Sequence tagging

BENSRHIER Nabil

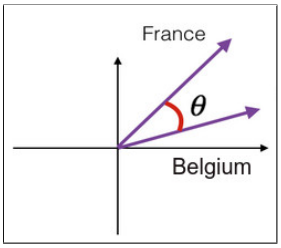
November 27, 2020

See code on **Github**.

In this project, we perform a RNN on CONLL03 named entity recognition (NER), and we will load pre-trained set of embeddings (100-dimensional GloVe vectors) to represent words since embeddings are very computationally expensive to train.

GloVe vectors:

These embedding vectors provide useful information about the similarity between words. To measure the degree of similarity between two vectors, we use the cosine function.
e.g. France and Belgium are quite similar means that θ is close to 0^0

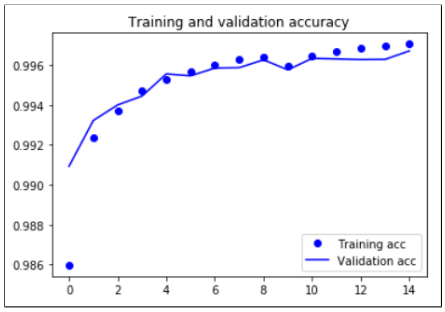


After collecting and extracting list of symbols X and Y, we merge training with pre-trained words in one dictionary $word2idx$ associating each word with an index starting the count from 2 (0 for padding and 1 for unknown words). Then we create the new emebdding matrix containing the pre-trained vectors and randomly initialized vectors for the unknown words.
Since the sentences have not the same length, we should add paddings in order to construct training sentences with the same shape. In this case, X_idx is the input of our model where the values are the keys of $idx2word$.

Evaluation: In this case, true positives and true negatives are not important compared to the crucial false negatives and false positives.
This is due to tht fact that we have imbalanced class distribution in our data between (padding, unknown and known classes).
So **F1-score** is a better metric to evaluate our model.

Model 1: Simple RNN

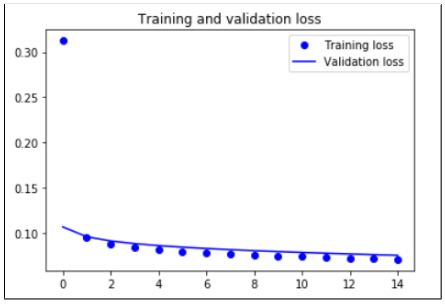
We use an **Embedding** layer which maps each input word to a 100-dimensional vector. We add the pre-trained weights without updating those embeddings.
The heart of the network is **SimpleRNN** with Bidirectional layer that gives a context determined by the data prior to and after each word. We dropout 20% of nodes to prevent overfitting. And a fully connected "dense" layer that generates probabilities for 11 target labels, using a **Softmax** activation function.



Here the model is compiled with 'rmsprop' optimizer and trained using the 'categorical_crossentropy' loss.
During the training, the network will try to minimize the log loss by adjusting the trainable parameters (weights).
Result:
We use the **predict** method to predict the tags of the whole test, and apply **conlleval** that reports the **F1-score**, then we get:

```
>>> F1 = 0.7938
```

Model 2: LSTM Network



This time we will use the same structure, but replacing simpleRNN by a LSTM cell, which has more additional mathematical operations and new set of weights.
And as we see, the is no sign of overfitting.

```
>>> F1 = 0.8309
```

Conclusion:
We conclude that LSTM gives better results, we can say that it controls the flow of Inputs as per trained Weights. So, LSTM gives us the most control-ability but comes with more Complexity and operation cost.

Contextual String Embeddings for Sequence Labeling by Akbik et al. (2018):

Different from our classic word embeddings, Akbik et al proves that his method of embeddings is highly useful for sequence labeling. Unlike our classic model, his method based on character level tokenization rather than word level tokenization. In other word, it converts the sentence to a sequence of characters, then computes embedding vectors under sentential context.
Stacking Embeddings:
It is the same idea that we used (stacking **GloVe** words with our **training** words), but in his case, Akbik et al uses different types of embeddings such as:

- **Proposed**$_{+WORD}$: Concatenating pre-trained static word embeddings with his contextual string embeddings.
- **Proposed**$_{+CHAR}$: Concatenating task-trained character features embeddings with his contextual string embeddings.
- **Proposed**$_{+WORD+CHAR}$: Concatenating both pre-trained words and task-trained character features.

Model Training and Parameters:
This approach uses the **BiLSTM-CRF** architecture. The same as our structure using LSTM variant of bidirectional recurrent neural networks, but here they add a decoding layer called (CRF).
They apply 25 hidden states to each word separately and extract the final hidden output states.

Results:

Approach	NER-English F1-score
proposed	
PROPOSED	91.97±0.04
PROPOSED$_{+WORD}$	93.07±0.10
PROPOSED$_{+CHAR}$	91.92±0.03
PROPOSED$_{+WORD+CHAR}$	**93.09**±0.12

Compared to our best F1-score= 83% their approach reached F1-score= 93.07% for the **Proposed**$_{+WORD}$ model, which produces the best results among the other models.
Therefore, they summarize that this approach **Proposed**$_{+WORD}$ captures world-level semantics that complement the strong character-level features of their Contextual String Embeddings.