TDBot at SemEval-2019 Task 3: Context Aware Emotion Detection Using A Conditioned Classification Approach

Sourabh Maity
Teradata India Pvt. Ltd.
Hyderabad, India
sourabh.maity@teradata.com

Abstract

This paper presents the system developed to detect the contextual emotion (*SemEval19 Task 3* (Chatterjee et al., 2019)) from conversational dialogue. The system models the fact that emotion of a dialogue depends on the context of the conversation and not independent. It uses multiple layers in the deep learning model where each layer *bootstraps* with the context of what has already been said in the conversation.

1 Introduction

Over the years, we are getting more and more comfortable in text based conversations over the web, leading to increased interest in emotion analysis. Such a conversation is no longer limited between humans, it is now mainstream to use chat bots at various domains, e.g., customer care, HR management, virtual doctor etc.

Needless to say that in a conversation human emotions needed to be handled with care and empathy. Due to this, the task of emotion detection is very important when our aim to use chat bots and voice assistants more effectively. It is more difficult when the conversation is text based, lack of facial expressions and voice modulations make detecting emotions in text a challenging problem (Gupta et al., 2017).

In this SemEval19 Task 3 there were total three turns of dialogues; *turn1* and *turn3* were spoken by one participant of the conversation and *turn2* was spoken by another participant as a reply of *turn1*. We are tasked to detect the emotion of *turn3*. So, *turn1* and *turn2* act as context for *turn3*. There are four different emotions in the data set, namely, *happy*, *sad*, *angry* and *others*. The problem is modeled as a four class classification problem where each of the emotions listed above is the target class.

2 System Description

2.1 Preprocessing

This section describes the preprocessing steps of the system. Few of the steps are standard; the steps are just mentioned and are not discussed in detail. Rather the steps which are critical for the performance in the task are discussed in detail. Standard steps are: converting all letters to lower case, removing numbers, removing white spaces, removing stop words, sparse terms and particular words. The most important preprocessing steps are:

Expanding abbreviations: In chat data there are infinite number of possible abbreviations or shorthand uses, most of which are not standard. Those abbreviations can not be left in the data set as is, because there are no embedding for those. In my system , it is chosen to expand the top 10% of such abbreviations and others are ignored. For this, I created a map of abbreviation to expansion manually by inspecting the data set.

Few examples: lol $\rightarrow$ laugh out loud, ur $\rightarrow$ you are etc.

Handling emojis: Emojis are the single most important piece of information in chat data. In most of the cases it is a huge clue about the emotion of the party in conservation. I had two options to handle emojis, one, to use some kind of embedding (Eisner et al., 2016) for emojis; two, convert emojis into text and then use word embedding. I chose to convert emojis into text; partly because of the robust performance of the word embeddings and partly because of lack of a proven quality embedding for emojis. Also, this conversion made the *weight of evidence* feature (see section 2.2.2) more effective.

Examples: 😄 $\rightarrow$ beaming face with smiling eyes, ☹ $\rightarrow$ sad face etc.

But, a conversion scheme shown in the above examples leads to infiltration of words like *face*, *with*. To avoid this, I created a list of stop-words

Proceedings of the 13th International Workshop on Semantic Evaluation (SemEval-2019), pages 335–339
Minneapolis, Minnesota, USA, June 6–7, 2019.

and removed those from the expanded text. With this modification the above examples will look like:
😄 → beaming smiling eyes, ☹ → sad.

2.2 Features

There were mainly two features, word embedding and weight of evidence. Each word in the conversation is embedded into a 300 dimensional embedding space and for each turn the weight of evidence is computed. I intentionally refrained myself from using any sentence encoder like BERT (Devlin et al., 2018) or ELMo (Peters et al., 2018), as I wanted to explore the lower level embedding of words rather than using sentence embeddings as back boxes.

