Application of Bi-LSTM CRF with Attention Model in In-game Toxicity Detection Task

Siwei Luo[1] and Zirong Guo[2]

[1] University of Sydney, NSW 2006, Australia
silu5841@uni.sydney.edu.au
[2]University of Sydney, NSW 2006, Australia
zguo3350@uni.sydney.edu.au

1 Data preprocessing

In this experiment, we first segment the text according to spaces, and then use the case folding method to convert all tokens to lowercase.

2 Input Embedding

We trained three different types of embeddings using POS tags information, training data, and additional dota2-related data. By comparing the performance of different embeddings on the baseline model, the best word embedding is selected. The experimental results and justification are detailed in section 2.4.2.

2.1 Syntactic textual input embedding (POS)

In this section, we use the part-of-speech information to generate dense word embeddings based on the Bi-LSTM CRF model.

The development of the POS tagging task has gone through three main stages. The first is the rule-based algorithm, followed by the probability model based on HMM and CRF, and finally the algorithm based on deep learning has emerged. The disadvantage of the Rule-based algorithm is that it is highly subjective, requires a lot of human work, and has ambiguity problems; its advantage is that the algorithm runs faster. The probabilistic POS tagging model is represented by the CRF algorithm, which solves the label bias problem of the MEMM algorithm and does not have strict independence assumptions; but its disadvantage is the long running time of the algorithm. The main idea of the deep learning-based algorithm is to use the Seq2Seq architecture to construct an N2N model, with word embedding as input and POS Tags as output.

In this experiment, we tested three different POS tagging algorithms, namely rule-based algorithm, lookup table-based algorithm and HMM probability model. As shown in Figure 1, to generate dense word embedding, we first use a POS tagging algorithm to label the training, validation and test data; then use one-hot labeled words as input and POS tags as output to train a Bi-LSTM CRF model; finally, the embedding layer of the Bi-LSTM CRF model is extracted as the Syntactic textual input embedding generated with POS information.

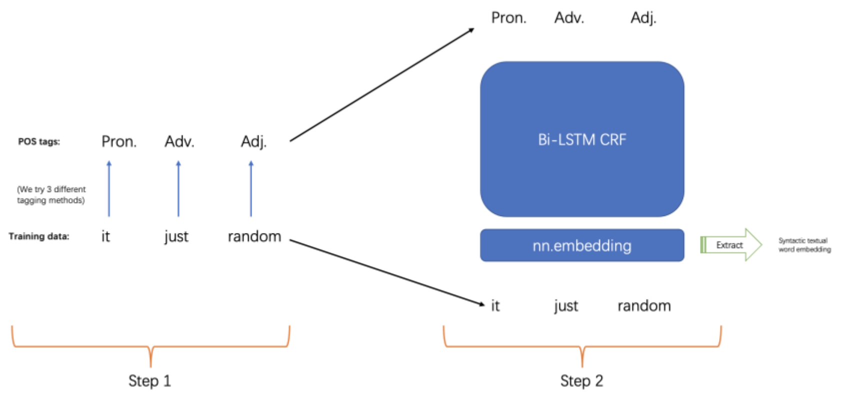

Fig.1. Syntactic textual embedding training process

We believe that POS information should be of great help to this In-game Toxicity Detection task because the label of this task has a certain correlation with POS information. By studying the training data, we found that tokens marked as 'Toxicity' are mostly nouns or verbs; tokens marked as 'Character' are mostly nouns (game characters); tokens marked as 'Pronoun' are pronouns; modal will be marked as 'Game slang'. Therefore, the model performance can be improved by learning the correlation between POS information and tags.

The reason why we choose dense word embedding method instead of one-hot embedding method when encoding POS information is: the number of POS tags may be large; sparse vector representation cannot reflect the relationship between different POS.

2.2 Semantic textual input embedding

In this section, we use the FastText-Skip-gram model and the Char2Vec model to train Semantic textual input embeddings using the data from training, validation, and test set.

