Sentim at SemEval-2019 Task 3: Convolutional Neural Networks For Sentiment in Conversations

Jacob Anderson
Sentim LLC, USA
`papers@sentimllc.com`

Abstract

In this work convolutional neural networks were used in order to determine the sentiment in a conversational setting. This paper's contributions include a method for handling any sized input and a method for breaking down the conversation into separate parts for easier processing. Finally, clustering was shown to improve results and that such a model for handling sentiment in conversations is both fast and accurate.

1 Introduction

The model for this paper was created for Task 3 of SemEval 2019, EmoContext (Chatterjee et al., 2019). The basic idea of the task is to classify the emotion (as "angry", "sad", "happy", or "others") that someone is expressing in the third turn of a three part conversation, given the previous two turns as context.

The dominant approach in many natural language tasks is to use recurrent neural networks or convolutional neural networks (CNN) (Conneau et al., 2016). For classification tasks, recurrent neural networks have a natural advantage because of their ability to take in any size input and output a fixed size output. This ability allows for greater generalization as no data is removed nor added in order for the inputs to match in length. While CNN's can also support input of any size, they lack the ability to generate a fixed size output from any sized input. In text classification tasks, this often means that the input is fixed in size in order for the output to also have a fixed size.

This work expands upon a previous work (Anderson, 2018), a way of using CNN's for classification to allow for any sized input length without adding or removing data. That work was expanded upon in this paper by making it simpler, using it in a different setting, and applying a new method to compensate for the previous model's deficiencies.

2 Model Description

The overall architecture of this model can be broken into two parts: the language understanding model and the emotion prediction model. The language understanding model takes in each part of the conversation and processes it separately in order to produce a latent vector representing the network's understanding of that part of the conversation. Then, the emotion prediction model takes all of the latent vectors from the language understanding model and combines them in order to make a prediction for what the emotion is.

More specifically, the language understanding model takes in the first and third turns of each conversation[1], processes them separately, and returns a respective understanding vector of size 128 for each input. The emotion model then concatenates those vectors to make one vector of size 256 and then processes that output through a small fully connected network in order to predict the emotional content.

The model was designed using Tensorflow (Abadi et al., 2015) and Keras (Chollet et al., 2015) and was trained using a Google Colab GPU.

2.1 Language Understanding Model

The idea behind the language understanding model (Figure 1) is to take in each part of the conversation separately and process them into a fixed size vector representing a machine understanding of the input text. In order to accomplish this, the input conversation was first embedded into a subword embedding using the byte-paired encoding provided by Heinzerling and Strube (2018) and SentencePiece (Kudo and Richardson, 2018). The subword embedding used a base vocabulary size of 10,000 with a vector size of 100. This em-

[1]The second turn was not used during my training because it was found empirically to make the results worse.

bedding was further extended with random vectors during data preprocessing to include emojis and other unique characters not found in the original vocabulary.

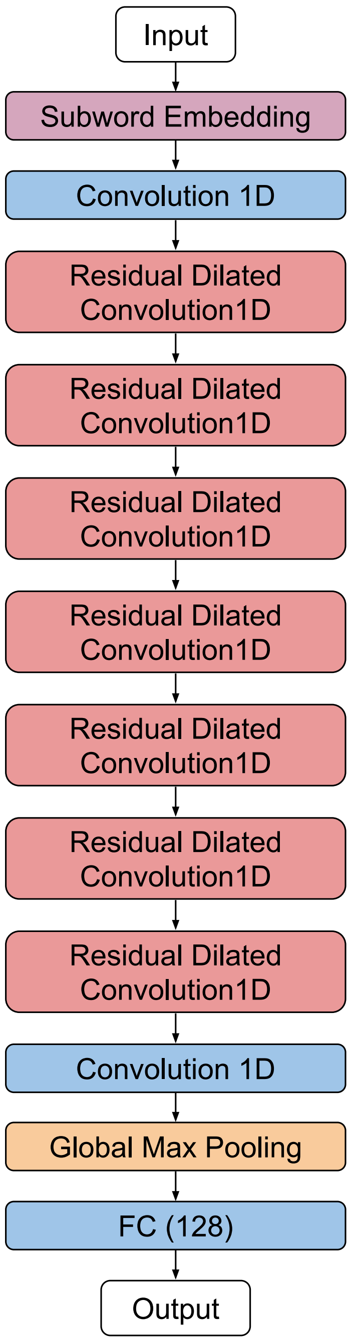
Input
Subword Embedding
Convolution 1D
Residual Dilated Convolution1D
Residual Dilated Convolution1D
Residual Dilated Convolution1D
Residual Dilated Convolution1D
Residual Dilated Convolution1D
Residual Dilated Convolution1D
Residual Dilated Convolution1D
Convolution 1D
Global Max Pooling
FC (128)
Output

Figure 1: The language understanding model.

A specialized convolutional stack was used in order to process the embeddings. The first layer of the stack was a convolutional layer with linear activation in order to set up the initial size of the input. This was followed by a stack of seven residual dilated convolutional layers (Figure 2) with relu activation. The number of layers was chosen so that the final layer would have a receptive field large enough to be able to see the whole input.

