Part of Speech tagging based on Deep Neural Networks

Qing Long Zheng
Qkz5068@psu.edu

Task

Part-of-speech (POS) tagging is a typical classification task, so task for this final project is to category words from English sentenses and Tweets content using CNN and RNN models.
Beside of that, to practice models and check their performance in real world case, at the end of this report I conducted predictions on ***in_domain unlabeled test set*** and ***out_of_domain unlabeled test set***, compare them to the ***in-domain labeled set*** from Professor Yin and ***out-of-domain labeled set*** from the one with highest score from our class.

Dataset

The training data in experiments is downing from [3]. This data consists of the same partitions of the ***Wall Street Journal*** corpus (WSJ) as the widely used data for noun phrase chunking: sections 15-18 as training data (211727 tokens) and section 20 as test data (47377 tokens). The annotation of the data has been derived from the WSJ corpus by a program written by Sabine Buchholz from Tilburg University, The Netherlands. The format is specified as one word per line, with a space separating the word and its label, and sentences separated by blank lines. Users can replace the training data in the '**Data/labeled train set**' directory.

In-domain unlabeled test set （***49388*** tokens） and ***out-of-domain unlabeled test set*** （***15971*** tokens） from midterm project and homework2. Users can replace the training data in the '**Data/unlabeled test set**' directory.

Preprocessing

Load dataset:

Since there is a column of data in the data set that we do not need, I used 'ConllCorpusReader' in the NLTK library to help us read only the 'WORD' and 'POS' parts and encapsulate the two parts in their own newly created arrays.

Prepare data:

The function involves saving ***tag*** and ***word tokenizers*** along with the models for future use. By initializing two instances of TensorFlow's Keras Tokenizer class: one for the words in the sentences, with an out-of-vocabulary token specified as '<OOV>', and another for the POS tags without an out-of-vocabulary token. No special OOV token is needed here since the set of tags is generally predefined and controlled. Then create an internal vocabulary based on the unique words and tags present in the provided sentences. After this conversion, both the word sequences ***(X)*** and the tag sequences ***(y)*** are padded or truncated to a fixed length of **200** using TensorFlow's 'pad_sequences' utility. This ensures that all input sequences have a uniform length for training models.

Load pre-trained vector:

Background and related work:

GloVe (Pennington et al., 2014) is a model based on global word-word co-occurrence statistics. We use one Glove's models. The first, which we name" Glove Twitter", trained on 2 billion tweets, contains 200-dimensional vectors for 1.2M words. The integration of these vectors is a key enhancement aimed at improving the accuracy and efficiency of POS tagging for text in tweets.

Dealing with Glove:

load_glove_model:

Function loads pre-trained embeddings from a ***GloVe*** file, returning a dictionary that maps words to their corresponding high-dimensional vectors. The function iterates over each line in the file. Each line is split into its components. The first component is assumed to be the word, and the remaining components are the elements of the word's embedding vector. The word is stored as a key in the dictionary.

create_embedding_matrix:

Subsequently, the ***create_embedding_matrix*** function constructs an embedding matrix compatible with a given word tokenizer we implemented in ***Prepare data***, which can be directly used in neural network layers. This matrix dimensions 'vocab_size=19462' x 'embedding_dim=200', where 200 is fixed dimensionality of the GloVe embeddings. Each row of this matrix will represent the embedding vector of a word in the vocabulary. The function returns the ***embedding_matrix***, which now contains the GloVe embeddings for the words present in the dataset's vocabulary.

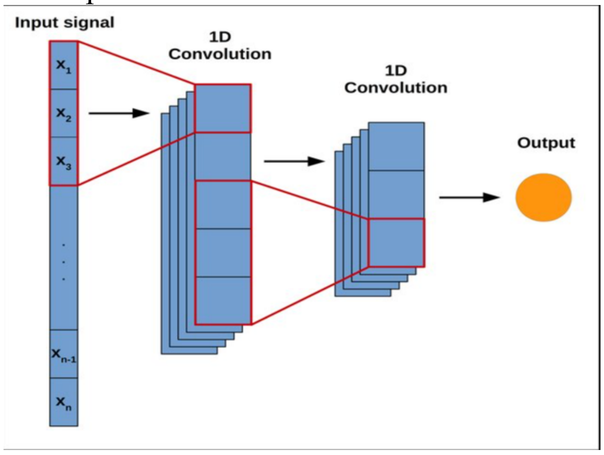
Architectures of two deep learning systems

CNN:

Background and related work:

More recently, one-dimensional CNNs have shown significant promise in dealing with structured language data in tasks such as machine translation and document classification. In 2018, Bai et al. [1] showed that, for many sequences modeling tasks, 1D CNNs using current best practices such as dilated convolutions often perform as well as or better than recurrent neural network architectures.

