Homework 1 - Natural Language Processing

In this project it has been used the paper 'State-of-the-art ChineseWord Segmentation with Bi-LSTMs' to build the model.
The built model consists in a stacking bidirectional LSTM that takes in input two concatenated embedding layers, one for unigrams, the second one for bigrams. The output is a 4-way Dense layer that outputs the probabilities of the BIES tags (See Figure 1).
The model has been tested on Google Colab on GPU.
Best Hyperparameters found:

Hyperparameters	Value
Batch size	512
Unigrams embedding size	128
Bigrams embedding size	64
Hidden size	256
Epochs	20
Line max length	200

The score on the gold test of MSR is 0.9645. See Figure 2 and Figure 3 for accuracy and loss functions.

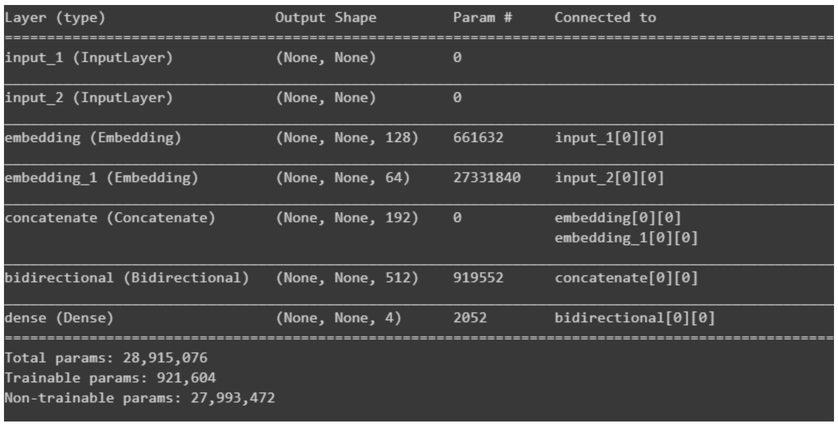
Dataset

The dataset used for training is msr,in the file there was some empty lines that have been deleted by the preprocess script. I've also made some tests on pku but it gave worst performances respect to msr. I tried also to concatenate msr and pku but the network easily overfits (See Fig 8 and Fig 9). To speed up the training, the lines are truncated at 200 characters in the batch generator if the maximum length in the batch exceeds the MAX LENGTH provided, otherwise they are truncated to the length of the line of maximum size in the batch. As development and test set I used the msr gold file obtaining the accuracy reported above but switching to another gold file, the accuracy decreases for example to 0.856 on pku gold.

Tests

To prevent overfitting I used the early stopping, so the training stops before the network begins to overfit. I've also used the pre trained chinese word vectors provided to this link . We can notice that without pre trained embeddings, making the embedding with the keras layers, the network overfits very fast, after 2 or 3 epochs (see Firgure 6 and Figure 7), whereas using the pre trained embedding, it overfits after 17 epochs.

The optimizer used in the project is Adam with a learning rate of 0.002 that performs quite well on the network, I've also tried to use the Momentum Optimizer with same learning rate and momentum 0.95 like suggested in the paper but the train became slow and I obtained better results with Adam optimizer. I tried also to add another bidirectional layer, but with two layers the network overfits after few epochs. Increasing the embedding size the performances improve, while decreasing the embedding size I got lower accuracy but not very bad results (See Figure 4 and Figure 5).



```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, None)         0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, None)         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, None, 128)    661632      input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 64)     27331840    input_2[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, None, 192)    0           embedding[0][0]
                                                                 embedding_1[0][0]
__________________________________________________________________________________________________
bidirectional (Bidirectional)   (None, None, 512)    919552      concatenate[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, None, 4)      2052        bidirectional[0][0]
==================================================================================================
Total params: 28,915,076
Trainable params: 921,604
Non-trainable params: 27,993,472
```

