In-game Toxicity Detection

Yexin Mao Shuxin Huang

University of Sydney, NSW 2006, Australia
ymao5456@uni.sydney.edu.au
shua8778@uni.sydney.edu.au

1 Data Preprocessing

To preprocess the data, word tokenization is applied in order to divide the data into a set of meaningful pieces. All words are then changed into lowercase, which helps get rid of duplicate words in different cases (Han, 2022, pp. 59–67). In addition, words and tags in the dataset are converted into indexes so they are consistent with the input and output format of the slot filling/tagging model.

2 Input Embedding

The input word vector is generated by using three different aspects of features, respectively syntactic textual feature embedding, semantic textual feature embedding and domain feature embedding. After generating features in these three aspects, different concatenations of features are applied and tested.

2.1 Syntactic Textual Feature Embedding

Part of Speech (PoS) tag information is used to generate syntactic feature embedding, and it is obtained from “en_core_web_sm”, an available trained pipeline for English provided by SpaCy. Each word in the word list is assigned a corresponding PoS tag. Then, the PoS tags are converted into one-hot encodings of length 38, which is the same as the number of all different PoS tags for words in the dataset.

The reason for choosing PoS tag information to embed syntactic textual features rather than dependency parsing is that the given dataset is collected from in-game chats, so the content tends to be short and may not be a complete sentence. According to Deng and Liu (2018), dependency parsing is investigating syntactic relations between words that may not be very helpful in this case, where PoS tags indicate the properties of each word itself. Since the other two feature embeddings are word embeddings, treating syntactic embeddings as word embeddings as well will facilitate the combination of embeddings. That is why each word has only one PoS tag and they are converted into one-hot encoding. Although this approach may tag the wrong POS information due to ignoring the context, it is acceptably accurate because of the previously mentioned dataset characteristics (short and non-sentential).

2.2 **Semantic Textual Feature Embedding**

To create a semantic feature embedding, the word embedding with Gensim is loaded, specifically, the Global Vectors for Word Representation (GloVe) embedding trained on Twitter data “glove-twitter-200” is used.

Since the given dataset is the content of in-game chat, its phrases should be relatively simple and informal, which is most similar to the content of twitter. As shown in the GloVe paper (Pennington et al.,2014), vectors of length 100–300 dimensions shall provide the most useful results. Thus, "glove-twitter-200" is chosen for the word embedding. The reason why not using ELMO to create the semantic textual feature embedding is that fine tuning for a pre-trained model is not allowed for this task, but fine-tuning plays an important role for ELMO as it can lead to significant drops in perplexity and increase in task performance (Peters et al., 2018). Therefore, ELMO may not work well without fine-tuning.

2.3 **Domain Feature Embedding**

Our own new feature embedding to solve this in-game chat word slot filling is a Gensim FastText Skip-Gram model representing the dota-game-domain, which uses a domain-related dataset that also contains content in chat during the dota game. Then, a domain feature embedding for the given dataset can be obtained. The dimension of embedding is set to 100 and the window size is set to 2.

Since the additional dataset is related to dota and chat content, it shares the domain with the given dataset, and thus the model is able to produce domain feature embedding. FastText can handle the Out-of-Vocabulary issue by breaking words into several sub-words, which is an improvement compared with Word2Vec. This is helpful, in particular for the given dataset, because words used in chat can sometimes be abbreviated or simplified that are not included in the vocabulary, and in this case FastText is still able to deal with these words. Using Skip-Gram instead of CBOW is because Skip-Gram works well with small datasets, and can better represent less frequent words (Mikolov et al., 2013), which is suitable for the given dataset. Similar to the reason of choosing the dimension for semantic textual feature embedding, length 100-300 is reasonable for the domain feature embedding. After some testing, 100-dimensional performs the best. Because of the shorter sentence length in the given dataset, window size is just set as 2.

2.4 Concatenation of Different Feature Embeddings

An embedding matrix is generated for each type of feature embedding because it simplifies the process of making a concatenated embedding matrix. The selected feature embeddings are the semantic feature embedding and the domain feature embedding. This is because semantic feature embedding contains general information for each word while domain feature embedding contains their specific information in a related field. The combination of these two features provides a more complete representation of the meaning of words. The reason why syntactic feature embedding is not included is that it is formatted as one-hot encoding, which would lose a lot of information but increase computation cost (Han, 2022, pp.49).

3 Slot Filling/Tagging Model

Our best model is a 1-layer bidirectional GRU with CRF. In this section, we introduce different settings on stacking layers, attention and CRF as well as provide some justification.

