COMP5046A2 Group222 Report

Yichen Chen [1]

[1] University of Sydney, NSW 2006, Australia

yche3494@uni.sydney.edu.au

1 Data preprocessing

No Content

2 Input Embedding

1. Syntactic embedding

Using Spacy library to generate 3 grammatical or syntactic embeddings: Part-of-speech tagging, Dependency parsing, Named entity recognition. The model of Spacy generating these embedding is pretrained on external corpus, there's no any manual tagging syntactic information in our CONDA data set, so we decide to use this Spacy library.

Justification:

For part-of-speech tagging, it can extract what part of word in a sentence, which could be a noun, a verb, an adverb etc. This information can be helpful to do our tagging detection task, because if model know a word is a noun, it will be more likely to be recognized as a D (Dota-specific) or C(Character).

As for dependency parsing, it can indicate the relationship between two phrases, depends on different relationship, model may adjust word's meaning using this dependency information, for example, 'I hate dota2', 'I' here will get a 'nsubj', which can distinguish it to other word 'I' which not get a 'nsubj'.

For Named entity recognition, it can tell you 'Lina' is a person, 'English' is a kind of language, etc. If model know a word such as 'Lina' is a person, it can tend to predict it's a C(Character) game character, I think.

2. Semantic embedding

Extra: Baseline pretrained embedding Glove-giga-word-25,

Justification: No special reason, just to reproduce baseline performance in USYD COMP5046 lab9 for comparison.

To generate semantic embedding for our task, I decide to using original CONDA dataset, extract corpus text from train, valid, test set. The method to generate this word embedding is fastText, using default n-gram setting max n-gram=6 and min n-gram=3, skip-gram=True, embedding_dim = 250.

Justification: There's several ways to generate semantic embedding, I choose fastText to train the original data set because there's some uncommon term in this dota2 corpus like 'gg', 'wp', which may not be included in popular pretrained word embedding models like glove-giga-word-25 etc. Another reason using this fastText is that in CONDA data set, there's many rows of sentences only contain one word, for word2vec it may make non-sense. FastText can split word into several parts and generate vector base on it, and for those uncommon terms, I wish fastText can extract some characters-based features, for example, 'ggwp' means 'good game well play', that's the typical latent feature which word2vec model can't extract.

3. Domain embedding.

In order to generate domain embedding, I find a dota2 in-game chat in Kaggle which called 'dota2_chat_messages.csv' [1], which contains corpus of in-game chat in dota2. I decide to use fastText model to generate this domain embedding using this external dota2 corpus, using default n-gram setting max n-gram=6 and min n-gram=3, skip-gram=True, embedding_dim = 250. Compare to word2vec embedding, it can catch characters-wise information by splitting word into several parts.

For this data set:

# match	# time	# slot	A text
0	1005.12122	9	ладно гг
0	1005.8544199999999	9	изи
0	1008.65372	9	од
0	1010.51992	9	ебаный
0	1013.91912	9	мусор на войде
0	1800.31402	9	мусор
0	1801.7188199999998	9	на войде
0	1802.98982	9	репорт
0	1808.40822	9	100%
1	-131.14018000000002	0	twitch.tv/rage_channel
1	-121.60481000000001	0	https://www.twitch.tv/rage_channel

As you can see there are many texts are in Russian or other languages not English, which may be not useful for our embedding.

So I traverse the text column and use python built-in function 'ord' to return the ASCII code of each characters in one row text, and create a function return text columns which only have character's ASCII code lower than 128, after that, we got a corpus consists of English corpus.

Justification: For this task, more corpus is always better, that's why I choose external dota2 chat corpus. As a dota2 player, I know that some words uncommon are not easy to generate. For example, 'gg' may be a common term in original CONDA which means 'good game' in dota2. But someone may type 'gggggg' or more 'g' instead, which have same meaning to 'gg'. Using large corpus can increase the showing chance or frequency of these uncommon representation, to well represent more uncommon term. And for solve this problem better, using fastText which can divide word into several piece like convert 'bed' into '-b', 'be', 'ed', 'd-'. In our situation, it will split 'ggggg' into many piece like '-g', 'gg' etc. Then 'ggggg' and 'gg' will have close word embedding since they all have factor 'gg' using 2-grams fastText.