The main idea of the Skip-gram model is to use the central word to predict the context words, so as to map the words to a high-dimensional space, so that semantically similar words are in similar positions in the space. The FastText-Skip-gram model uses N-gram mode to segment words, which alleviates the OOV issue. We think this model is very helpful for this task, because players often make typos or use uncommon words when communicating in the game. The classic Word2Vec-Skip-gram model may perform worse due to heavy OOV issue. The Char2Vec model is another solution to the OOV issue and misspelling problem. The model represents a sequence of symbols of arbitrary length as a fixed-length vector, and the spelling similarity between words is represented by the distance measure between the vectors.

2.3 Domain textual input embedding

In this section, we make domain embeddings combining information gathered from the web about Dota2 and textual statistics.

We collected additional data from 8 different dimensions of Dota2 through Dota2 Wikipedia[1], game forums[2,3,4], and the official website[5]. These dimensions are hero name, item name, game mechanics, world mechanics, character state, game term, dirty words, and game name. There are many records of 2 or 3 tokens in hero names and item names, so we divide the data of these two dimensions according to spaces, and remove prepositions and punctuation marks. Through studying the data, we found that the data we gathered are related to the data used in this experiment. For example, hero names will be marked as 'Character', and item names will be marked as 'Dota-specific' . Therefore, we believe that using this information to make embeddings will be of great help to the In-game Toxicity Detection task.

We first use the idea of Word2Vec-CBOW to encode words and generate a dense word embedding. The specific method is: for each dimension, the data is used as input, and the dimension name is used as output to make training data; the Word2Vec-CBOW model is used for modeling and word embedding is obtained. As shown in Figure 2, we believe that the model maps words to a high-dimensional space, and that data with the same dimension name will form a convex polyhedron in the space, and the center of the convex polyhedron is the dimension name. Meanwhile, there is a certain distance between different convex polyhedron.

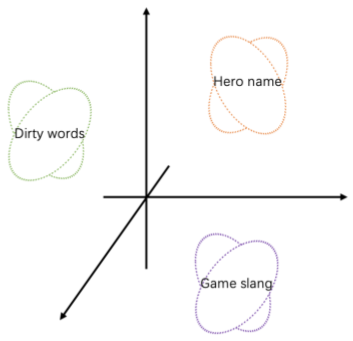

Fig.2. Domain embedding in high-dimensional space

Then, we use the one-hot encoding method to make a domain embedding based on textual statistics and concatenate these two domain embeddings together as our final domain embedding. The statistics we use are: is it '[sepa]', does it contain numbers, does it contain punctuation, is it less than 4 characters, and is it greater than 8. We think these statistics will also be helpful in improving model performance, as studying the data we found that tokens that are too long and tokens that contain numbers or punctuation are classified as 'Others'.

3 Slot Filling/Tagging model

3.1 Stacked Seq2Seq model (Bi-LSTM)

Bi-LSTM is a classic and efficient Seq2Seq model, which is widely used in various time series models. Increasing the complexity of the model by stacking Bi-LSTM can theoretically improve the ability of the model to fit data, that is, increase the memory ability of the model, but the possible disadvantage is that the model needs more data for training, and it is prone to overfitting.

In this section, we choose the Bi-LSTM CRF model to study the effect of stacking Bi-LSTM layers on model performance. We chose the number of stacked layers to be 2, 3, and 4; and tested the effect of different learning rates and weight decay rates on model performance. The experimental results and justification are detailed in section 2.4.3.

3.2 Self-Attention

The Self-Attention mechanism (as shown in Figure 3) is widely used in various fields of deep learning because it gives the model the ability to discriminate. For example, in machine translation applications, the Attention mechanism assigns different weights to each word in a sentence, making the learning of neural network models more flexible.

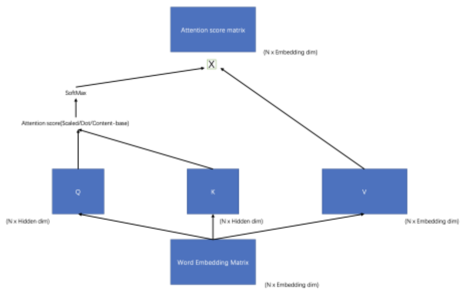

Fig.3. Self-Attention mechanism

In this section, we tested three attention score calculation methods, namely Scaled Dot Product, Dot Product and Content-base, in two different positions (as shown in Figure 4) which are between Embedding layer and Bi-LSTM layer and between Bi-

LSTM layer and Dense layer. The experimental results and justification are detailed in section 2.4.4.