Figure 1: Model structure

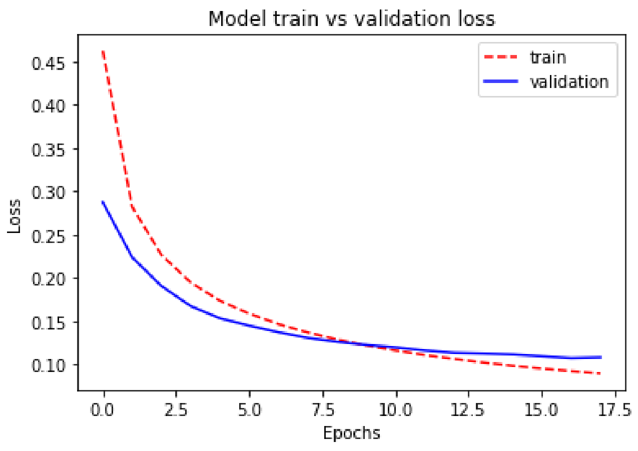


Figure 2: Loss function best model

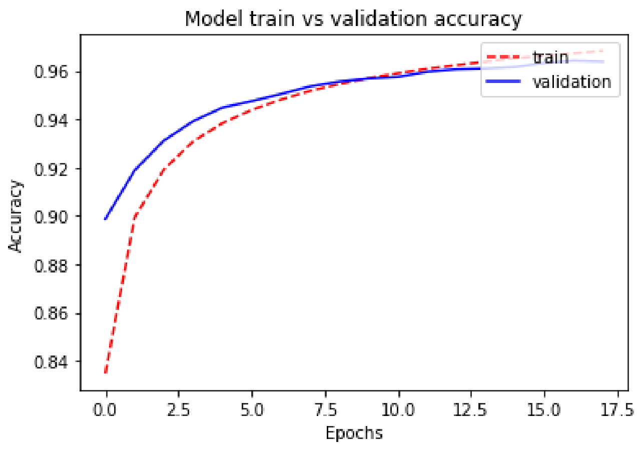


Figure 3: Accuracy best model

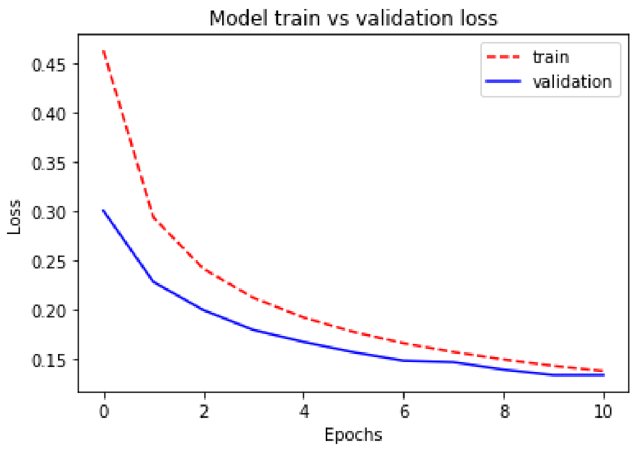


Figure 4: Loss function 64/32 embedding size

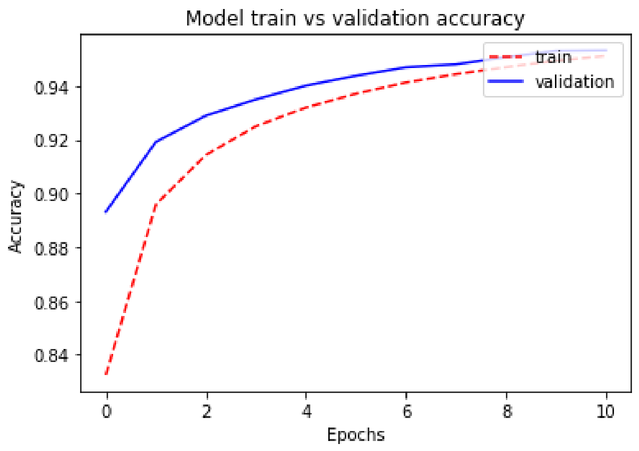


Figure 5: Accuracy 64/32 embedding size

