Report of the 1st NLP Homework: Chinese Segmentation

Manuel Prandini ID: 1707827

I. INTRODUCTION

Reading [1], i have created a system made with Python 3.6.5 and I utilized libraries such as TensorFlow, Keras, Numpy to do preprocessing on the dataset files, and then to create the Bi-Lstm model and do training and test phases. The training was been done using the GPU on Google Colab.

II. PREPROCESSING

I downloaded the entire dataset from http://sighan.cs.uchicago.edu/bakeoff2005/ and i've semplified the content of two sub-datasets AS and CITYU from Traditional Chinese to Simplified Chinese. After i created the corresponding BIES format (the label) from the files because this files have separately words yet. I've create the input format, joining all characters for each row and eliminating the white space between words. AS and CITYU datasets represent the white space with the character *u3000* so i've substituted this with a white space and then i removed it. I've created two vocabulary, one for unigram and one for bigram, on the concatenation of all the four sub-datasets. In this way, i have more terms in the vocabulary. These two vocab contain also the UNK key useful to map the OOV words and each key has an integer as value. Then, i also created the vocabulary for the label, with four keys, the BIES letters. For each row of the input file, i splitted this row in unigram and bigram elements into two separately arrays, then for each array created, i map the corresponding elements using the unigrams and bigrams vocab created before. So i've created two arrays that correspond to my bi-lstm input. Using this method, i've converted also the label with the label's vocabulary, splitting each label row in unigrams. The i-row of the input unigram array has the same length of the i-row of the input bigram array and label array. This because when i split the row in bigrams, i join the last character of the sentence with <END> tag to make last bigram. To make the matrix input of the unigrams and bigrams, i've added zero padding.

III. MODEL

For the model i've implented a Bi-Lstm parallel that takes two inputs: the unigrams array and the bigrams array. After this, i make the *Embedding* of both, masking zero values and then do the *Dropout* on both the Embeddings. Then i concatenate they in a unique input for the Bidirectional Lstm layer. Also the Bidirectional-Lstm layer has the dropout and the recurrent dropout. For the ouput i've used *Time Distributed Layer* that takes as input a Dense softmax layer with 4 output: B, I, E, S. At the end, i've compiled the model with *Adam optimizer*, because it's more fast then *SGD* in training phase and i set the loss function to *categorical-crossentropy*. To pass the output label to the model i've had to convert the label array in the categorical form.

IV. TRAINING AND TESTING

I've train the model using the **batch generator**. This has improved the time of the training phase and the accuracy. I've utilized the hyperparameters used on the paper and also other values of hyperparameter to improve the models. I've trained different models on the four different datasets obtaining different results as shown in I. At the end i decided to train a model also on the concatenation of the 4 files. For each model i saved the best weights with *ModelCeckpoint* and i used also the *EarlyStopping* to prevent the overfitting but despite this the models have a little overfitting as showned in Figure[1,2,3,4]. Only the concatenation is going better as shown in Figure 5. For the training i used fixed hyperparameters such as **hidden size** to 256, **char embedding size** and **bigram embedding size** to 32. The various results are shown in Table I. Then i predict every single trained model on every single dataset and one on the concatenation of all datasets, as showned in Table II.

V. CONCLUSIONS

Changing the different hypereparameters i've obtained little changes from a model to another. All the model are in overfitting, but they have more or less **the same average precision, from 0.86 to 0.90**. The model that generalize better, in my opinion is the model **trained on the concatenation**, because obtain the **bigger value** on the concatenation of datasets and good values on the single datasets **from 0.89 to 0.92**.

