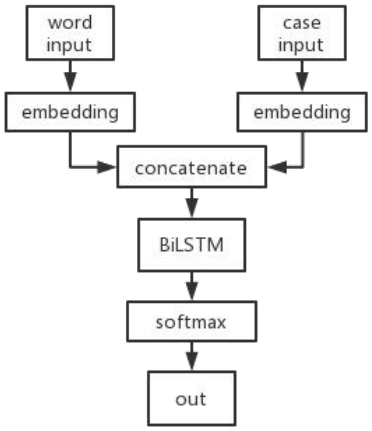
- BiLSTM

1. tagger description



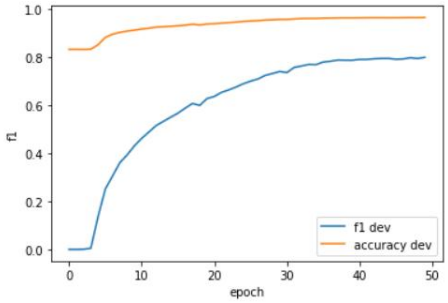
For the input of the model, I encode the word after lowercase and to provide more information I also classify the words into 8 cases ("PAD", "numeric", "allLower", "allUpper", "initialUpper", "other", "mainly_numeric" and "contains_digit").

After concatenate word and case, I put them into Bidirectional LSTM, and them softmax the output layer to prediction matrix. Then use sparse_categorical_crossentropy to calculate the loss and choose nadam as optimizer.

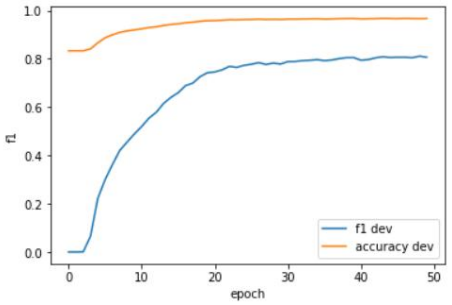
Hyperparameters: epoch = 50, embedding_size = 50, batch_size = 500, LSTM_state_size = 200, dropout = 0.5, dropout_recurrent = 0.25.

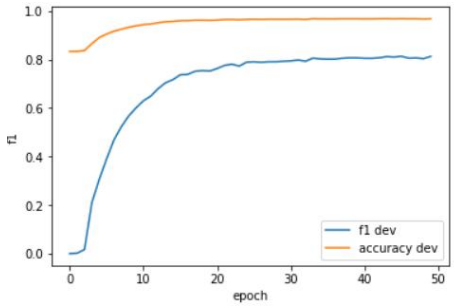
2. Fine tuning

Initial parameters: epoch = 50, embedding_size = 20, batch_size = 500, LSTM_state_size = 200, dropout = 0.5, dropout_recurrent = 0.25. Under this setting, f1 of dev set is 0.79 after training finished and it has been convergent.

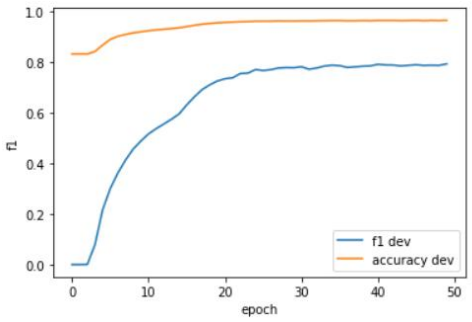


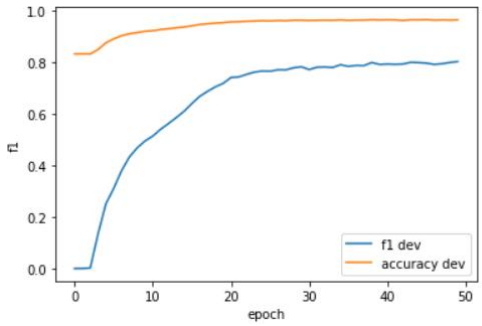
I change embedding size to **50** and 100 separately. F1 slightly increased to 0.807 and 0.813 and the number of epoch which model need to convergence f1 get smaller. But it spend a lot more time to train when embedding_size = 100, so I think it is better to choose 50.



