Model comparison：

Normal RNN：

encoder：two layers bidirectional LSTM
encoder output：all hidden is used as context vectors, the last hidden is used as encoder hidden

decoder：one layer unidirectional LSTM (with attention and copy attention)

encoder and decoder use the same word embedding，the size of vocab is 30522
encoder and decoder use the same tokenize function as what BERT used

number of parameters：10,683,392 (encoder) + 29,103,922 (decoder) = 39,787,314

RNN with BERT：

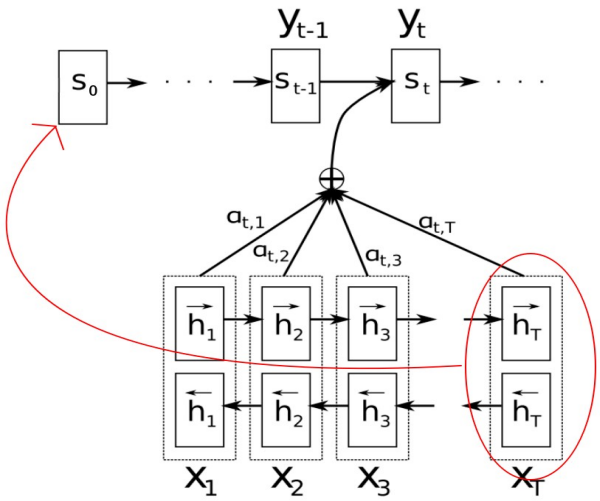
encoder：BERT
encoder output：the last layer of BERT encoder layers is used as context vectors, and its last element is used as encoder hidden

decoder：one layer unidirectional GRU (with attention and copy attention)

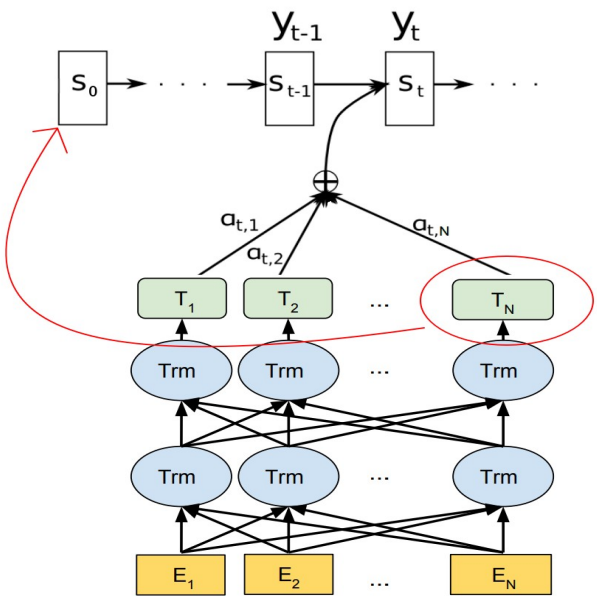
the word embedding of encoder is already in BERT，the size of vocab is 30522
decoder use another word embedding function，the size of vocab is 30522，and it is initialized as the weight as the word embedding in BERT
decoder use the same tokenize function as what BERT used

number of parameters： 109,875,968 (encoder) + 43,166,010 (decoder) = 153,041,978

Comparison chart：



Normal RNN



RNN with BERT

Training data：

the number of training data：2719767
the number of validating data：73（small amount for quick validating）

Example:

Source Text
Just-In-Time (JIT) Compilation is a technology used to improve speed of virtual machines that support dynamic loading of applications. It is better known as a technique that accelerates Java programs. Current JIT compilers are either huge in size or compile complete methods of the application requiring large amounts of memory for their functioning. This has made Java Virtual Machines for embedded systems devoid of JIT compilers. This paper explains a simple technique of combining interpretation with compilation to get a hybrid interpretation strategy. It also describes a new code generation technique that works using its self-code. The combination gives a JIT compiler that is very small (10K) and suitable for Java Virtual Machines for embedded systems.
Keyphrasess
Java; dynamic compilation; jit; embedded system; code generation

There are five keyphrases in the above, so the above is split into five samples (each sample is put into the seq2seq model for training) :

(Source: Just-In-Time (JIT) … ; Target: Java)
(Source: Just-In-Time (JIT) … ; Target: dynamic compilation)
(Source: Just-In-Time (JIT) … ; Target: jit)
(Source: Just-In-Time (JIT) … ; Target: embedded system)
(Source: Just-In-Time (JIT) … ; Target: code generation)

Experimental comparison：

For some objective reasons, the model was run only half an epoch。

The batch size of the Normal RNN model is set to 128.
The RNN with BERT model batch size is set to 10 due to memory limitations.

The number of batches that the RNN with BERT model needs to train is about 16 times that of the Normal RNN model, so we do validating of the 20000 and 2000 steps respectively.

“loss” means the average loss generated during training, and “f1 score” means the f1 score using top 5 predictions when validating.

RNN with BERT model:

Epoch	batch id	finish	Train-time	loss	PPL	f1 score
1	20000/351020	5.70%	4098s (1.14h)	4.7217	112.4	0.1325
1	40000/351020	11.40%	8217s (2.28h)	4.4182	82.94	0.1246
1	60000/351020	17.09%	12364s (3.43h)	4.2848	72.59	0.1547
1	80000/351020	22.79%	16484s (4.58h)	4.2024	66.85	0.126
1	100000/351020	28.49%	20619s (5.73h)	4.1508	63.48	0.1348
1	120000/351020	34.19%	24778s (6.88h)	4.1135	61.16	0.1346
1	140000/351020	39.88%	28917s (8.03h)	4.0842	59.4	0.1388
1	160000/351020	45.58%	33050s (9.18h)	4.0613	58.05	0.1242

Normal RNN model:

Epoch	batch id	finish	Train-time	loss	PPL	f1 score
1	2000/22024	9.08%	827s (0.23h)	4.0396	56.8	0.198
1	4000/22024	18.16%	1655s (0.46h)	3.6271	37.61	0.2208
1	6000/22024	27.24%	2483s (0.69h)	3.4135	30.37	0.2149
1	8000/22024	36.32%	3315s (0.92h)	3.2709	26.34	0.2529
1	10000/22024	45.41%	4139s (1.15h)	3.1634	23.65	0.2452

It can be seen from the above operation diagram:

* The training time required for the Normal RNN model is much smaller than the RNN with BERT model, and its f1 score is basically increasing at each time step, but the f1 score of RNN with BERT model is always maintained at a very low level.

* You can see that when the RNN with BERT model is trained to 160,000 and the Normal RNN model to 2000 the loss and PPL are similar, but the f1 score on the validating set is much different.