Analysis and Implementation of the State-of-the-Art Chinese Word Segmenter with Bi-LSTMs

Giada Simionato, 1822614
DIAG, Sapienza University of Rome – Via Ariosto, 25, 00185 Rome, Italy

I. Datasets Preprocessing

The first step in the preprocessing was to convert to Simplified Chinese all the files from the *AS* and *CITYU* datasets. The `preprocessing.py` script then created into the folder `code` a nested-folders-kind-of environment in order to provide all the data required from the other scripts. In the training data, it removed all the empty lines in the original sets while converting all the digits, punctuations and Latin letters to half-width to have a better match with the testing sets. Another key point in this process was to detect which symbols acted as separator for each dataset: a space for the *AS* dataset, the ideographic space (`U+3000`) for *CITYU* and double spaces for *PKU* and *MSR*. The RegEx syntax has been used to obtain the elements for the creation of the label files in BIES format. The lines were then unsegmented and put everything in the right folders. Analysing the test files it's been noticed that, conversely from what stated in (Ma et al., 2018), also these suffer from having those symbols in both half and full-width formats, hence, for the sake of consistency, these sets where normalized, while removing also blank sentences. The gold segmented files for the testing were translated and everything was make available in the correct folders of the environment. One more thing from this process is worth to be pointed out: during the analysis of the *MSR* test and test label files, some inconsistencies were discovered: by looking at the surroundings of these lines it's been noticed that the errors were from the incorrect segmentation among different lines in the dataset, in this way some characters that belong to a particular sentence where indicated as belonging to another one.

II. Baseline Model and Hyperparameters Tuning

This work focuses mainly on the *AS* dataset due to the fact that the two vocabularies (one for the characters and one for the bigrams), the training process and the testing one were done on it in its full dimension. Using the `Gensim` library, two vocabularies were obtained containing all the chars and bigrams with at least three occurrences, augmented to deal with padding and unknown chars or bigrams. The dataset was than processed to obtain two *numpy* arrays containing all the indices of the corresponding chars and bigrams to feed the network. In order to train it, an array of labels, that were been translated from *BIES* format to one-hot encoding, was also provided. The baseline architecture of the NN have the first three layers symmetric for the chars and the bigrams: a *input* layer, a *masking* one, for masking the effects of padding, and an *embedding* one for their creations. Then the embeddings of the chars and bigrams with size 64 and 32 respectively, were concatenated and fed into a *Bidirectional LSTM* layer with 256 hidden units and 0.2 of probability of both *dropout* and *recurrent dropout*. The output dense layer had 4 units. The optimizer of the network was a *Stochastic Gradient Descent with momentum* with *learning rate* of 0.04 and *momentum* 0.9. The *batch size* was 32 and *maximum length* 30. In this work all these hyperparameters were changed and compared w.r.t. at least another value, then the best was taken and used for subsequent analyses. Deeper considerations on the results of comparisons are made directly under the tables and plots. The choice of the optimizer was the first one: the results of *Adam*, *Adadelta* and *SGD with momentum* are collected in Table I. The second one was the *embedding size* of the *bigrams* among *16, 32* and 64 with the results in Table II. Also the *dropout* and *recurrent dropout* rates were compared to 0.5 as stated in Tables III and IV. A deeper analysis of the dataset described in Fig. 1, forced to change the *maximum length* to 50. It was then proved that the current version was better than the stacked one depicted in Fig. 2 according to results in Table V. The number of *hidden units* in the Bi-LSTM was set to 512 as shown in Table VI. The batch size appeared not to influence the performances, while using more datasets to build vocabularies decreased them.

III. Best Model Architecture and Results

The best model is a three-layered Bi-LSTM network whose input, masking, embedding and dense layers have the same structure of the baseline but followed from the three above-mentioned layer. After a grid-search for the hyperparameters for this network, based on the previous comparisons, it's been trained with the hyperparameters described in Table VII. Results and comparison with the baseline are shown in Table VII. It's clear that to improve these performances it's necessary to use pre-trained embeddings, as *ElMo* ones, that provide contextualized vector representation. An usage of these was tried but the unavailability of enough resources forced to abort the experiment.

