Chinese Segmentation, Report of NLP Homework 1

Silvio Severino 1705967

I. DATA PREPROCESSING

I begin removing the spaces from the datasets sentences (training data) and converting them in BIES form (label data). Following [1] I decided to use both unigrams then bigrams, concatenate them and use it as input. So, I splitted the sentences in unigrams and bigrams and I built a dictionary for each one. These two was necessary to map each sentence in order to obtain a numerical form, necessary for the network. Initially, I thought to build one dictionary for each dataset. But after some experiment, it emerged that the outcome was not good. So I decided to make a huge dictionary merging the unigrams/bigrams from all the dataset and the results improved. The outcome vocabolary has been used to map both the training sentences then the validation sentences. This was necessary to obtain correlation between input and output. Following [4] I decided to use two parallel embedding layers, one for the unigrams and one for the bigrams. I tried to use three different embeddings, two pretrained (wang2vec [2][5] and Baidu Netdisk [3][6]) and the keras embedding layer. Unfortunately, Baidu Netdisk is available only with embedding size 300 and I could not use it due to hardware limitations. However, during the training both the acc then the val_acc had good results, so with a performant hardware it could be well. So I tried to use wang2vec with embedding size 100 obtaining $\sim 86\%$ of precision on MSR. I obtained my best results with keras embedding layer no pretained using 32 as embedding size, obtaining 96.6% of precision on MSR.

II. MODEL

In order to reproduce the model of the main paper [1], I decided to use the Functional API of keras. In this way, it has been possible to have two parallel embedding layers and to build the non-stacking Bi-LSTM model. My model begins with two input layer, one for the unigrams and one for the bigrams. Then each input layer passes into an embedding layer. Each embedding layer has the dictionary size according to its input. After that, at each output I applied a Dropout layer and the output is concatenated. I tried to use both the stacking model then the non-stacking. For the stacking model I used an LSTM with the Bidirectional wrapper. For the non-stacking I used two parallel LSTM's, one that goes in forward way and another one that goes in backward way. I obtained my best results with stacking model. So the concatenated embedding goes in a Bidirectional LSTM and the output is given to a Dense layer with 4 as output size (B, I, E, S). Finally, I applied the TimeDistributed wrapper at the Dense layer because it allows to apply a layer to every temporal slice of an input. As optimizer I decided to use Adam due to hardware limitations. I try to use SGD too but training for few epochs I didn't obtain good results. Moreover I used $tanh$ as LSTM activation function, $Softmax$ as Dense activation function and $categorical\ crossentropy$ as loss function. I trained the model in batch mode, using a batch generator which splits the input data in some batch and applies the padding for each chunk, pairs to the maximum length of the batch.

III. TESTING

Due to hardware limitation, I did my test with a manual grid search. I did my main test on MSR, testing it both with more than one layer then only one (Tab. I, Fig. 1). With this dataset I obtained my best result using a single layer BiLSTM, obtaining $\sim 96.6\%$ of precision. The close one result was obtained with two layers BiLSTM. Moreover, MSR has been proved very robust to overfitting, as well as AS and CityU (Fig. 1, 3a, 4a). To avoid overfitting I used EarlyStopping technique. The most intresting one experiment was with the balanced concatenation of all four datasets, which predicted very well on all the dataset, although it was in overfiting (Fig. 6). From my standpoint training with that could be the best solution, also using SGD, grid search and more epochs.

IV. CONCLUSIONS

Following [1] I trained a model for each dataset. In my opinion, to obtain a very generalization, the best way to train the model is to use a balanced concatenation of all 4 datasets, so I decided to predict with that dataset obtaining the results shown in Tab. II.

TABLE I: Training with MSR trying three architectures

Training MSR			
Param.	2 BiLSTM	BiLSTM	2 LSTM
uni. emb.	32	32	32
big. emb.	32	32	32
hidd. siz.	256	256	256
uni. drop.	0.2	0.2	0.2
big. drop.	0.2	0.2	0.2
BiLSMT1 drop.	0.3	0.3	n.
BiLSMT2 drop.	0.3	n.	n.
LSTM drop.	n.	n.	0.2
f. LSTM drop.	n.	n.	0.2
b. LSTM drop.	n.	n.	0.2
l. rate	0.001	0.001	0.001
bat. siz.	128	128	128
epochs	15	10	10
stp. per ep.	100	100	100
val. per ep.	100	100	100
val loss	0.1108	0.0991	0.1306
val accuracy	0.9623	0.9668	0.9550
precision	0.962	0.9665	0.952