Figure 1. Simple 1D convolutional neural network architecture with two



convolutional layers

Each of the convolutional stages in Figure 1 shows a set of learnable convolutional filters followed by a pooling operation. These convolutional filters act to extract the high-level features (such as edges and curves in an image) from a supplied input by convolving a set of weights with the input and applying a non-linear activation function. The outputs of this are then fed into a pooling operation which reduces the spatial size of the features extracted by the convolutional filters whilst emphasizing the dominant features learned by each filter. As the input progresses through the convolutional stages (left to right on Figure 1), the network learns more problem-specific features.

Implement & Structure:

This network uses a Sequential model, combining an embedding layer, two convolutional layers, a dropout layer, and a Time Distributed Dense layer, then computes the sparse categorical cross entropy loss Adam optimizer is used with learning rate of. 0.001.

Formula for categorical crossentropy (S - samples, C - classes, $s \in c$ - sample belongs to class c) is:

$$-\frac{1}{N}\sum_{s\in S}\sum_{c\in C} 1_{s\in c} \log p(s \in c)$$

Layers:

Embedding Layer: Utilizes pre-trained word embeddings.

1D Convolutional Layers: The model uses ***64*** filters in this layer, with kernel sizes of ***5***. The ***'ReLU'*** activation function introduces non-linearity, helping the model learn complex patterns. The output size is the same as the input size, preserving spatial dimensions.

1D Convolutional Layers: The model uses ***64*** filters in this layer, with kernel sizes of ***3***. The ***'ReLU'*** activation function introduces non-linearity, helping the model learn complex patterns. The output size is the same as the input size, preserving spatial dimensions.

Dropout Layer: This layer randomly sets a fraction (***30***% in this case) of the input units to 0 at each update during training, which helps prevent overfitting. Overfitting is common in deep learning models, especially when the training data is limited, and dropout is an effective way to mitigate this.

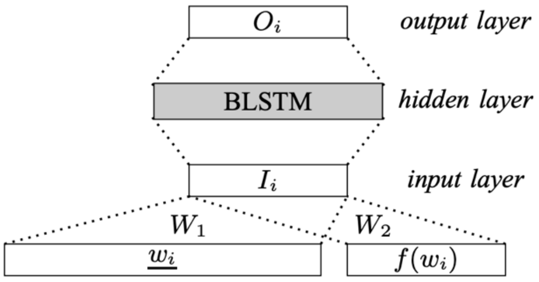
Time Distributed Dense Layer: This layer applies a dense (fully connected) layer to every temporal slice of the input (each word in the sequence). The number of units in the dense layer is set to the number of unique tags plus one, presumably to account for a padding tag or another category. The ***SoftMax*** activation function is used for ***multi-class classification***, outputting probabilities for each tag.

Architectures of two deep learning systems

RNN:

Background and related work:

Bidirectional long short-term memory (BLSTM) is a type of recurrent neural network (RNN) that can incorporate contextual information from long period of fore-and-aft inputs. It has been proven a powerful model for sequential labeling tasks. For applications in natural language processing, it has helped achieve superior performance in language modeling, language understanding, and machine translation. Since POS tagging is a typical sequential labeling task, it seems natural to expect BLSTM RNN can also be effective for this task.[2]



Implement & Structure:

Employs a Sequential model with an embedding layer, a bidirectional LSTM, a dropout layer, and a dense layer, then computes the sparse categorical cross entropy loss Adam optimizer is used with learning rate of 0.001.

Layers:

Embedding Layer: Transforms words into vectors using pre-trained embeddings.

Bidirectional LSTM: This layer is a bidirectional LSTM (Long Short-Term Memory) layer is added with fixed ***32*** units, which is the size to get "reasonable" performance[2]. The LSTM layer is wrapped in a Bidirectional layer, allowing it to process the sequence data in both forward and backward directions, capturing dependencies from both ends of the sequence. Setting parameter '***return_sequences=True***' ensures that the LSTM layer returns the full sequence of outputs, which is necessary for sequence labeling tasks where a label is generated for each input element.

Dropout Layer: This layer randomly sets a fraction (***30***% in this case) of the input units to 0 at each update during training, which helps prevent overfitting.

Dense Layer: Uses a SoftMax activation function for predicting the probability distribution of part-of-speech tags.

<u>Training details:</u>

<u>Spilite&merge dataset:</u>

To be more robust, we merged train set and test set from listed website, as one synthetic training set. To test models are reliable, the new training set is randomly divided into proportions, with a ratio of 7:3. Shown in TABLE I.

I.

Data Set	***Train (Tokens)***	***Validation (Tokens)***
Before merging	211,727	47377
After merging	181372	77732

These 30% of data is used as validation data to monitor the model's performance on unseen data.