3 Slot Filling/Tagging model

The main structure of this slot filling/tagging model is bilstm plus crf, just like lab9. In addition, add attention and stacked lstm layer mechanism.

Justification:

Stacked seq2seq (bilstm): 'RNNs are inherently deep in time, since their hidden state is a function of all previous hidden states. The question that inspired this paper was whether RNNs could also benefit from depth in space; that is from stacking multiple recurrent hidden layers on top of each other, just as feedforward layers are stacked in conventional deep networks.t' [5]. The paper shows that applying a stacking layer LSTM, beating benchmarks in some tasks, the depth of the LSTM network is also important and can increase the performance. I will test 1,2 and 3

stacking layers in my model and find the best number of stacking layers, maybe these stacking layers can extract more advanced and abstract features from the output of first LSTM layer, which can make prediction more precisely.

Attention:

As mentioned in *Attention is all you need*, we can see a result [6]

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Showing that using attention mechanism can achieves better performance than previous state-of-the art models. Attention can firstly calculate the similarity between the vector and get an softmax weight for each of these vectors as a new feature. Instead of directly use the BILSTM output, I decide to use an self-attention mechanism to calculate the attention between sequence factors of BILSTM output, and treat them as new BILSTM features for crf layers, expect it can get over the shortage of BILSTM which is vanishing gradient due to long sentence and get higher performance than original BILSTM features.

For calculation strategy, testing three kinds of method: scaled dot product, cosine, and dot-product.

The difference between scaled dot product and dot product is, scaled dot product divided by an standardized value, which can prevent some extreme value which is too large or small. As for Cosine, it just calculate the cosine similarity between two vectors.

Testing these three different strategy aim to find a best strategy for our task and dataset, because it can't say which strategy is the best one, just try and choose the best one to get higher performance for our model.

And since we test 1,2,3 number layers of LSTM, we can have different position of attention layer. I only test position layer in LSTM layer number = 2: Attention between 2 LSTM layer, and Attention after 2LSTM layer. The specification of attention make it can apply nearly every position of model. In *attention is all you need*, they even remove the LSTM structure and get higher performance. So, I expect different modification and implementation of position(included below) or number of attention layer(not included in this assignment) can get a higher performance.

CRF:

'Conditional random fields (CRFs) are graphical models that can leverage the structural dependencies between outputs to better model data with an underlying graph structure. '[7]

I will explain my justification of using this CRF layer using my own empirical result from my model.

Unnamed: 0	Model_name	T-F1
9	Bilstm+crf+Attention(scaled-d-a,1layer)+domain_emb	0.990526
19	Bilstm+Attention(scaled-d-a,1layer)+domain_emb	0.987528

You can see according to the above table, second row is the model without CRF layer, whose performance is slightly lower than the first one which having a CRF layer. It may because CRF layer can make some fixed rules to prevent some abnormal but possible dependency or entity word order in model.

4 Evaluation

4.1 Evaluation setup

Environment: Google Colab pro, library including pandas, sklearn, pytorch, spacy, genism

Hyper-parameter:

Input embedding dimension: Hidden dimension = 250 for all results

Learning rate: 0.0005 for all results

Optimizer: AdamW, weight_decay = 1e-4 for all results

Epoch: 2 for all results

4.2 Evaluation result

Model comparison:

Baseline model:

Model_name	T-F1	T-F1(O)	T-F1(T)	T-F1(P)	T-F1(SEPA)	T-F1(S)	T-F1(D)	T-F1(C)
Baseline(Bilstm+crf+glove25)	0.979882	0.983839	0.949341	0.995158	1.0	0.972511	0.813049	0.934729

My final model:

Model_name	T-F1	T-F1(O)	T-F1(T)	T-F1(P)	T-F1(SEPA)	T-F1(S)	T-F1(D)	T-F1(C)
Bilstm+crf+Attention(scaled-d-a,1layer)+domain_emb	0.990526	0.992309	0.963905	0.99582	0.999861	0.991217	0.921151	0.975976

Analysis:

Baseline model is Bilstm with crf with word embedding glove without attention, my best model is bilstm with crf with domain embedding with 1layer attention using scaled dot product. As you can see, the total F1 of baseline is 0.979882 and my model is 0.990526, it can be conclude that my model have better total performance in metric micro F1 with the help of scaled dot product attention layer, and domain embedding. Why these mechanisms make the result better? It may because the domain embedding using a huge dota2-related corpus which can more easily catch some uncommon words' features than glove 25, and attention layer can get over the issue caused by huge length of sequence, which lead to vanishing gradients, by fairly calculating all of the time step's similarity in a sequence.