*The above precision is train and test on the same dataset

TABLE II: Training with all four datasets

Training MSR+AS+PKU+CITYU	
Dataset	**Precision**
AS	0.9108
CITYU	0.9302
MSR	0.8792
PKU	0.9308

*The above precision is train on concatenated datatset and test on each one

TABLE III: Training with AS

Training AS	
Param.	BiLSTM
uni. emb.	32
big. emb.	32
hidd. siz.	256
uni. drop.	0.6
big. drop.	0.6
BiLSMT drop.	0.6
l. rate	0.001
bat. siz.	128
epochs	10
stp. per ep.	100
val. per ep.	100
val loss	0.1805
val accuracy	0.9416
precision	0.9434

*The above precision is train and test on the same dataset

TABLE IV: Training with CITYU

Training CITYU	
Param.	BiLSTM
uni. emb.	32
big. emb.	32
hidd. siz.	256
uni. drop.	0.6
big. drop.	0.6
BiLSMT drop.	0.6
l. rate	0.001
bat. siz.	128
epochs	10
stp. per ep.	100
val. per ep.	100
val loss	0.1404
val accuracy	0.9545
precision	0.9465

*The above precision is train and test on the same dataset

TABLE V: Training with PKU

Training PKU	
Param.	BiLSTM
uni. emb.	32
big. emb.	32
hidd. siz.	256
uni. drop.	0.5
big. drop.	0.5
BiLSMT drop.	0.5
l. rate	0.007
bat. siz.	64
epochs	10
stp. per ep.	100
val. per ep.	100
val loss	0.2094
val accuracy	0.9453
precision	0.9219

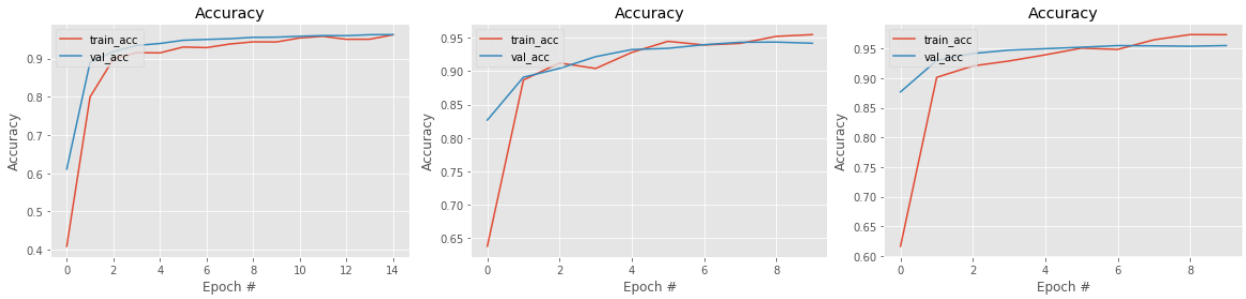
*The above precision is train and test on the same dataset

TABLE VI: Training with all datasets

Training MSR+AS+PKU+CITYU	
Param.	BiLSTM
uni. emb.	32
big. emb.	32
hidd. siz.	256
uni. drop.	0.5
big. drop.	0.5
BiLSMT drop.	0.5
l. rate	0.007
bat. siz.	64
epochs	10
stp. per ep.	100
val. per ep.	100
val loss	0.2811
val accuracy	0.9127
precision	0.9011

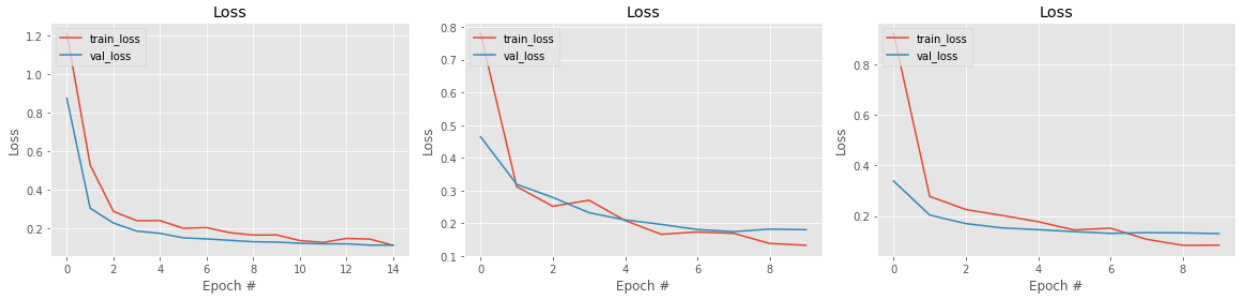
*The above precision is train and test on the same dataset