IV. Tables and Figures

Optimizer	Training		Validation		
	Loss	Acc.	Loss	Acc.	time [s]
SGD+momentum	0.0833	0.9652	0.1633	0.9486	**2280**
Adadelta	0.0959	0.9675	0.1647	0.9465	2577
Adam	**0.0555**	**0.9801**	**0.1376**	**0.9514**	2345

Table I: The network was trained for three epochs with the following hyperparameters: *batch size: 32*, *char emb. size: 64*, *bigram emb. size: 32*, *dropout* and *rec. drop.: 0.2*, *max length: 30*, *hidden size: 256*. The optimizers have the default values except for the SGD with *learning rate: 0.04* and *momentum: 0.9*. Adam optimizer provided the best results and was the fastest to converge. SGD was the fastest to elaborate the data but considering the available resources Adam was preferred.

Bigram Emb. Size	Training		Validation		
	Loss	Acc.	Loss	Acc.	time [s]
16	0.0515	0.9815	0.1435	0.9500	**2345**
32	0.0555	0.9801	**0.1376**	**0.9514**	3250
64	**0.0471**	**0.9831**	0.1509	0.9512	3904

Table II: The network was trained for three epochs with the following hyperparameters: *batch size: 32*, *char emb. size: 64*, *optimizer: Adam*, *dropout* and *rec. drop.: 0.2*, *max length: 30*, *hidden size: 256*. Since the results on the validation set are more important, the bigram embedding size was set to 32.

Dropout	Training		Validation		
	Loss	Acc.	Loss	Acc.	time [s]
0.2	**0.0555**	**0.9801**	**0.1376**	**0.9514**	**3250**
0.5	0.0594	0.9788	0.1567	0.9501	3479

Table III: The network was trained for three epochs with the following hyperparameters: *batch size: 32*, *char emb. size: 64*, *bigram emb. size: 32*, *optimizer: Adam*and *rec. drop.: 0.2*, *max length: 30*, *hidden size: 256*. Despite one of the main problem for this network is the overfitting, use a stronger dropout doesn't improve the performances. The dropout rate remained 0.2.

Recurrent Drop.	Training		Validation		
	Loss	Acc.	Loss	Acc.	time [s]
0.2	0.0555	0.9801	**0.1376**	**0.9514**	**3250**
0.5	**0.0533**	**0.9808**	0.1500	0.9503	2363

Table IV: The network was trained for three epochs with the same hyperparameters described in Table 3. For the same reasons the recurrent dropout remained at 0.2.

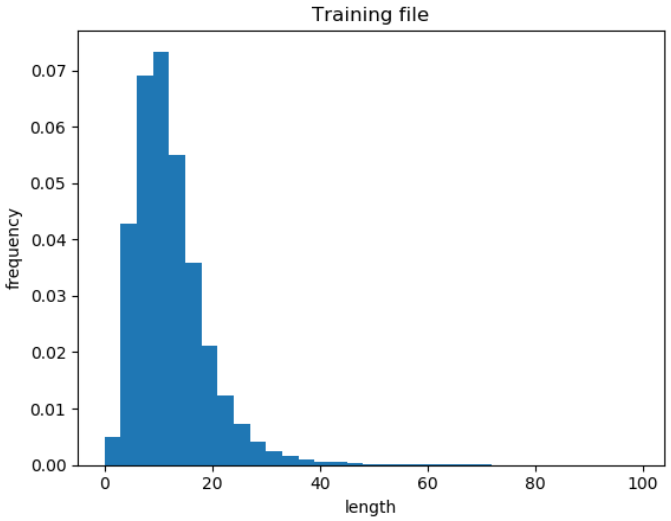

Figure 1: This plot represents the relative frequencies of the lengths of all the sentences in the AS dataset. This shows that most of the lines have length under 20 characters. For a better training process the maximum length of the sentence was then set at 50 to reduce the number of lines to truncate.

