Emotion Classification from Three-turn Conversation Data using Variations of RNN Layers with Attention

0. Abstract

This report implements a simplified NELEC model proposed by a research team from Microsoft as a part of the third task of 'Contextual Emotion Detection inText' as part of SemEval-2019 (Agrawal & Suri, 2019). The model uses neural embeddings from GloVe and Emoji2Vec to encode the corpora. The model features in LSTM with attention layer, GRU with attention layer, global max-pooling layer and global average pooling layer to train and regulate the weights and bias in the deep neural network. The code is posted on the google colab file.

1. Introduction

Sentiment analysis of contextual data has been known for classifying movie reviews, product reviews and customer satisfaction. The popular application of sentiment analysis has focused on a binary-positive-or-negative classification or a binary classification with a third, neutral class. Emotion classification with specific emotion categories, such as happy, sad, angry, etc., has gained popularity in recent years. Being able to detect emotions, the development of an emotionally intelligent agent will move one step closer to a fully automated chatbot with which humans can communicate emotions. During daily social interactions, humans perceive body language and facial expressions along with the conversation to reveal the true emotions. Lack of contextual cues creates ambiguity to both humans and machines. In modern days, people have leveraged emojis and face text to unveil their emotions and intensity of their emotions. For machine learning tasks, a combined network of emojis and texts have shown significant improvement on recognizing sarcasm (Subramanian, Sridharan, Shu, & Liu, 2019).

In this report, I included a pre-trained emoji vector with the text data embedding and implemented a neural deep learning model that includes a Gated Recurrent Unit, Long Short Term Memory and an Attention Mechanism. The model yields an F1-score of 68%.

2. Preprocessing

2.1 Data analysis

The EmoContext data provided by Microsoft contains 5 columns with unique ID, turn 1, 2, 3 and emotion context label (happy, sad, angry and others). The latter turn is the response to the former. The conversations include typos, abbreviations and emojis which are a truthful representation of common text messages. It's not hard to see from the table below that the class "Others" take a big portion of the distribution of categories. Since the "Others" category is not important for evaluation, it’s been excluded. The percentage of the “Others” class is 1 - the rest of the class.

	Train (%)	Test (%)	Dev (%)
Happy	14.2	13.3	0.53
Sad	18.0	18.6	0.45
Angry	18.2	18.6	0.54
Emoji	17.5	17.9	11.1
Data Coverage	99.1	98.9	99.2

Table 1: Global Statistics of the Train, Test and Dev Data

2.2 Text Preprocessing

The conversations are first cleaned with a regular expression program to revert the abbreviations and typos back to the unabbreviated and correct form. Unlike typical data processing procedures, lamentation and normalizations (remove of question marks and stop words) are not used because a paper suggests that the unlamentized words are a better representation of the magnitude of emotions (Agrawal & Suri, 2019). After cleaning, I tokenized the word and padded the tokenized sentences to equal length vectors.

After the data cleaning step, the text data needs to be encoded into numerical embedding to pass into a model. Embeddings can be created through different methods; the most popular ones are Bag-of-Words (BoW) and Term Frequency - Inverse Term Frequency (TF-IDF), which represents the word count or the relative frequency of each word. Those measures have difficulty revealing the conditional probability of the appearance of one word given that a certain other word is present. The Global Vectors for Word Representation (GloVe) provides pre-trained embeddings: each row represents a word and each column represents the trained weight in the hidden layer. The embeddings used models are trained with 6 billion tokens, 400,000 words in 300 dimensions. The vectors are essentially the factorization of the co-occurrence probability of the words, and the GloVe is a log-bilinear model with a weighted least squares objective function to calculate such vectors. The assumption that the co-occurrence probability of words uncovers the context of the words and its relationship with other words has been relatively accurate when the model is trained on a large dataset. The co-occurrence probability takes into account the local context also the global statististics and has been successful on word analogy task; it’s accuracy bypasses the Word2vec model by more than 10% and is more robust to more training cycles (Pennington, Socher, & Manning, 2014).

Aside from the GloVe embedding, the Emoji2Vec embedding is used to capture the meanings of emojis. The Emoji2Vec embedding is in the same shape of the GloVe embedding: each row represents a word, and each column is the value of the weights in the hidden layer. The Emoji2Vec model collects the sum of the Word2Vec embedding vectors of the description of the emoji, and trains a sigmoid function of the sum and a trainable vector x through a logistic loss function. Similar to the popular Word2vec model, it is able to generate patterns like 👑-🚹+🚺=1 : 👸; 2 : 👑 ; 3 : 🏰; 4 : 💂; 5 : 🏇. The augmented Word2Vec and Emoji2Vec has reported significant improvement on classification performance on the full corpus and tweets that contain emojis (Eisner, Rocktaschel, Augenstein, Bosnjak, & Riedel, 2016).