Training AS			
N test	1	2	3
input drp	0.3	0.3	0.5
lstm drp	0.4	0.3	0.3
learning rate	0.035	0.04	0.0099
batch size	128	256	256
epochs	10/10	9/20	10/20
steps per epoch	100	200	200
val loss	0.1650	0.1492	0.1526
val acc	0.9357	0.9525	0.9509
precision	**0.9345**	**0.9524**	**0.9509**

Training CITYU			
N test	1	2	3
input drp	0.3	0.2	0.2
lstm drp	0.3	0.2	0.2
learning rate	0.04	0.04	0.035
batch size	256	256	128
epochs	5/20	5/20	11/20
steps per epoch	200	200	50
val loss	0.2276	0.1428	0.1370
val acc	0.9346	0.9559	0.9570
precision	**0.9342**	**0.9556**	**0.9566**

Training MSR			
N test	1	2	3
input drp	0.2	0.15	0.15
lstm drp	0.2	0.1	0.2
learning rate	0.002	0.002	0.03
batch size	128	128	128
epochs	10/10	10/10	12/15
steps per epoch	100	100	70
val loss	0.1017	0.0990	0.0919
val accuracy	0.9651	0.9669	0.9692
precision	**0.9628**	**0.9673**	**0.9696**

Training PKU		
N test	1	2
input drp	0.3	0.2
lstm drp	0.3	0.2
learning rate	0.002	0.035
batch size	128	128
epochs	12/20	6/20
steps per epoch	20	50
val loss	0.1803	0.1795
val accuracy	0.9409	0.9421
precision	**0.9419**	**0.9427**

Training Concatenation	
N test	1
input drp	0.4
lstm drp	0.4
learning rate	0.03
batch size	128
epochs	12/15
steps per epoch	70
val loss	0.2563
val accuracy	0.9120
precision	**0.9113**

TABLE I: Result of different training test on each single dataset and on the concatenation of them with various huperparameters. Each sub-table represent one model trained on one dataset

Test based Training AS	
Dataset	Precision
AS	0.9524
CITYU	0.8955
MSR	0.8430
PKU	0.8777
average	0.8921
concatenation	**0.8931**

Test based Training CITYU	
Dataset	Precision
AS	0.9027
CITYU	0.9566
MSR	0.8447
PKU	0.8961
average	0.9000
concatenation	**0.8896**

Test based Training MSR	
Dataset	Precision
AS	0.8424
CITYU	0.8159
MSR	0.9696
PKU	0.8511
average	0.8697
concatenation	**0.8796**

Test based Training PKU	
Dataset	Precision
AS	0.8598
CITYU	0.8557
MSR	0.8783
PKU	0.9427
average	0.8841
concatenation	**0.8878**

Test based Training Concatenation	
Dataset	Precision
AS	0.9282
CITYU	0.9129
MSR	0.8949
PKU	0.9090
average	0,9112
concatenation	**0.9113**

TABLE II: Result of the test. Each sub-table represent a model trained on one single dataset and also on the concatenation and the test is done on the other datasets and on the concatenation of all datasets. The average is between the four result of each single dataset

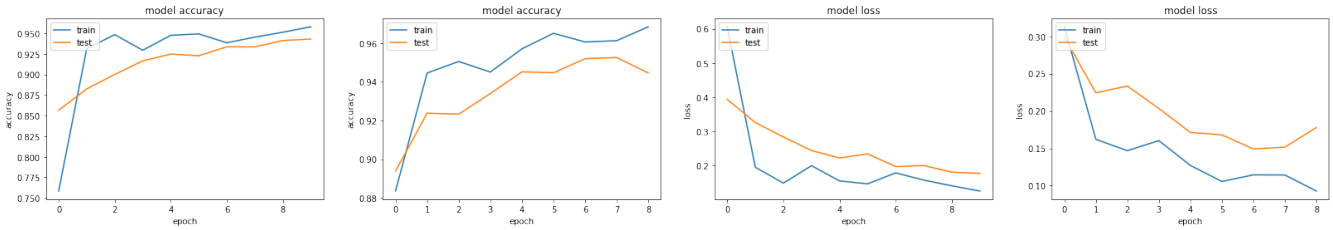

Fig. 1: Difference between accuracy and loss of respectively worst and better model trained on AS