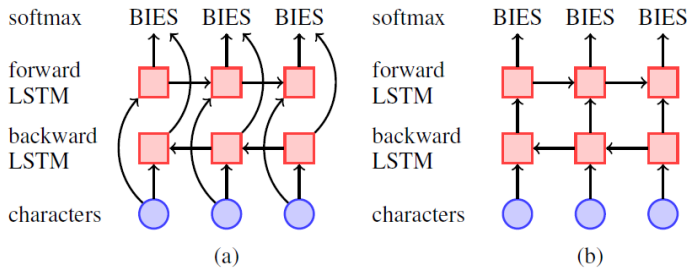

Figure 2: This plot from (Ma et al., 2018) shows the two structures compared in Table 5. a) non-stacked model, the baseline of the implementation while b) stacked model, implemented with the same hyperparameters of a) allowing a comparison.

	Training		Validation		
Model	Loss	Acc.	Loss	Acc.	time [s]
a	**0.0555**	**0.9801**	**0.1376**	**0.9514**	**3250**
b	0.2413	0.3711	0.2568	0.3544	2652

Table V: These results were obtained after a three epochs training over the implementation of the two models depicted in Fig 2. After only three epochs the stacked model shows a non-convergence behaviour in the short run. For timing reason the first model, the tuned baseline one, was selected for further hyperparameters tuning.

	Training		Validation		
Hidden Size	Loss	Acc.	Loss	Acc.	time [s]
256	**0.0555**	**0.9801**	0.1376	0.9514	**3250**
512	0.1159	0.9587	**0.1352**	**0.9561**	4283

Table VI: The first row shows the results obtained after have trained the network for 3 epochs with *batch size: 32*, *char emb. size: 64*, *bigram emb. size: 32*, *dropout* and *rec. drop.: 0.2*, *max length: 50* and *optimizer: Adam*. While the second rows shows the model, trained for only one epoch and with the same parameters but with the double of hidden units in the Bi-LSTM layer.

	Training		Validation		
Implementation	Loss	Acc.	Loss	Acc.	time [s]
Baseline	0.0833	0.9652	0.1633	0.9486	**2280**
Best Model	**0.0627**	**0.9778**	**0.1284**	**0.9575**	11844

Table VII: The baseline model was trained for three epochs with the hyperparameters described Section II. The best model, however was trained only for two epochs with the same hyperparams of the baseline but with the *first layer* of 512 *hidden units*, the *second and third* layers of 256 *units*, *dropout, first layer recurrent dropout* of 0.2 while *second, third layers rec. drop.* of 0.33 and *Adam* as *optimizer*. Is worth to mention that nevertheless the best model gives better performances, the amount of time required for training is a feature to consider.

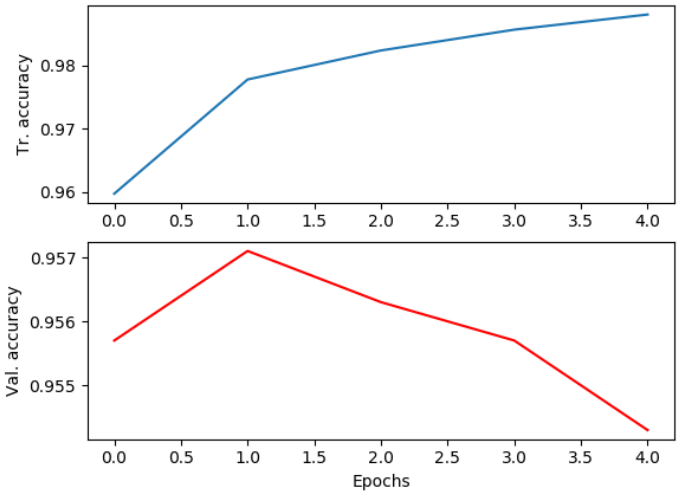

Figure 3: The best model was firstly trained for one epoch to test its potential. Once having become the best one, it was trained for five epochs. This gave rise to the two behaviours depicted in the plots above. The blue line represents the behaviour of the training accuracy and shows an increasing trend over the epochs, while the red line, namely the validation accuracy, shows a maximum peak of accuracy and then starts to slightly decrease: this may be due to the presence of overfitting. A further development could be insert a *L2* regularization term to reduce this drawback.