The pretrained embeddings are like word dictionaries that we store the rearranged word vectors to a matrix where the index of each row maps to the index of the tokenized data.

3. Construct the Model

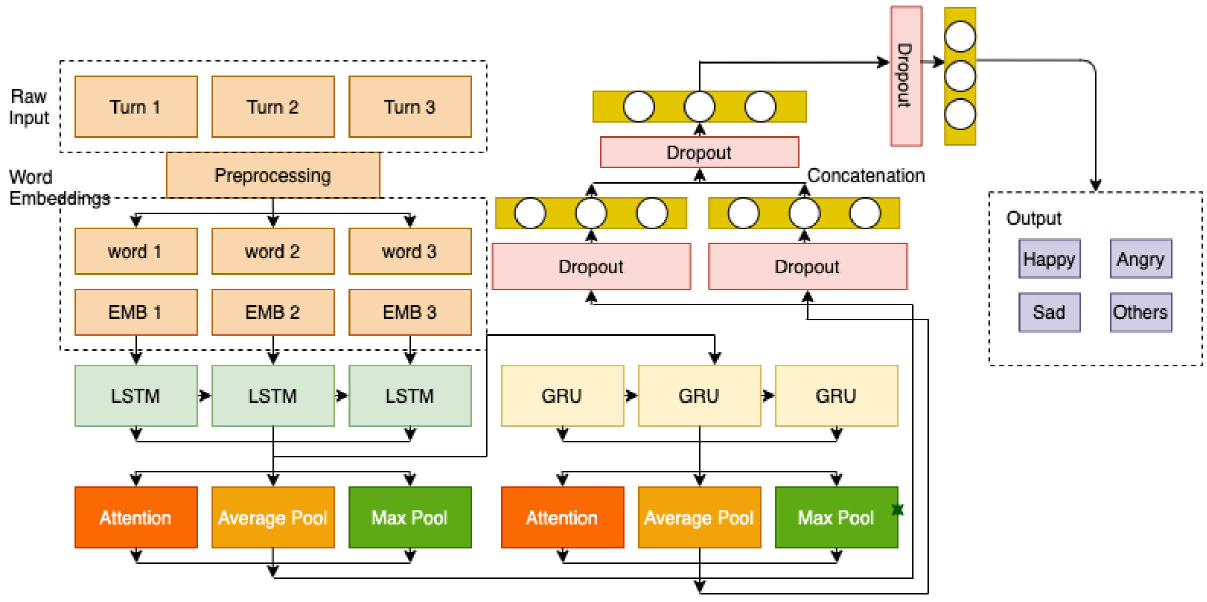


Figure 1: Model Diagram

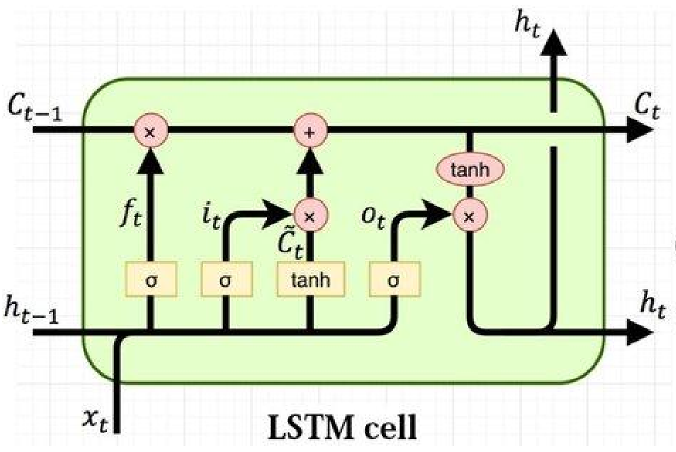
The model I trained is a simplified NELEC model which includes a Long Short Term Memory (LSTM) layer, a Gated Recurrent Unit (GRU) layer and a feed-forward attention layer (Agrawal & Suri, 2019). After the words in each turn are cleaned and tokenized and the embedding is created. We pass in word tokens as the input layer, and the embedding as the embedding layer. The output of the embedding layer will be passed in a series of LSTM, GRU, dropout and attention layers. The LSTM and GRU cells are variations of RNN cells. An RNN allows a feedforward neural network to carry information from the previous state, but it fails to reveal the long-term dependencies when the input sequence becomes long. This is because RNN is prone to vanishing gradients, which is a term to describe the gradients becoming smaller and insignificant and the weight changes becoming insignificant. In the world of Natural Language Processing (NLP), long corpora as input are common. The LSTM and the GRU to preserve the information.