2.2.1 Word Embeddings

In the system, word embeddings are created as an average of three word vectors, GloVe (Pennington et al., 2014), FastText (Bojanowski et al., 2016) and Paragram (John Wieting and Livescu, 2015). I used 300 dimensional word embeddings. The embedding vocabulary could cover ∼85% of the data set vocabulary (unique words in the data set) which in turn covered ∼97% of the entire text of the data set.

2.2.2 Weight of Evidence

Weight of evidence (WOE) is a measure of how much the evidence supports or undermines a hypothesis. Here the intention is to weigh the evidence of each word in determining the emotion of the conversation. WOE is defined as:

$$WOE_{word,event} = \ln \frac{\frac{N_{word}^{non-event}}{N_{word}^{totalnon-event}}}{\frac{N_{word}^{event}}{N_{word}^{totalevent}}}$$

where,
$N_{word}^{non-event}$: number of other class records that has the word
$N_{word}^{totalnon-event}$: total number of other class records
N_{word}^{event}: number of records of the class that has the word
$N_{word}^{totalevent}$: total number of records of the class

Top 1000 most common words for each of the emotion classes were collected and then their WOE is computed for each of the four emotion classes. In the example below the words are represented by four dimensional vector, those dimensions belong to the four emotion classes.

$WOE_{word,event}$	Word	
	`smiling`	`sad`
$WOE_{word,happy}$	0.9	0.2
$WOE_{word,sad}$	0.1	0.8
$WOE_{word,angry}$	0.2	0.1
$WOE_{word,others}$	0.6	0.5

Table 1: $WOE_{word,event}$ for words.

For each turn I add up the WOE vectors of the words in that turn. So, each turn also has a WOE embedding of four dimensions. This embedding is fed into the model as an auxiliary feature. When emojis were converted into text, the WOE vector of the words explaining an important emoji reflected the emotion nicely. Also, when an emoji is used multiple times, its effect is multiplied into the WOE embedding of the turn. For example, "😄😄😄 " in a turn produces the below WOE embedding:

$WOE_{word,event}$	turn = 😄😄😄
$WOE_{smiling,happy}$	2.7
$WOE_{smiling,sad}$	0.3
$WOE_{smiling,angry}$	0.6
$WOE_{smiling,others}$	1.8

Table 2: WOE embedding for turn "😄😄😄 ".

Please note that "😄 😄 😄 " was first converted into text as: beaming smiling eyes beaming smiling eyes beaming smiling eyes. In the above table WOE vector for the word "smiling" is shown. Similar exercise can be done for other words.

2.3 Deep Learning Model

Given the turns of a conversation, the target emotion label can be modeled in different ways. One, model the target label based on *turn3* only. Two, Consider all the turns as one single input of text (may be separated by EOS tokens) and from this learn the target label. But, none of the options are truly context aware. Construction of my model is based on the idea that every turn in a conversation builds on top of the previous turn. Also this task is treated as a multi-class classification problem where each emotion is treated as individual classes.
At the core of the system are three bi-directional

(Schuster and Paliwal, 1997) Gated Recurrent Unit (GRU) (Cho et al., 2014) layers, one each for the three turns in the conversation. Second and third layer are *derived* from their immediate previous layer. This is achieved by using the hidden states of a turn GRU layer to initialize the subsequent turn's GRU layer. Hence, when turn three layer starts with the hidden state of turn two layer which has already summarized the *context* of the ongoing conversation, it is building on top of the existing context. I see it as each layer is conditioned on the what has already been conversed before it. Used model is depicted in Figure 2. Then the additional features, i.e., the WOE values were introduced into the model by concatenating with the intermediate latent representation of the conversation.

2.3.1 Gated Recurrent Unit: GRU

A GRU unit (in Figure 1) can be represented by the following equations:

$$
\begin{aligned}
z_t &= \sigma\big(x_t U^z + h_{t-1} W^z\big) \\
r_t &= \sigma\big(x_t U^r + h_{t-1} W^r\big) \\
\tilde{h}_t &= \tanh\big(x_t U^h + (r_t * h_{t-1}) W^h\big) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

Here r is the reset gate, and z is the update gate. Intuitively, the reset gate determines how to combine the new input with the previous memory, and the update gate defines how much of the previous memory to keep. And h_t is the new hidden state.