Fig. 1: MSR Accuracy



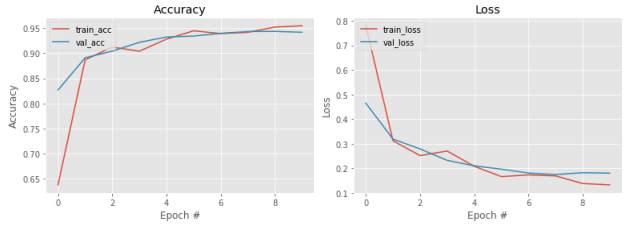
(a) 2 BiLSTM model **(b)** BiLSTM model **(c)** 2 LSTM model

Fig. 2: MSR Loss function



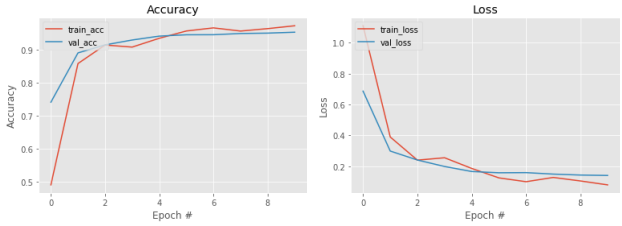
(a) 2 BiLSTM model **(b)** BiLSTM model **(c)** 2 LSTM model

Fig. 3: AS Accuracy and Loss



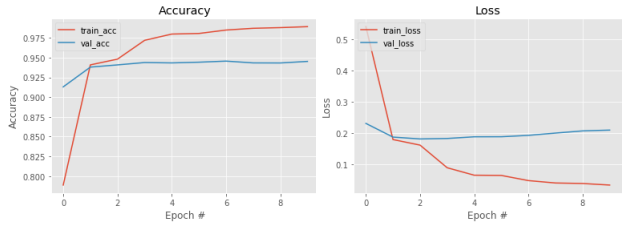
(a) BiLSTM model **(b)** BiLSTM model

Fig. 4: CITYU Accuracy and Loss



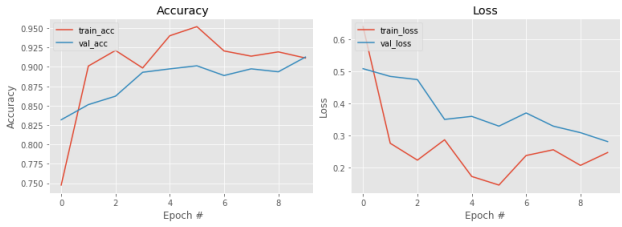
(a) BiLSTM model **(b)** BiLSTM model

Fig. 5: PKU Accuracy and Loss



(a) BiLSTM model **(b)** BiLSTM model

Fig. 6: All datasets Accuracy and Loss



(a) BiLSTM model (BEST) **(b)** BiLSTM model (BEST)

REFERENCES

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

[2] W. Ling, C. Dyer, A. W Black, I. Trancoso, Two/too simple adaptations of word2vec for syntax problems. In Proceedings of the 2015 Conference of the North American Chap-ter of the Association for Computational Linguis-tics: Human Language Technologies, pages 1299–1304, Denver, Colorado. Association for Computa-tional Linguistics. 2015.

[3] L. Shen, Z. Zhe, H Renfen, L. Wensi, L. Tao, D. Xiaoyong, Analogical Reasoning on Chinese Morphological and Semantic Relations, Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), 2018, Association for Computational Linguistics, 138–143, Melbourne, Australia, http://aclweb.org/anthology/P18-2023

[4] J. Yang, Y. Zhang, S. Liang, Subword Encoding in Lattice LSTM for Chinese Word Segmentation, in CoRR, abs/1810.12594, 2018

[5] icw2b-data: http://sighan.cs.uchicago.edu/bakeoff2005

[6] wang2vec: https://github.com/wlin12/wang2vec

[7] Baidu Netdisk: https://github.com/Embedding/Chinese-Word-Vectors