LSTM Cell

Each Long Short Term Memory cell adds a forget gate, an input gate and an output gate, which are designed to decide what information to keep, what information is relevant to the current step and what information to pass to the next hidden state. The value from the hidden layer in the previous state $h_{t-1}W^f$ and current input x_tU^f are added and pass through a sigmoid activation function. The closer the result is to 0, the more likely the information will be "forgotten". This is called the forget gate.

The previous hidden state $h_{t-1}W^i$ and the current input x_tU^i also passes through a sigmoid function and a tanh function. The results of the two functions are multiplied. The closer the result is to 0, the less important is the current information. This operation is called the input gate.

The results from the forget gate will be pointwise multiplied with the value from the previous cell state $C_{t-1}$and added to the result of the input gate, whose result will be passed to a tanh function and combine with the result of $sigmoid(h_{t-1}W^o + x_tU^o)$. The $W^*$and $U^*$are the weight matrices.

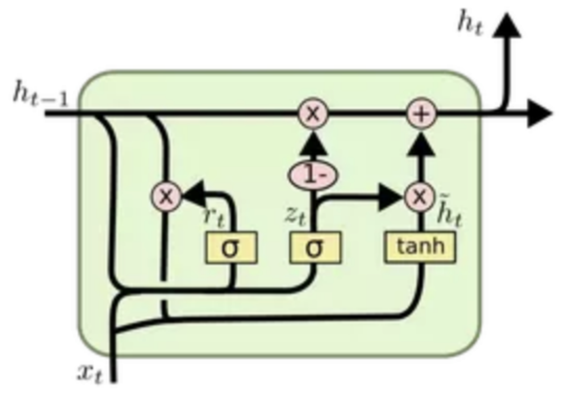


$$i_t = \sigma\left(x_tU^i + h_{t-1}W^i\right)$$
$$f_t = \sigma\left(x_tU^f + h_{t-1}W^f\right)$$
$$o_t = \sigma\left(x_tU^o + h_{t-1}W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_tU^g + h_{t-1}W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Figure 2: LSTM Cell

GRU Cell

The GRU cell, on the other hand, has a reset gate acting like the forget gate and a reset gate acting like the combination of input and output gates.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3: GRU Cell

The vanishing gradient issue is far less prominent because the multiplications used for RNN cells have been replaced by addition in many operations. During backward propagation, we are less likely to have gradients that are too small, which would stall the training.

After embedding, the final status of the hidden state of the LSTM model as the output is splitted into two directions. One goes to an attention layer, a global max pooling layer and global average layer in parallel. The other one goes to a GRU layer whose output will feed into an attention layer, a global max pooling layer and a global average layer in parallel. The attention cell here is the feed-forward attention proposed by Raffle and Ellis (Raffel & Ellis, 2016).

3.2 Feed-forward Attention

The attention mechanism is essentially an extra vector storing the importance weights of each token, which allows for a more direct dependency between the state of the model at various time steps. The Attention layer used in this model is a simplified version of the first Bahdanau's attention model. The Bahdanau's attention layer takes in the hidden state h_t of the LSTM model as input and feeds it into a trainable function $a(s_{t-1}, h_t)$. Each time, the weight of a is calculated by the softmax of the h_j and the previous state s_{t-1} (Yang et al., 2016). The feed-forward attention layer in the NELEC model, however, is a simplified version of Bahdanau's attention model. It does not include the previous state s_{t-1}while calculating the values for a. The final output vector $c$is computed as a weighted average of h_t for all the time t whose weights are given by a. This variation of the attention layer can handle variable sequence length and allows for parallel processing, but it takes more epochs to train (Raffel & Ellis, 2016). In the actual implementation, I found the advantages do not overweigh the disadvantage, so I decided to use the original implementation.