We can see all the term f1 results except 'SEPA' of my final model are better than baseline. In my opinion [SEPA] is very easy to predict since all word in this class are the same '[SEPA]'. Maybe it's because my model is too complex compared to the baseline model, so in 2 epoch it can't converge very well.

Ablation study: (typo: sym->sem which means semantic)

Unnamed: 0	Model_name	T-F1
0	Baseline(Bilstm+crf+glove25)	0.979882
1	Bilstm+crf+glove25+domain_emb	0.990046
2	Bilstm+crf+glove25+syn_emb+sym_emb	0.980212
3	Bilstm+crf+glove25+syn_emb+domain_emb	0.989776
4	Bilstm+crf+domain_emb	0.988397
5	Bilstm+crf+sym_emb+domain_emb	0.990436
6	Bilstm+crf+syn_emb+domain_emb	0.989117
7	Bilstm+crf+Attention(scaled-d-a,1layer)+syn_emb	0.774900
8	Bilstm+crf+Attention(scaled-d-a,1layer)+sym_emb	0.986179
9	Bilstm+crf+Attention(scaled-d-a,1layer)+domain_emb	0.990526
10	Bilstm+crf+Attention(scaled-d-a,1layer)+syn_emb+sym_emb	0.986748
11	Bilstm+crf+Attention(scaled-d-a,1layer)+syn_emb+domain_emb	0.987917
12	Bilstm+crf+Attention(scaled-d-a,1layer)+sym_emb+domain_emb	0.989237
13	Bilstm+crf+Attention(scaled-d-a,1layer)+syn_emb+sym_emb+domain_emb	0.985729
14	Bilstm+crf+Attention(dot_product,1layer)+domain_emb	0.988517
15	Bilstm+crf+Attention(cos,1layer)+domain_emb	0.989746
16	Bilstm+crf+Attention(scaled-d-a,2layer,attention position after 2 lstm)+domain_emb	0.989507
17	Bilstm+crf+Attention(scaled-d-a,2layer,attention position between 2 lstm)+domain_emb	0.988367
18	Bilstm+crf+Attention(scaled-d-a,3layer,attention position after 3 lstm)+domain_emb	0.986418
19	Bilstm+Attention(scaled-d-a,1layer)+domain_emb	0.987528

The table above include all the results I have tried, and I will do ablation study base on it.

Different input embedding: (typo: sym->sem which means semantic)

Unnamed: 0	Model_name	T-F1
7	Bilstm+crf+Attention(scaled-d-a,1layer)+syn_emb	0.774900
8	Bilstm+crf+Attention(scaled-d-a,1layer)+sym_emb	0.986179
9	Bilstm+crf+Attention(scaled-d-a,1layer)+domain_emb	0.990526
10	Bilstm+crf+Attention(scaled-d-a,1layer)+syn_emb+sym_emb	0.986748
11	Bilstm+crf+Attention(scaled-d-a,1layer)+syn_emb+domain_emb	0.987917
12	Bilstm+crf+Attention(scaled-d-a,1layer)+sym_emb+domain_emb	0.989237
13	Bilstm+crf+Attention(scaled-d-a,1layer)+syn_emb+sym_emb+domain_emb	0.985729

Justification and result explanation

I use these 7 combinations of embeddings, because I think this concatenate strategy can give model chance to utilize information from different embedding, to get a 1+1>2 performance. For example, if I combine domain embedding and semantic embedding, some word not appeared frequently in domain corpus may appeared many times in semantic embedding, it expected to get higher performance.

In these 7 combinations of embedding, the trend can be summarized like for single embedding, syntactic is the worst one, semantic is better, and domain embedding is the best one. It can be conclude that adding or concatenating more embedding seems not always useful, they all perform lower than single domain embedding. It may because it's difficult for model to catch the relationship between 2 or 3 types of embedding since epoch is only 2. Domain

embedding perform well may because it have a larger corpus of dota2 compare to the original CONDA data set, so it can represent meaning or relationship between words more precisely. As the reason mentioned above, I will use domain embedding as a component to my final model.