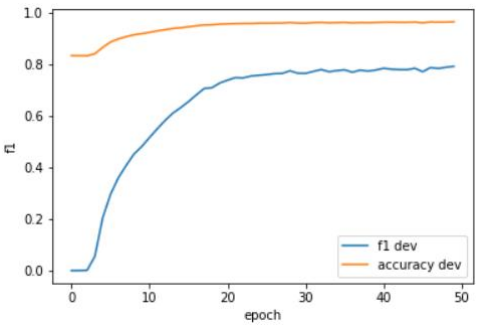


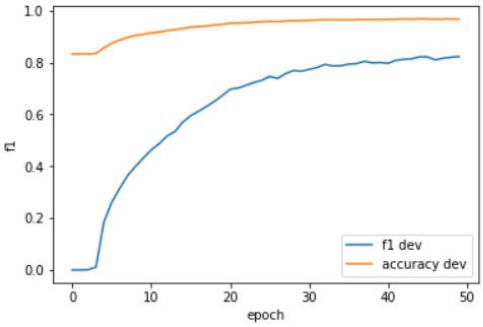
I change dropout rate between hidden layers to 0.2 and 0.3 separately. There is not clear change in show in fig.





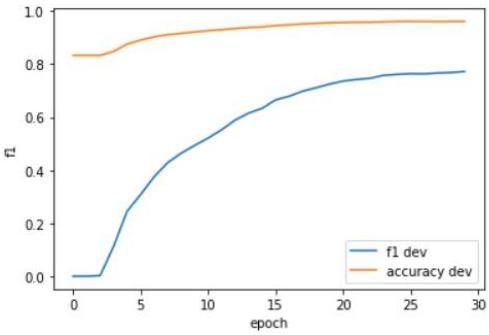
I change dropout rate between input and hidden layer to 0.2 and 0.8 separately. There is not clear change in show in fig.





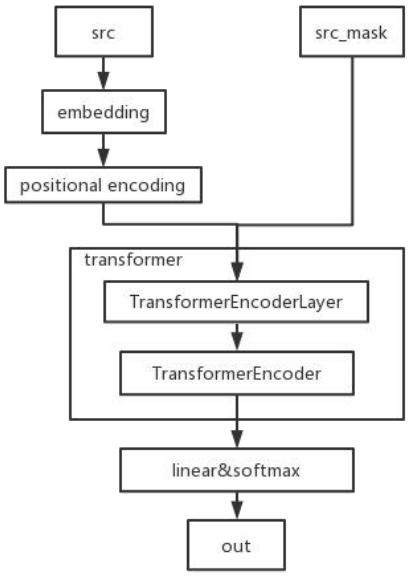
I change LSTM_state_size to 100. There is not clear change in show in fig.

Finally I change epoch to 30 which is enough for f1 to get convergent. epoch = 50, embedding_size = 50, batch_size = 500, LSTM_state_size = 200, dropout = 0.5, dropout_recurrent = 0.25.



- Transformer

1. tagger description



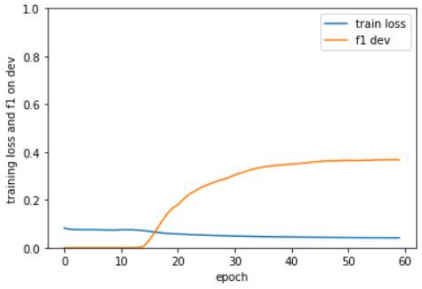
For the input of model, I still use encoded words in the sentence and use positional encoding to add position info, besides I generate src_mask for the model to just pay attention on previous words.

After putting src and src_mask into transformer and completing encoding, I use linear and softmax to get prediction. Then calculate loss and use SGD to optimize.

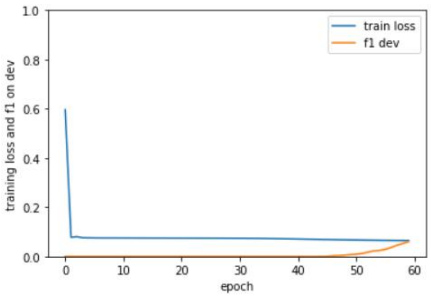
Hyperparameters: epoch = 60, embedding_size = 90, d_hidden = 100, n_layers = 2, n_head = 3, dropout = 0.2.