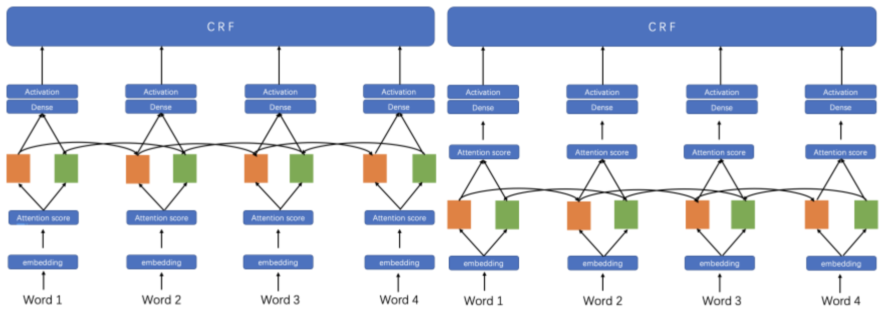

Fig.4. Adding Attention mechanism in different position

3.2.1 Scaled Dot Product

The formula of this method is:

$$score(q_t, k_i) = \frac{q_t k_i^T}{\sqrt[2]{n}}$$

Where n is the dimension of the source hidden state.

3.2.2 Dot Product

The formula of this method is:

$$score(q_t, k_i) = q_t k_i^T$$

3.2.3 Content-base

The formula of this method is:

$$score(q_t, k_i) = cosine[q_t, k_i]$$

3.3 CRF attachment

The CRF model is a conditional probability model that is often used to label or analyze sequence data. Compared with the MEMM model, the CRF model considers global features and thus solves the problem of label bias. Compared with HMM, the CRF model does not have strict independence assumptions.

To study the contribution of the CRF algorithm to the problem of improving sequence labeling, we test the performance of the Bi-LSTM model and the Bi-LSTM with CRF model respectively. The experimental results and justification are detailed in section 2.4.5.

4 Evaluation

4.1 Evaluation setup

We used epoch=2 for all experiments. The hidden size for Bi-LSTM is set to 50 for all experiments. In 4.2.1, 4.2.4 and 4.2.5, we set word-embedding size to 115 (because we use the concatenated embedding matrix trained by ourselves), set the number of Bi-LSTM layers to 1. In section 4.2.2, to compare different word embeddings more reasonably, we froze the embedding matrix while training and learning rate is still set to 0.01. In section 4.2.3, we tried learning rate as 0.006, 0.01, 0.05 and 0.09 in various number of layers.

4.2 Evaluation result

4.2.1 Performance Comparison

Our best model used Bi-LSTM + CRF + Scaled Product Attention, from the performance results, our best model achieves a great improvement in all evaluation indicators compared to the baseline model.

It is worth noting that for the classification result about label 'D', the baseline model achieved 71% F1 score, while the best model we proposed achieved 90% F1 score. This shows that our model comprehensively utilizes information from various dimensions of word embedding; also, the CRF layer considers and utilizes the dependency information between labels.

T-F1 refers to the micro F1 score of labels T, P, S, D, C.

	T-F1	**T-F1(T)**	**T-F1(S)**	**T-F1(C)**	**T-F1(D)**	**T-F1(P)**	**T-F1(O)**
Baseline model	94.74%	91%	97%	90%	**71%**	98%	97%
Best model	97.14%	97%	99%	98%	**90%**	100%	99%

Table 1: Comparison between baseline and best model

4.2.2 Ablation study – different input embedding model

We tried the method of solely using syntactic word embedding and found that the results of various methods in syntactic (i.e., POS-RE, POS-Look-up-table and POS-HMM) word embedding are not very different. Therefore, we chose the POS-RE, which is a little higher than the other two, as syntactic word-embedding method.