3.1 Stacking Layers

Since we are using the RNN-based seq2seq framework (Bi-GRU), we test the effect of stacking layers by adding more bidirectional layers to our model. The layer sizes we choose are 1-layer, 2-layer and 3-layer. To change the layers in the model, we need to not only change the number of layer, but also change the hidden layer initialization function correspondingly.

The reason why we decide to test how many layers can be stacked for this model is that several layers tend to improve the model. In the multi-layer model, each layer processes parts of the task, and then passes it on to the next (Hermans, 2013). This in turn can make the result more accurate. However, there are no rules on the choice of hidden layers and too many layers will lead to overfitting (Goodfellow, 2016). Thus, it is necessary to test several layers and therefore get the best decision of the number of layers.

3.2 Attention

3.2.1 Attention Position

We apply attention at the different positions in our 2-layer bidirectional GRU. The first position we choose is the output of layer-1. In this setting, we first apply attention mechanism to the output of layer-1. Then the new outputs are fed into the second layer of GRU as input. The second position we choose is the output of layer-2. In this setting, we first feed the embeddings into a 2-layer bidirectional GRU. Then, we apply attention to its outputs, which are fed into CRF later.

3.2.2 Attention Score Calculation

The attention score calculation methods we use are cosine product, dot product and scaled dot product. The formulas are as follows (Luong et al., 2015; Vaswani et al., 2017):

Cosine product: score(s,h) = $\frac{s*h}{||s||*||h||}$

Dot product: score(s,h) = $s * h$

Scaled dot product: score(s,h) = $\frac{s*h}{\sqrt{n}}$

Only by calculating attention score using different methods can we decide which calculation method is the most suitable for our model.

3.3 **Conditional Random Field (CRF)**

In our model, we implement the Viterbi algorithm in order to add a CRF layer. The CRF model addresses the labeling bias issue and eliminates two unreasonable hypotheses in HMM. It is proven to improve the performance by adding CRF to the sequence model (Han, 2022, pp. 91). This is why implementing the CRF is a necessary attempt when building the model. However, CRF is computationally complex at the training stage and it makes it difficult to re-train the model when new data is available, so whether using CRF in the final model depends on the model performance as well as the computation cost.

4 Evaluation

4.1 **Evaluation Setup**

4.1.1 Dataset

The dataset used is part of the CONDA Dataset (a CONtextual Dual-Annotated dataset for in-game toxicity understanding and detection) from the University of Sydney NLP Group, which focuses on in-game word slot tagging(filling) from in-game chat. Three files (train, val, test) are provided, having 26078, 8705, 500 samples respectively. The train and val sets contain two columns, one for chat sentences and another for slot types. The test set has just one column, without slot types. There are six distinct slot labels: T (Toxicity), C (Character), D (Dota-specific), S (game Slang), P (Pronoun) and O (Other). The additional dataset used in domain feature embedding is obtained from kaggle

(https://www.kaggle.com/datasets/devinanzelmo/dota-2-matches?select=chat.csv), containing 1439474 samples with 5 columns(match_id, key, slot, time, unit). Only column "key" (chat content) is used to train the model.

4.1.2 Baseline

The baseline slot filling/tagging model is from the Lab 09 (Bi-LSTM CRF section). It is a 1-layer Bi-LSTM with CRF without Attention. The input embedding is combination of semantic textual feature embedding and domain feature embedding.

4.1.3 Implementation Details

For all slot filling/tagging models, epoch = 2, learning rate = 0.01, optimiser = SGD, hidden dimension = 100, weight decay=0.0001. Input embedding dimensions for syntactic feature embedding, semantic feature embedding and domain feature embedding are 38, 200, 100. All the experiments are conducted on Colab, with 12GB NVIDIA Tesla K80 GPU and with CUDA 11.1.

4.2 **Evaluation Result**

4.2.1 Performance Comparison

Model	T-F1(T)	T-F1(S)	T-F1(C)	T-F1(D)	T-F1(P)	T-F1(O)	T-F1
baseline	0.976438	0.994115	0.984088	0.954198	0.999111	0.994955	0.994094
best_model	0.975795	0.99487	0.983237	0.954315	0.999111	0.9949	0.994064

Table 1. Performance Comparison

The difference between our model and the baseline model is the use of GRU or LSTM. As we can see from the Table 1, the performance of these two models is similar. However, the training of our model is faster than the baseline. This is because GRU is less complex than LSTM because it has less number of gates (Chung, 2014). Since our model is the best compared to other settings by stacking layers, using attention etc, another interesting point is that the model architecture or complexity seems not so important in this task. Both the baseline or our simple model can achieve very good results as long as we have good input embedding.