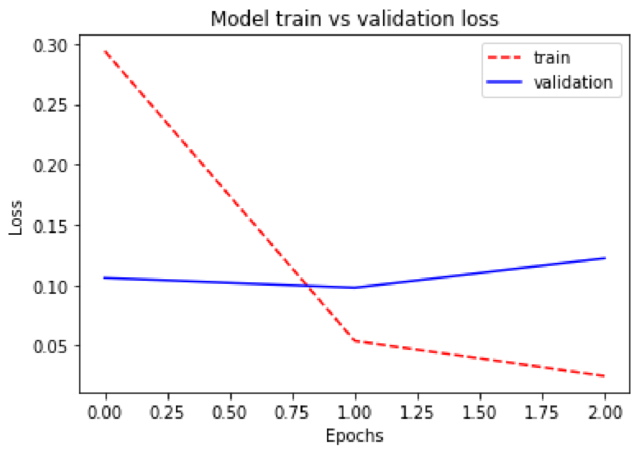


Figure 6: Loss function without pre trained

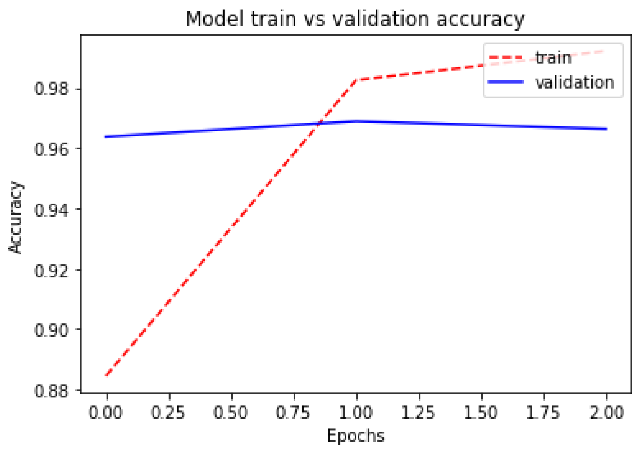


Figure 7: Accuracy without pre trained

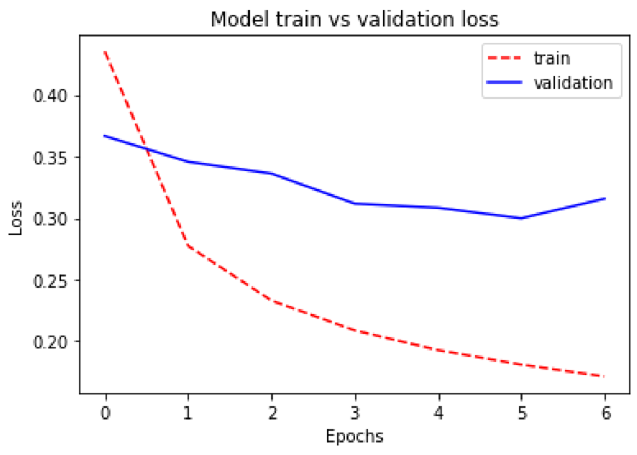


Figure 8: Loss function msr concatenated pku

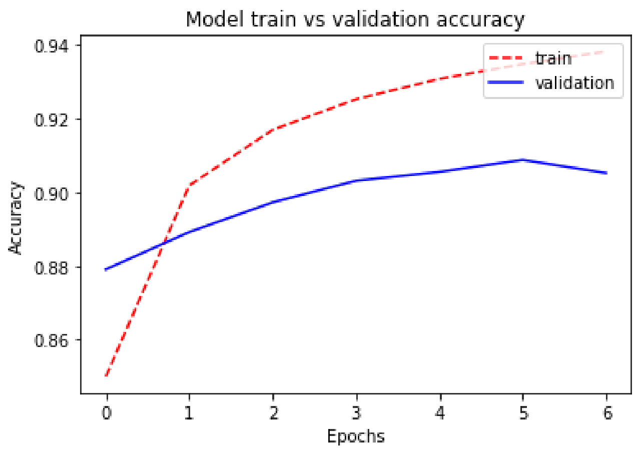


Figure 9: Accuracy msr concatenated pku