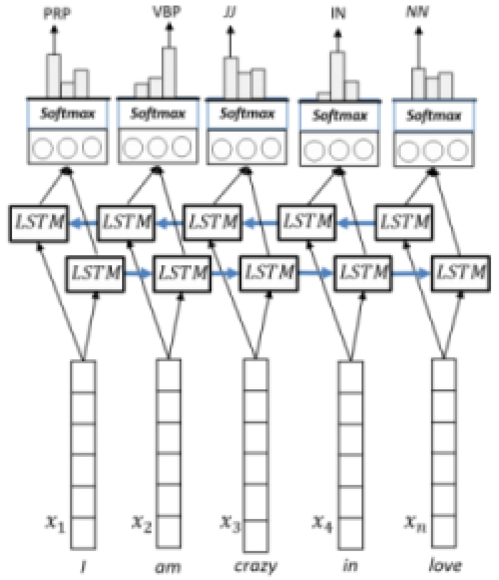
Methodology-Models

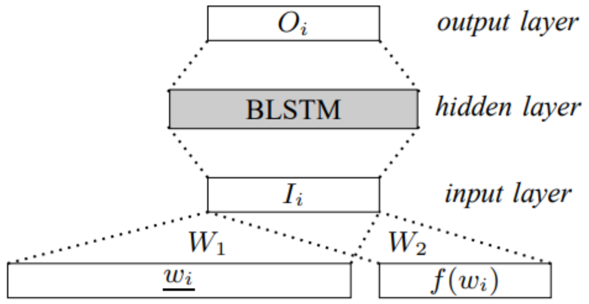
- *POS tagging could be the fundamentals of many NLP/NLU tasks, such as Name Entity Recognition (NER) and Abstract Meaning Representation (AMR). In this project. The following are the candidate models:*
    - *Bidirectional RNN(Bi-RNN)*
        - The principle of Bi-RNN is to split the neurons of a regular RNN into two directions, one for positive time direction (forward states), and another for negative time direction (backward states). Those two states' output are not connected to inputs of the opposite direction states. By using two directions, input information from the past and future of the current time frame can be used unlike standard RNN which requires the delays for including future information.
    - *Bidirectional LSTM (Bi-LSTM)*
        - Bi-LSTM is a variant of Bi-RNN in which the RNN cell is replaced by LSTM cell
- *We will apply the above models on continuous POS tagging task.*





Introduction-Details of Models





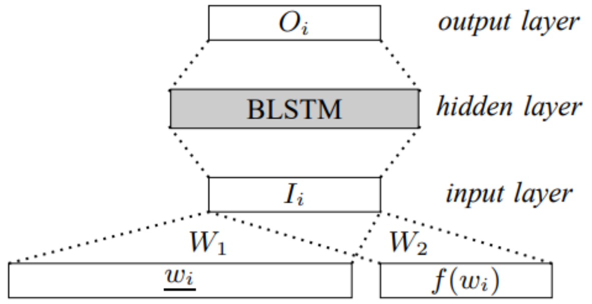
In this task, a POS tagger was trained with all train data (3131 sentences), and tested with test data (783 sentences), with each sentences padded to 217 length. The following is the architecture:

1. *Word embedding layer*
   1. *In this experiment we use the nn.Embedding api provided by torch.*
2. *Bi-LSTM Layer (as hidden state)*
   1. *Input: list of embedding*
   2. *Output: 2*hidden_size units processed by Bi-LSTM model.*
3. *Output layer*
   1. *Input: 2*hidden_size units processed by Bi-LSTM model*
   2. *output: list of probability with size of tag_size by using softmax activation. Take tag with the highest probability as the decision.*

Reference: [1510.06168.pdf (arxiv.org)](1510.06168.pdf)

Methodology-Code

- *Packages: PyTorch, nltk, numpy, sklearn*
- *The process of coding:*
- *1. obtained and pre-process the data*
    - *Downloaded the dataset form the Penn treebank*
    - *Get the input(list of words) and output(list of tags)*
    - *Vectorize input and output sequences for each sentence*
    - *Pad sequences*
    - *Transform the dataset into batches*
- *2. Build the model*
    - *Using the word embedding to encode each word in the batch*
    - *Using Bi-LSTM/Bi-RNN layer(s) to process the list of word embeddings.*
    - *Pass the Bi-LSTM/Bi-RNN layer(s) processing results to last layer to get the output.*
- *3. Evaluate the model on the test set*



Result

- Comparision between different Model

Model	Test Accuracy
Bi-LSTM (lr = 0.001, hidden_dim = 256, embedding_dim = 128)	88.40%
Bi-LSTM (lr = 0.005, hidden_dim = 256, embedding_dim = 128)	91.99%
Bi-LSTM (lr = 0.005, hidden_dim = 128, embedding_dim = 128)	92.14%
Bi-LSTM (lr = 0.005, hidden_dim = 128, embedding_dim = 256)	92.70%
Bi-RNN (lr = 0.001, hidden_dim = 256, , embedding_dim = 128)	88.72%
Bi-RNN (lr = 0.005, hidden_dim = 256, , embedding_dim = 128)	90.96%
Bi-RNN (lr = 0.005, hidden_dim = 128, , embedding_dim = 128)	90.85%
Bi-RNN (lr = 0.005, hidden_dim = 128, , embedding_dim = 256)	90.89%

Result Analysis

- **Why Bi-LSTM performs better than Bi-RNN?**
  - *One of the fundamental differences between an RNN and an LSTM is that an LSTM has an explicit memory unit which stores information relevant for learning some task. In the standard RNN, the only way the network remembers past information is through updating the hidden states over time, but it does not have an explicit memory to store information.*
  - *On the other hand, in LSTMs, the memory units retain pieces of information even when the sequences get really long.*
- **How learning rate influence the result?**
  - *The learning rate is a configurable hyperparameter used in the training of neural networks that has a small positive value.*
  - *The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs.*
  - *A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to get stuck to a small local area, in our experiment, for learning rate = 0.01, it maybe relatively small and finally stop at a local optimal.*

Result Analysis

- **How embedding dimension influence the result?**
  - *May be more embedding dimension will carry more semantics information which will result in a good result.*
  - *Reference:*
    - *https://ai.stackexchange.com/questions/28564/how-to-determine-the-embedding-size.*
- **How hidden dimension influence the result?**
  - *Have an impact on the number of trainable parameters.*
  - *Some possible rules:*
    - *The number of hidden neurons should be between the size of the input layer and the size of the output layer.*
    - *The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.*
    - *The number of hidden neurons should be less than twice the size of the input layer.*
  - *Reference:*
    - *https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw*

Overfitting Problem

- *Solutions:*
    - *1. batch normalization*
    - *2. dropout*
    - *3. early stopping*
    - *4. data augmentation*

Conclusion

- *Tasks: POS Tagging*
- *Model:*
  - *1. Bi-LSTM model*
  - *2. Bi-RNN model*
- *Control Variable Experiment:*
  - *1. learning rate*
  - *2. embedding size*
  - *3. hidden size*