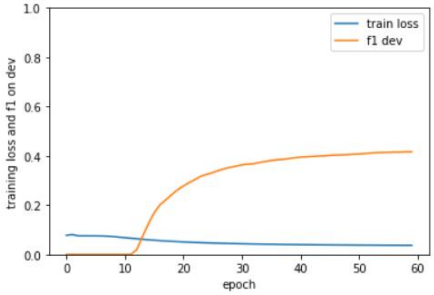
2. Fine tuning

Initial parameters: epoch = 50, embedding_size = 50, d_hidden = 200, n_layers = 2, n_head = 2, Dropout = 0.2. F1 score at last epoch is 0.367, and it has been convergent.

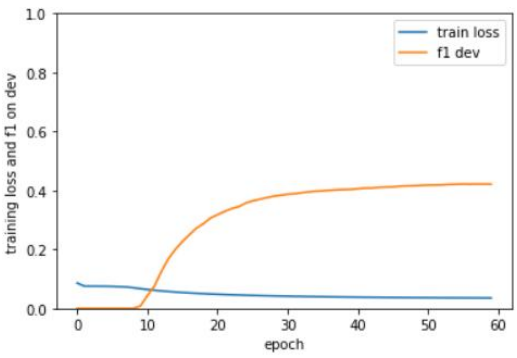


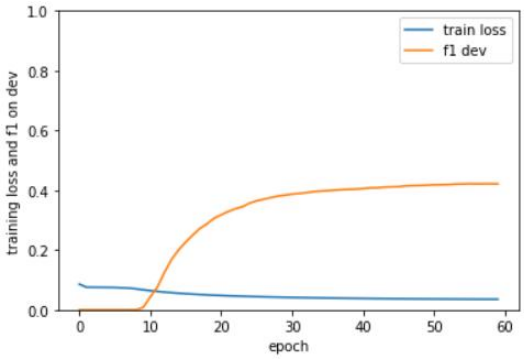
I change embedding size to 20 and **100** separately. There is a sharp drop of f1 at last epoch when embedding_size is 20 and f1 increases to 0.416 when embedding_size is 100.It is a better choice.



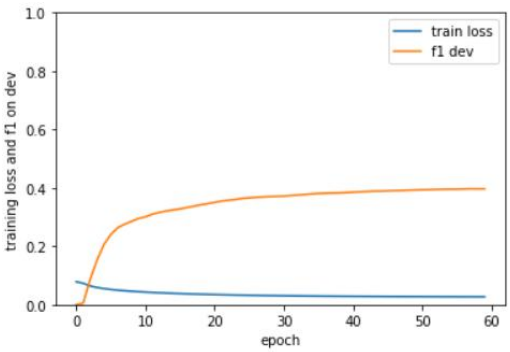


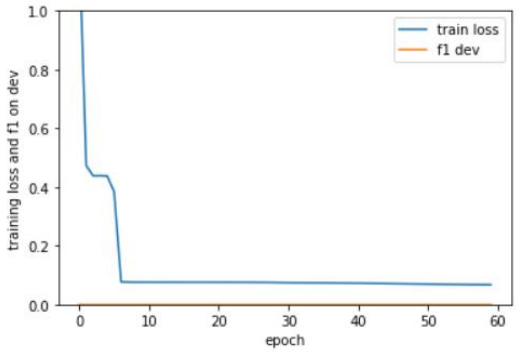
When I change the dimension of feed forward network in transformer encoder to **100** and 200, f1 slightly increases to 0.421 and decreases to 0.386 separately. So it is better to choose 100.