In semantic word embedding, our first intuition is to use Skip-gram/C-BOW or FastText to generate word embedding matrix from scratch and after our experiments, we found that the results of these three methods are similar, so we chose the FastText as semantic word-embedding. However, after have a specific look on the dataset, we found that there are lots of abbreviations in this in-game chatting dataset (e.g., 'wtf','ez','rofl') and our intuition is that the performance will be improved if our model can understand the letter-level information from each word. Furthermore, word2vec model is sensitive to spelling errors (e.g., 'globl' should be 'global') which are very common in this in-game chatting dataset. We were thinking that only applying word-level word2vec might not enough for this task, so we include a letter-level word2vec model named Char2vec to let the embedding model try to understand the letter-level information. We finally concatenated word-level and letter-level word embedding together, which performed best.

The solely domain embedding gets the worst performance because the vocabulary size of the corpus we use to train the domain embedding is much smaller than the vocabulary size of the corpus for this assignment. In this case, many tokens in our training, validation, and test sets are labeled as 'UNKNOWN', so the domain embedding does not perform well. But we still believe that adding domain embedding as supplementary information to semantic embedding will bring an improvement in the final model performance.

We were thinking that solely using one type (i.e., syntactic, semantic or domain) of word embedding might not enough to precisely describe a word. Therefore, we tried concatenate different types together and then the result shows that the combination of POS+Char2Vec+FastText+Domain embedding performed best, so we use this matrix as our best model word embedding matrix.

T-F1(mean) refers to the micro F1 score of all labels, including T, P, S, D, C, O and SEPA.

Input Embedding Type	**Method**	**T-F1(mean)**
Syntactic textual input embedding	POS-Look-Up-Table	95.39%
	POS-RE	**95.63%**
	POS-HMM	95.39%
Semantic textual input embedding	FastText (Skip gram)	**90.08%**
	Char2Vec	88.49%
Domain textual input embedding	CBOW (domain knowledge) + statistical features	75.97%
Concatenation	POS-RE + FastText+ Domain	97.55%
	POS-RE+Char2Vec+Domain	97.17%
	POS-RE+FastText+Char2Vec+Domain	**98.11%**

Table 2: Comparison among different Word Embedding

4.2.3 Ablation study – different stacked layers

We tried 4 different number of Bi-LSTM layers (i.e., 1,2,3,4) for ablation study and we found that 1-layer Bi-LSTM performs best. The worst result of 4-layer Bi-LSTM is not expected, our first intuition is that it may be that the number of layers increases, which leads to too many parameters, which makes the model complicated and requires a higher learning rate to fit it. Thus, we tried various learning rates for different number of layers and found that as the number of layers increases, the increase in learning rate is more helpful for model fitting which confirms our intuition. But the results all show that 1-layer is the best which is the opposite of what we predicted. This may be because the task is not so complicated that requires 2 or 3 or 4 layers to fit.

T-F1(mean) refers to the micro F1 score of all labels, including T, P, S, D, C, O and SEPA.

Stacked Layers	Learning Rate	Weight Decay	T-F1(mean)	T-F1(O)	T-F1(T)	T-F1(P)	T-F1(SEPA)	T-F1(S)	T-F1(D)	T-F1(C)
2	0.09	1e-4	99.30%	100%	98%	100%	100%	99%	93%	97%
	0.05	1e-4	99.33%	100%	97%	100%	100%	99%	92%	98%
		5e-4	**99.37%**	100%	98%	100%	100%	99%	94%	98%
		7e-4	99.15%	100%	97%	100%	100%	99%	87%	95%
		9e-4	99.25%	100%	97%	100%	100%	99%	91%	97%
	0.01	1e-4	98.77%	99%	96%	99%	100%	99%	83%	97%
	0.006	1e-4	98.74%	99%	95%	99%	100%	99%	81%	97%
3	0.09	5e-4	98.64%	99%	96%	98%	100%	99%	90%	97%
	0.05	1e-4	**99.04%**	99%	96%	100%	100%	99%	88%	97%
	0.01	1e-4	98.01%	99%	93%	99%	100%	97%	51%	96%
	0.006	1e-4	96.47%	99%	77%	99%	100%	95%	16%	94%
4	0.09	1e-4	**98.58%**	99%	96%	98%	100%	98%	75%	97%
	0.05	1e-4	98.52%	99%	96%	99%	100%	98%	72%	97%
	0.01	1e-4	96.43%	98%	85%	98%	100%	94%	43%	91%
	0.006	1e-4	92.15%	97%	59%	97%	99%	86%	0	66%

Table 3: Comparison among different stacked Bi-LSTM layers

4.2.4 Ablation study – different attention strategy

In this section, we tried tree different attention score calculation methods: Scaled Dot Product, Dot Product and Content-base. The result shows that Scaled Dot Product got the best result. Our intuition is that the Dot Product without the scaled process might cause gradient exploding for backward propagation and the scaled calculation can handle this problem.