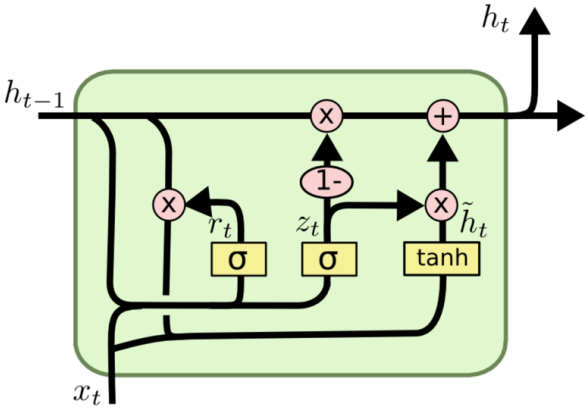

Figure 1: Gated Recurrent Unit. Figure adapted from (Olah, 2015).

2.3.2 Class Weights

The given data set is not well balanced (see Table 4 for details). To combat this issue I used $class\_weights$ for weighting the loss function, in a way it is to say the model which class to concentrate on. A *balanced* class weight is used to automatically adjust weights to be inversely proportional to class frequencies in the input training data. Weight of a class c_i is given by:

$$
weight_{c_i} = \frac{n\_samples}{n\_class \times n\_samples_{c_i}}
$$

Where,
$n\_samples$: total number of data sample
n_class: number of class present
$n\_samples_{c_i}$: number of samples of class c_i
The $class\_weights$ that were used are listed in Table 3.

class	$class\_weights$
$angry$	2.145
$happy$	0.841
sad	1.085
$others$	0.702

Table 3: $class\_weights$ for the input classes.

2.4 Data Description

We were provided 48544 data points to train our model. The class representations are shown in Table 4. It can be clearly seen that the data is highly imbalanced. This imbalance is handled by using weighted loss function and by fine tuning the model based on the micro-averaged f1 score (see section 2.5 for details).

Label	# data points
$angry$	5656
$happy$	14426
sad	11176
$others$	17286

Table 4: Class representation in training data.

2.5 Training Details

Data set is split (90 : 10) into train and validation. For class representation in the whole data set please see Table 4. In validation data generation, the proportion of class representation was kept similar to the data set. Table 5 shows the data split details.

I trained the model on the training data set and fine-tuned on the validation data set based on the