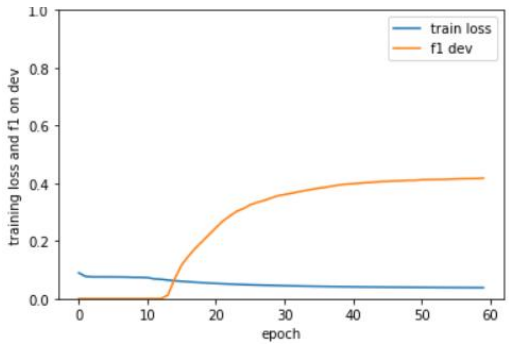


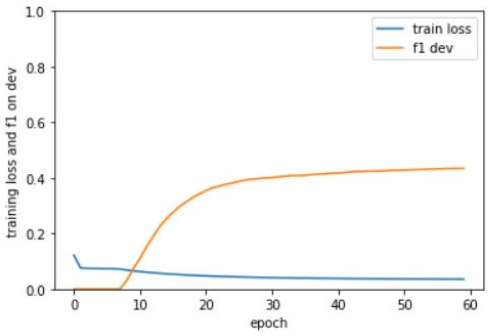
I change the number of encoder layers in the transformer to 1 and 3 separately. When n_layers = 1, f1 at last epoch slightly decreases to 0.397 though it performs better in smaller epoch. When n_layers = 3, it may take a lot more epoch to learn.



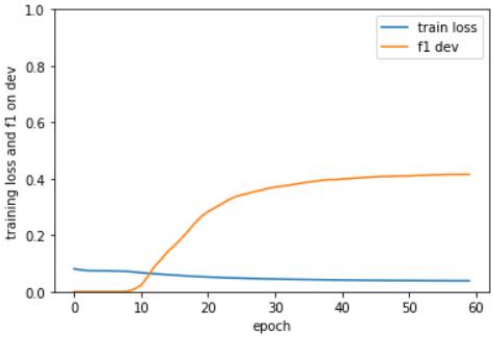


When I change the number of heads in multi head attention to 1 and 3(corresponding embedding_size = 90), f1 slightly decreases to 0.417 and increases to 0.434 separately. So it is better to choose n_head=**3** with epoch = **90**.





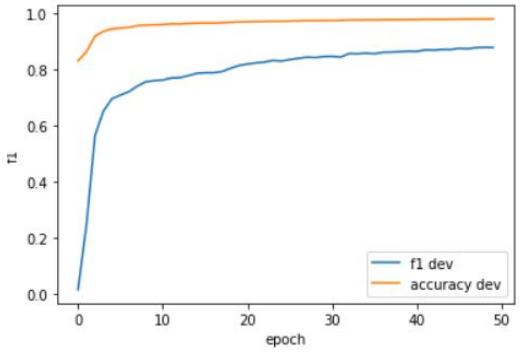
When I changing the dropout rate to 0.4, f1 decreases to 0.415.



Finally the hyparameters is get is: epoch = 60, embedding_size = 90, d_hidden = 100, n_layers = 2, n_head = 3, dropout = 0.2.

- BiLSTM with GloVe

Instead of random word embedding like what I have used in the first case, this time I used pre-trained wording embedding GloVe with embedding dimension of 50, and increase epoch to 50 to get f1 convergent while other parameters don't change.



At last epoch, f1 reaches 0.878.

- Result and analysis

	BiLSTM			BiLSTM with GloVe			Transformer		
	Train	Dev	Test	Train	Dev	Test	Train	Dev	Test
f1	0.968	0.772	0.662	0.926	0.878	0.829	0.476	0.434	0.389
accuracy	0.995	0.962	0.937	0.989	0.980	0.969	0.903	0.889	0.872

We can analyse from the result table that:

a. Accuracy is not that useful in the NER task, because most tags in the dataset is "O", even if all we predict is "O", we can still get an accuracy which is quiet high. So we next focus on f1 score.

b. We can see that f1 on the train set is always higher than that on devset and higher then that of train set. It is a quite common result, because we learn the characteristic on train set, and adjust the parameters on dev set.

c. Compared with BiLSTM, Transformer performs worse on NER task. Maybe because the positional encoding it uses is lack of direction info and position info is also lost during self-attention. It is better to use Bert.

d. BiLSTM with GloVe performs better than that without GloVe. Because GloVe provides the meaning of words and can indicate relationship between words.