4 Model Result Analysis

The original model was implemented in Keras and I reimplemented it with Pytorch. I found a Pytorch deep learning object, nn.modules, that is easier to debug and requires me to calculate the dimensions of the layers, which help me understand the flow of the tensors. The training script is also more flexible for customization. The model is trained on 24128 rows of conversation, 300 dimensions, 35 maximum sequence length. The training procedure uses cyclic learning rate trained on 100 epochs (Smith, 2017).

	Keras Model (Original)	**Keras Model (Trained)**	**Pytorch Model (Implemented)**
Class Happy	Precision : 0.508 Recall : 0.694 F1 : 0.586	Precision : 0.512 Recall : 0.754 F1 : 0.610	Precision : 0.768 Recall : 0.458 F1 : 0.574
Class Sad	Precision : 0.496 Recall : 0.752 F1 : 0.598	Precision : 0.599 Recall : 0.800 F1 : 0.685	Precision : 0.547 Recall : 0.784 F1 : 0.645
Class Angry	Precision : 0.546 Recall : 0.799 F1 : 0.649	Precision : 0.517 Recall : 0.893 F1 : 0.655	Precision : 0.568 Recall : 0.867 F1 : 0.686
Macro Precision, Recall and F1	Precision : 0.5165 Recall : 0.748 F1 : 0.6111	Precision : 0.5427 Recall : 0.8156 F1 : 0.6518	Precision : 0.5244 Recall : 0.8061 F1 : 0.6354
Micro Precision, Recall and F1	Precision : 0.5179 Recall : 0.7488 F1 : 0.6123	Precision : 0.5370 Recall : 0.8177 F1 : 0.6483	Precision : 0.5217 Recall : 0.8082 F1 : 0.6341

Table 2: Model Results

4.1 Conclusions and Improvements

The pytorch model has reached a similar precision, recall and F1 scores as the model shown in the NELEC github and also the retrained Keras model. The ratio in Table 1 suggests the data imbalance in the sample, where the "Others" Class outnumbers of the other class. A few winning models for this contest have chosen the multitask models. For instance, the ANA model proposed by a research team from University of Alberta is a multi-head hierarchical model, the probability of a text sequence belonging to one class is calculated separately. It doesn't seem to completely solve the data imbalance problem, however, as they also suggest using a binary classifier for the "Others" and "Not Others" class to help with training (Huang, & Zaıan, 2019). The other solution is to feature-engineer from more traditional NLP methodology. The final NELEC model in the paper has also used the combination of neural features (scores from neural classifiers) and lexical features (Turn Wise Word n-Grams, etc).

Reference

Agrawal, P., & Suri, A. (2019, April 05). NELEC at SemEval-2019 Task 3: Think twice before going deep. Retrieved March 24, 2020, from https://arxiv.org/abs/1904.03223

Chatterjee, A., Narahari, K., Joshi, M., & Agrawal, P. (2019, June). SemEval-2019 task 3: Emocontext Contextual emotion detection in text. Retrieved March 25, 2020, from https://www.aclweb.org/anthology/S19-2005/

Eisner B., Rocktaschel T., Augenstein I., Bosnjak M., & Riedel S. (2016, November 20). Emoji2Vec: Learning emoji representations from their description. Retrieved March 25, 2020, from https://arxiv.org/abs/1609.08359

Huang C., A. T., & Zaıan, O. R. (2019, May 31). ANA at SemEval-2019 Task 3: CONTEXTUAL emotion detection in conversations through Hierarchical LSTMs and BERT. Retrieved March 26, 2020, from https://arxiv.org/abs/1904.00132

Pennington, J., Socher, R., & Manning, C. (2014, October). Glove: Global vectors for word representation. Retrieved March 24, 2020, from https://www.aclweb.org/anthology/D14-1162/

Raffel, R., & Ellis, D. P. (2016, March 28). Feed-forward networks with attention can solve some long-term memory problems. Retrieved March 25, 2020, from https://arxiv.org/abs/1512.08756v4

Smith, L. N. (2017, April 04). Cyclical learning rates for training neural networks. Retrieved March 25, 2020, from https://arxiv.org/abs/1506.01186

Subramanian, J., Sridharan, V., Shu, K., & Liu, H. (2019, July 09). Exploiting emojis for sarcasm detection. Retrieved March 24, 2020, from https://link.springer.com/chapter/10.1007/978-3-030-21741-9_8

Yang Z., Yang D., Dyer C., He X., Smola A., & Hovy E. (2016, June). Hierarchical attention networks for document classification. Retrieved March 25, 2020, from https://www.aclweb.org/anthology/N16-1174/