T-F1(mean) refers to the micro F1 score of all labels, including T, P, S, D, C, O and SEPA.

Position	Strategy	Learning Rate	Weight Decay	T-F1(mean)	T-F1(O)	T-F1(T)	T-F1(P)	T-F1(SEPA)	T-F1(S)	T-F1(D)	T-F1(C)
Between embedding layer & Bi-LSTM layer	Scaled Dot Product	0.01	1e-4	98.82%	99%	97%	99%	100%	99%	81%	97%
		0.02	1e-4	**99.08%**	99%	97%	100%	100%	99%	91%	98%
		0.03	1e-4	73.18%	83%	43%	53%	75%	64%	5%	34%
		0.05	1e-4	63.74%	74%	21%	45%	54%	58%	0	18%
	Dot Product	0.005	1e-4	**98.18%**	99%	95%	99%	100%	97%	67%	97%
		0.003	1e-4	97.01%	98%	91%	98%	100%	96%	23%	95%
		0.001	1e-4	74.31%	82%	0	14%	100%	71%	0	1%
	Content-base	0.09	1e-4	95.53%	98%	81%	97%	98%	94%	54%	86%
		0.05	1e-4	97.58%	99%	91%	98%	99%	97%	69%	92%
		0.01	1e-4	**97.63%**	99%	91%	99%	100%	97%	69%	92%
		0.006	1e-4	96.29%	98%	86%	98%	99%	94%	14%	91%
Between Bi-LSTM & Dense layer	Scaled Dot Product	0.01	1e-4	**98.97%**	-	-	-	-	-	-	-
	Dot Product	0.01	1e-4	98.68%	-	-	-	-	-	-	-

Table 4: Comparison among different inserting position and calculating method of attention

4.2.5 Ablation study – with/without CRF

After removing CRF, the result became lower. Our intuition is that the CRF helps the model to handle the dependency between predicted tags. If we removed the CRF, the Bi-LSTM model is only able to learn the contextual dependency or relationships between features of each sentence but not contextual dependency between tags. However, the contextual dependency between tags is critical for current tasks.

The difference between with or without CRF is huge. Our intuition is that the output of Bi-LSTM relies heavily on the CRF layer to predict the correct path of tags. Therefore, after removing CRF, the result will crash to 0.

T-F1(mean) refers to the micro F1 score of all labels, including T, P, S, D, C, O and SEPA.

Model	**Learning Rate**	**Weight Decay**	**T-F1(mean)**	**T-F1(O)**	**T-F1(T)**	**T-F1(P)**	**T-F1(SEPA)**	**T-F1(S)**	**T-F1(D)**	**T-F1(C)**
With CRF	0.02	1e-4	99%	99%	97%	100%	100%	99%	88%	97%
Without CRF	0.02	1e-4	1%	0%	1%	1%	0%	0%	1%	0%

Table 5: Comparison between with/without CRF

References

1. Dota 2 Wiki. (2022). Retrieved 5 June 2022, from https://dota2.fandom.com/wiki/Dota_2_Wiki
2. List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/en at master · LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words. (2022). Retrieved 5 June 2022, from https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/blob/master/en
3. Alternate Hero Names. (2022). Retrieved 5 June 2022, from https://notenoughdota.wordpress.com/alternate-hero-names/
4. Dota 2 Glossary - Dota Vocabulary [All Words Explained]. (2022). Retrieved 5 June 2022, from https://dota2freaks.com/glossary/
5. Dota 2. (2022). Retrieved 5 June 2022, from https://www.dota2.com/home