Different attention strategy:

	Unnamed: 0	Model_name	T-F1
9	9	Bilstm+crf+Attention(scaled-d-a,1layer)+domain_emb	0.990526
14	14	Bilstm+crf+Attention(dot_product,1layer)+domain_emb	0.988517
15	15	Bilstm+crf+Attention(cos,1layer)+domain_emb	0.989746

Justification and result explanation:

I just use the best model components combination from above embedding ablation part which is domain embedding to test this different attention strategy part. Compare to cos and scaled dot product and dot product, you can see that they are quite close. Reason why scaled dot product perform better than dot product may because it have standardize term, to prevent output some extreme large value for example. And cos strategy perform better than dot product may because 'mode' information is not important in this particular task. Or because cosine strategy have a output range between [-1,1], dot product get some extreme large value due to not having a standardized term. As the reason mentioned above, I will use scaled dot product as a component to my final model.

Different stacked layer numbers and attention position:

Unnamed: 0	Model_name	T-F1
9	Bilstm+crf+Attention(scaled-d-a,1layer)+domain_emb	0.990526
16	Bilstm+crf+Attention(scaled-d-a,2layer,attention position after 2 lstm)+domain_emb	0.989507
17	Bilstm+crf+Attention(scaled-d-a,2layer,attention position between 2 lstm)+domain_emb	0.988367
18	Bilstm+crf+Attention(scaled-d-a,3layer,attention position after 3 lstm)+domain_emb	0.986418

Justification and result explanation:

I use the best combination of components mentioned above to test this different stacking layer numbers and attention positions.

If you compare row1 and row2, you can get the conclusion that one stacked layer lstm with attention perform better than the 2 stacked layers lstm. If you compare row 2 and row4, you can conclude that 2layer perform better than 3 stacked layers lstm. Seems like add more layer to this lstm attention model doesn't make any improvement. That may because even though the model is more complex, it can't converge to a better result if you train it only 2 epoch, I mean, it may suffer from under fitting. If we add train epochs, more layers lstm may perform better since they can get more information with a deeper network.

If you compare row2 and row3, you can get the result that in the same condition of having two stacked lstm layers, attention position after lstm perform slightly better than the attention position, not very significant. I prefer to believe in this task, change the position of attention does not make any difference.

Thus, according to the micro f1 performance, we decide to add only 1 stacked layer lstm with attention after it to the final model.

With/without CRF:

Unnamed: 0	Model_name	T-F1
9	Bilstm+crf+Attention(scaled-d-a,1layer)+domain_emb	0.990526
19	Bilstm+Attention(scaled-d-a,1layer)+domain_emb	0.987528

Justification and result explanation:

We can conclude that the first row model performs slightly better than second row, which means CRF layer can get 0.3% improvement in our model. Because we only use the current best combination of components mentioned above and the improvement is quite small, it may not be appropriate to say this CRF layer really helps model to predict well. The reason why model having CRF perform better may because CRF layer can make some fixed rules to prevent some abnormal but possible dependency or entity word order in our model using LSTM output features, it always make sentence more reasonable. But sometimes people may say some sentence having not very reasonable grammar or dependency structure especially in a game, it also can explain why improvement is very small.

We decide to add CRF component in our final model.

References

1. GOSU.AI Dota 2 Game Chats, https://www.kaggle.com/datasets/romovpa/gosuai-dota-2-game-chats
2. Pytorch library, https://pytorch.org/
3. Sklearn library, https://scikit-learn.org/stable/
4. Attention strategy from github, https://github.com/ROBINADC/BiGRU-CRF-with-Attention-for-NER
5. Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton Speech Recognition With Deep Recurrent Neural Networks, 2013
6. Vaswani, Ashish, et al. "Attention is all you need."Advances in Neural Information Processing Systems. 2017.
7. Exploring Conditional Random Fields for NLP Applications https://hyperscience.com/tech-blog/exploring-crfs-for-nlp-applications/
8.CONDA https://github.com/usydnlp/CONDA