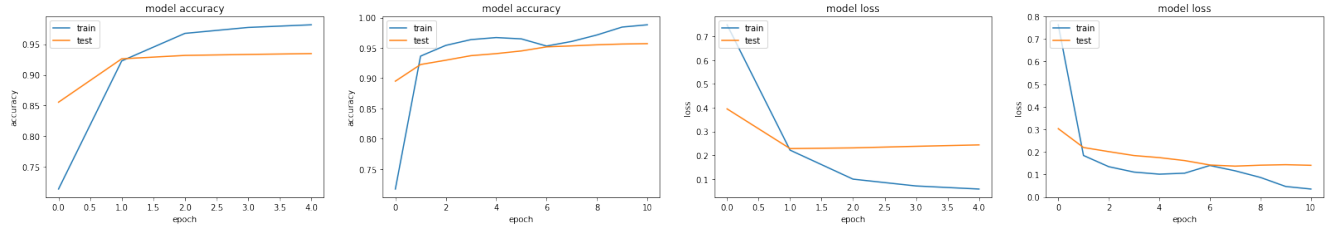

Fig. 2: Difference between accuracy and loss of respectively worst and better model trained on CITYU

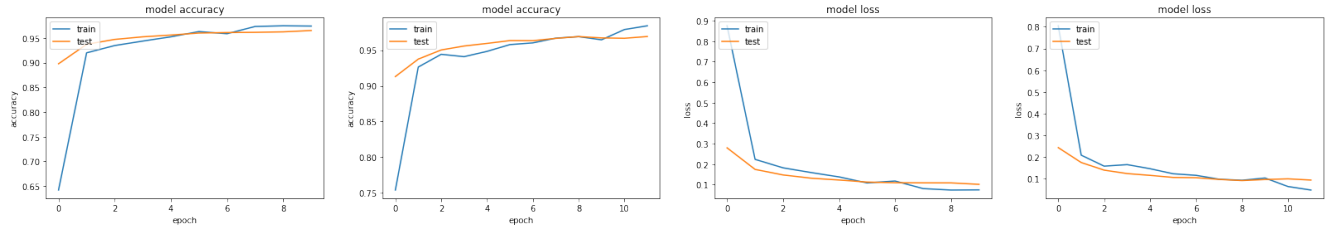

Fig. 3: Difference between accuracy and loss of respectively worst and better model trained on MSR

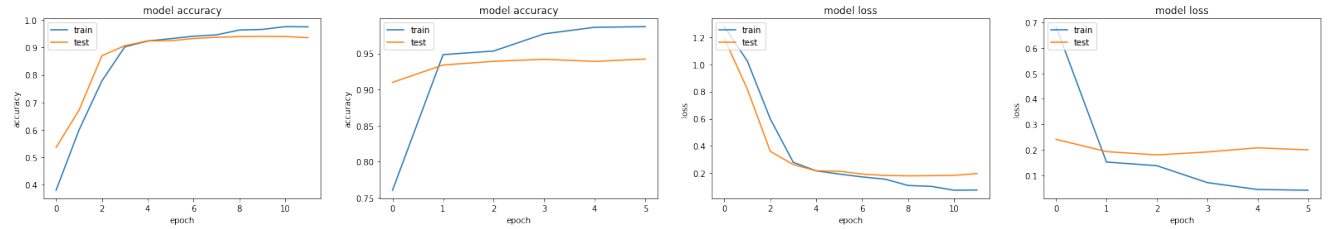

Fig. 4: Difference between accuracy and loss of respectively worst and better model trained on PKU

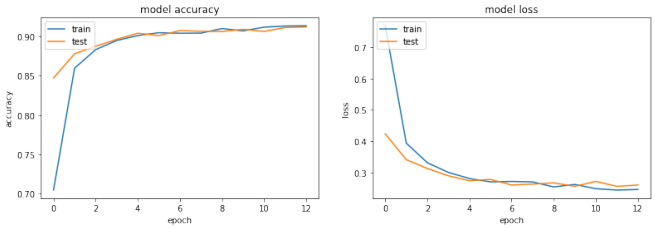

Fig. 5: Accuracy and loss of the model trained on the Concatenation

REFERENCES

[1] J. Ma, K. Ganchev, D. Weiss, State-of-the-art Chinese Word Segmentation with Bi-LSTMs, in CoRR, Aug 2018.