4.2.2 Ablation Study - different input embedding model

Model	T-F1(T)	T-F1(S)	T-F1(C)	T-F1(D)	T-F1(P)	T-F1(O)	T-F1
sementic	0.973214	0.989896	0.972443	0.901554	0.99759	0.991693	0.990316
domain	0.971546	0.995324	0.98564	0.945545	0.99873	0.995289	0.994094
sementic_domain_syntactic	0.974341	0.994876	0.982606	0.954082	0.999365	0.994717	0.993884
sementic_domain	0.975795	0.99487	0.983237	0.954315	0.999111	0.9949	0.994064

Table 2. Different Input Embedding Model

As we can see from the Table 2, the semantic embedding alone has the worst performance. It is because the semantic embedding is generated using twitter data. Although tweets are relatively similar to spoken language, there are still differences from the chat in games, for example, the use of game slang. However, if we incorporate domain feature into our embedding, the F1 score becomes high. The domain feature itself achieves a slightly higher score compared to the combination of domain and semantic features. However, we still choose the combinatination in our model because the domain embedding is trained on a small domain-related dataset, which may not be reliable by itself only. By adding the syntactic feature into the embedding degenerates the performance a bit. We hypothesize that this is because the input sentence is too short that the syntactic information is not powerful enough. Also, the use of one-hot encoding for the syntactic embedding limits the information it contains (Han, 2022, pp.49).

4.2.3 Ablation Study - different attention strategy

Attention Position

Model	T-F1
cosine_attention_after_layer_2	0.989716
cosine_attention_after_layer_1	0.988427

Table 3. Different Attention Position

The position we apply attention is after the first layer of GRU or the second layer of GRU. By inserting attention into the stacked GRU layers performs worse than appending it after all the GRU layers.

Attention Score Calculation

Model	T-F1
cosine	0.986958
dot	0.83909
scaled_dot	0.992295

Table 4. Different Attention Score Calculation

The scaled dot product achieves best performance in our model while the cosine product is competitive. However, the F1 score of dot product is very bad compared to other two methods. This is hypothesized in (Vaswani et al., 2017) that for large embedding dimension, the dot products grow too large in magnitude, which pushes

the softmax function into regions with small gradients.

4.2.4 Ablation Study - different Stacked layer or # of encoder/decoder strategy

Model	T-F1
1_layer	0.994064
2_layer	0.994004
3_layer	0.993554

Table 5. Different Stacked Layers

It can be seen from the Table 5 that using 1-layer GRU or 2-layer GRU has the same performance. We choose the 1-layer one in our model because of the training time that the 2-layer GRU runs much slower than its 1-layer counterpart. However, it is also reasonable to choose the 2-layer one as it has more parameters to produce better results if not overfitting. The F1 score of the 3-layer GRU becomes lower, which is assumed that the model starts overfitting (Goodfellow, 2016).

4.2.5 Ablation Study - with/without CRF

Model	T-F1
use_crf	0.994064
no_crf	0.993434

Table 6. With or Without CRF

According to Table 6, the model with CRF has the F1 score of 0.9940, which is a bit higher than the one without CRF, whose F1 score is 0.9934. This is reasonable since CRF makes use of neighbour tag information in predicting current tags (Han, 2022, pp. 91). However, the improvement is not significant, especially considering the computation cost for CRF. This is because sentences in the given dataset are relatively short, and in some cases, the content only consists of one word. As a result, the effect of neighbour tags cannot be obvious.

References

1. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555.*
2. Deng, L., & Liu, Y. (2018). Deep Learning in Natural Language Processing [E-book]. In *4.2.2 POS Tagging, 4.2.3 Syntactic Parsing* (p. 81). Springer Publishing.
3. Goodfellow, I., Bengio, Y.,, Courville, A. (2016). Deep Learning. *MIT Press*. ISBN: 9780262035613
4. Han, C.(2022). Lecture 1: Introduction to the NLP. *COMP5046 Natural Language Processing*. 49
5. Han, C.(2022). Lecture 5: Language Fundamental. *COMP5046 Natural Language Processing*. 59-67
6. Han, C.(2022). Lecture 6: Part of Speech Tagging. *COMP5046 Natural Language Processing*. 91
7. Han, C.(2022). Lecture 10: Advanced NLP: Attention and Reading Comprehension. *COMP5046 Natural Language Processing*. 43
8. Hermans, M., & Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. *Advances in neural information processing systems*, 26.
9. Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025.*
10. Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. (2018). Deep contextualized word representations. In *NAACL.*
11. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*
12. Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).
13. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems,* 30.