<u>Train Model:</u>

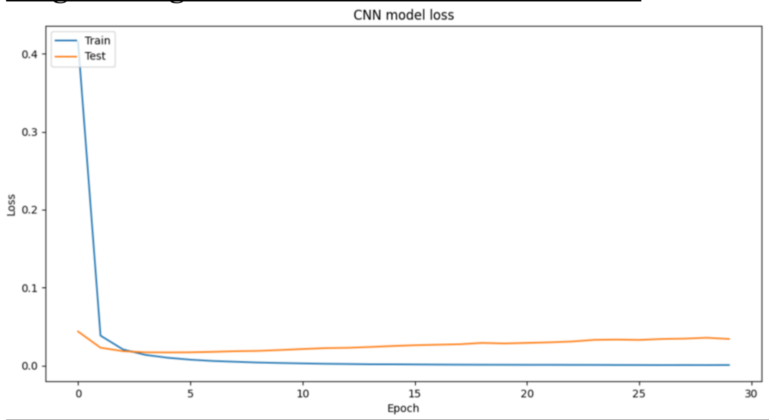
The model is trained for 30 epochs, setting batch size to 32 for each epoch. Plotting the training and validation loss by creating checkpoint callbacks during the process

<u>Evaluate Model & Result:</u>

<u>WSJ dataset:</u>

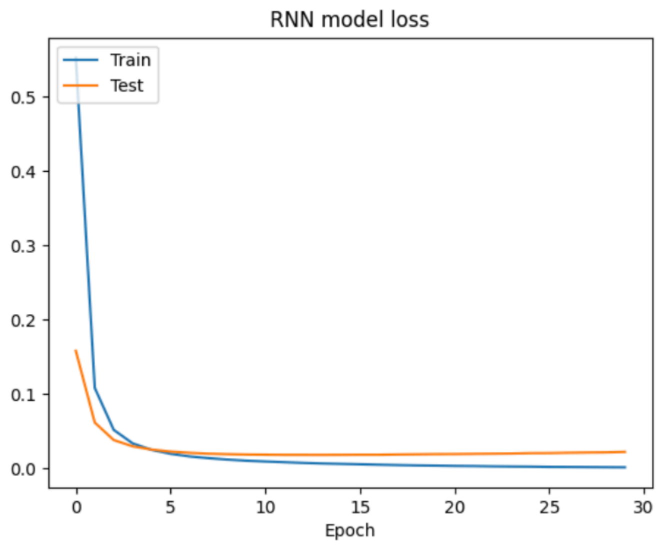
<u>CNN:</u>

```
              precision    recall   f1-score   support
   accuracy                          0.97      259025
  macro avg        0.93      0.87     0.88      259025
weighted avg       0.97      0.97     0.97      259025
```



<u>RNN:</u>

```
              precision    recall   f1-score   support
   accuracy                          0.97      258971
  macro avg        0.91      0.86     0.87      258971
weighted avg       0.97      0.97     0.97      258971
```



By looking at result of confusion metrics and loss functions, we can clearly see that the training results of CNN and RNN are similar around 97% on WSJ dataset and there is no sign of overfitting.

In-domain dataset:

CNN:

Total lines: 47377
Different lines: 2038
Percentage of differences: 4.301665365050552%
Correctness: 95.69833463494945%
==

RNN:

Total lines: 47377
Different lines: 2234
Percentage of differences: 4.715368216645207%
Correctness: 95.28463178335478%

Out-of-domain dataset:

CNN:

Total lines: 15185
Different lines: 3982
Percentage of differences: 26.22324662495884%
Correctness: 73.77675337504115%
==

RNN:

Total lines: 15185
Different lines: 4039
Percentage of differences: 26.598617056305564%
Correctness: 73.40138294369444%
==

Results & observations & conclusions:

Back to ***Evaluate Model*** part, the macro average precision and recall for the RNN are lower than for the CNN, which may indicate that the CNN performs better across all classes equally and a better balance between precision and recall across all classes. RNNs (especially BiLSTM in this case) are typically more complex and require more computational resources to train and run compared to CNNs. From the observation, RNN takes longer time than CNN does to train. In conclusion, 1D Convolutional Neural Network may be the best choice for part-of-speech task.

Challenges & Obstacles:

When doing this project, the largest issue I was facing that to tune hyperparameters of those models that training these models are time consuming subject. Besides, traditional CNN model usually has maxpooling layer, which mine does not because accuracy will reduce if I have built that in for this case.

References:

1. Bai, S.; Kolter, J.Z.; Koltun, V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. https://arxiv.org/abs/1803.01271
2. Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network https://arxiv.org/abs/1510.06168
3. https://www.cnts.ua.ac.be/conll2000/chunking/.