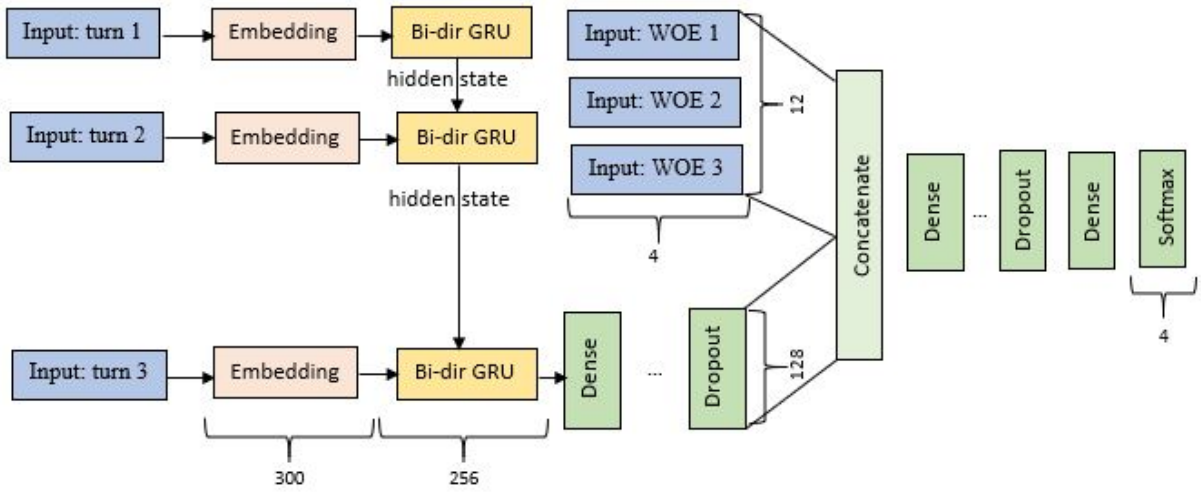

Figure 2: The deep learning model used in the system.

#Label	Training	Validation
#*angry*	5091	565
#*happy*	12983	1443
#*sad*	10058	1118
#*others*	15557	1729

Table 5: Class representation in training and validation data.

micro-F1 score. Since the data set is highly unbalanced, a weighted categorical cross-entropy loss is used, see Table 1 for the class weights. Adam (Kingma and Ba, 2015) optimizer is used with a learning rate of 0.001 and batch size of 128. Learning rate was decreased by 15% after each 3 epochs. Hidden state size of 256 is used for the bi-GRU gates. All the dense layers are of dimension 128 and a dropout of 0.5 is used for all of those.

3 Results

Here the detailed result of the system performance is presented. The performance shown in Table 6 is on the test data set.

label	precision	recall	f1-score	support
others	0.96	0.91	0.94	4677
happy	0.56	0.74	0.64	284
sad	0.61	0.80	0.69	250
angry	0.60	0.81	0.69	298

Table 6: System performance details

In the task the evaluation metric is micro-averaged F1 score only for the three emotion classes *happy*, *sad* and *angry*. Table 7 shows the confusion matrix of different classes.

label	*others*	*happy*	*sad*	*angry*
others	4246	162	119	150
happy	69	211	4	0
sad	36	3	200	11
angry	52	0	5	241

Table 7: Confusion matrix.

Precision and recall values for *happy*, *sad* and *angry* classes are 0.783653 and 0.589511 respectively. My system score is 0.6729 thereby beats the baseline (score 0.5868) convincingly.

4 Conclusion

With the system it is shown how to use the context information while detecting the emotion in a dialogue. Some guidelines about how to handle emojis is also laid out. While developing this system I realized the importance of pre-processing in conversational text data, or in general NLP related tasks; it can not be over emphasized.

Acknowledgments

I want to thank my mentors at Teradata, Ramesh Bhashyam and C Jaiprakash for the never ending support and to my teammates Lovlean Arora and Naveen TS for all the engaging discussions we had.

I want to apologize to my wife Samarpita, for all the weekend plans which were cancelled due to me working on TDBot. I promise to make up for those!

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *CoRR*, abs/1607.04606.

Ankush Chatterjee, Kedhar Nath Narahari, Meghana Joshi, and Puneet Agrawal. 2019. Semeval-2019 task 3: Emocontext: Contextual emotion detection in text. In *Proceedings of The 13th International Workshop on Semantic Evaluation (SemEval-2019)*, Minneapolis, Minnesota, USA.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bosnjak, and Sebastian Riedel. 2016. emoji2vec: Learning emoji representations from their description. *CoRR*, abs/1609.08359.

Umang Gupta, Ankush Chatterjee, Radhakrishnan Srikanth, and Puneet Agrawal. 2017. A sentiment-and-semantics-based approach for emotion detection in textual conversations. *CoRR*, abs/1707.06996.

Kevin Gimpel John Wieting, Mohit Bansal and Karen Livescu. 2015. From paraphrase database to compositional paraphrase model and back. volume 3, pages 345–358.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Christopher Olah. 2015. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Online; visited 29/03/2019.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Trans. Signal Processing*, 45:2673–2681.