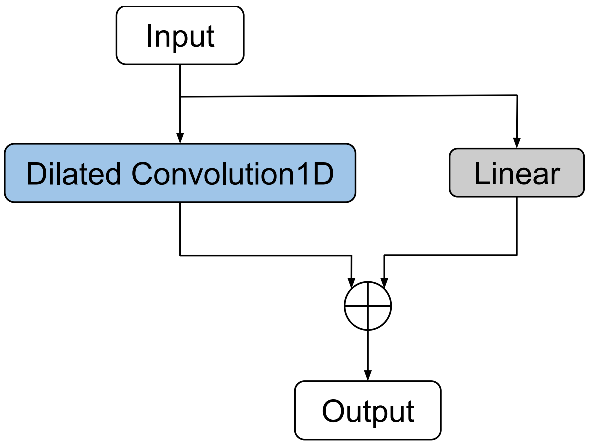

Figure 2: A residual dilated convolution, without skip connections.

One more convolutional layer was used with a linear activation function and the “same” padding method. This layer was originally added to increase the number of filters, but after a grid search was performed to determine the optimal hyperparameters, this layer was left in as a way to do any necessary linear transformations on the latent vector. Note that all convolutional layers in the final version had a kernel size of 2 and 300 filters. Additionally, the padding method varied depending on the experiment. See the “Experiments and Results” section for more information on that topic.

The convolutional stack was followed by a global max pool layer to bring the network down to a constant size, and then an FC Layer. The idea of this last FC layer is to allow for any fine tuning of the language understanding model as necessary and to compensate for the max pool layer’s inability to provide perfect information.

2.2 Emotion Prediction Model

The purpose of the emotion prediction model (Figure 3) is to predict the emotion given the latent vectors representing the conversation. The emotion prediction model starts off with the concatenation all of the language understanding outputs. This is then fed through one fully connected layer of size 256 with sigmoid activation, followed by the final prediction fully connected layer of size 4, also with sigmoid activation.

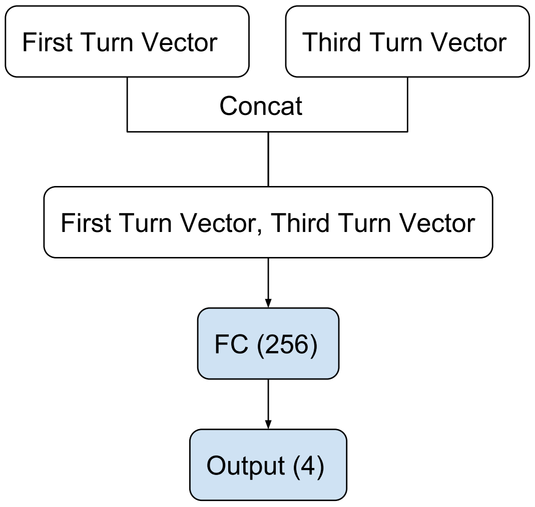

Figure 3: The layout of the emotion prediction model.

2.3 Training and Loss Functions

The network is end to end trainable without having to train the networks separately. Triplet loss was used (with triplets chosen via the all anchor-positive method) from Schroff et al. (2015) to cluster similarly labeled data together for the language understanding model, and softmax cross entropy loss was used for the emotion prediction model. The models were trained using the Adam (Kingma and Ba, 2014) optimization method with a learning rate of .001.

The loss function can be written as

$$L = T(a_s, p_s, n_s) + T(a_e, p_e, n_e) + S(labels, logits)$$

Where $T(anchor, positive, negative)$ is the triplet loss and $S(labels, logits)$ is the softmax cross entropy loss given the true labels and logit predictions from the emotion prediction model. Additionally, a_s, p_s, n_s are the anchor, positive, and negative examples corresponding to the first turn and a_e, p_e, n_e are the anchor, positive, and negative examples corresponding to the last turn respectively.

To further clarify, each first turn text in the batch was matched against every other first turn text so that they (the anchor) would be closer to similarly labeled data (positive examples) and farther away from every other category of data (the negative examples). The same thing was done for the last turn of the conversation. The first turn was specifically not clustered to be close to both first turn and last turn labels in order to preserve any contextual information that could be available in the first turn of a conversation versus the last turn.

An interesting thing to note is the time taken to train the model. Perhaps due to using the clustering loss and the softmax cross entropy loss simultaneously, the fast speed of training residual convolutions, or the small size of the dataset, the model always finished training within 7 minutes (or 9 epochs). Any more than that would never improve the micro f1 score and would start overfitting. Furthermore, roughly twenty percent of the time the model would show the best results within 1.5 minutes (or 2 epochs).

In one experiment, the networks were trained using just softmax cross entropy loss, but the networks never improved beyond the initial prediction of always pick the “others” class. This could be because of wrong hyperparameters (the same model was used as the one with the clustering loss), didn’t train for long enough (one model was trained for 24 hours (or 1755 epochs over the training data)), or didn’t try enough random restarts (10 restarts were attempted in this paper). Regardless of the case, this shows that using the clustering loss helped the network find the right solutions faster and with less hyperparameter tuning than just using the softmax cross entropy loss.

3 Experiments and Results

The experiments for this paper were all run on the dataset provided by the EmoContext organizers. The main dataset contains roughly 30,000 conversations, while the test dataset contains roughly 2,750, and the final evaluation dataset contains about 5,500. The order of each conversation goes Person A, Person B, then Person A again for all of the datasets.

Model Type	**Micro F1 Score**
118 network ensemble	.7295
Same padding ensemble	.7262
Causal padding ensemble	.7357
Best single model	.7255
2nd best single model	.707
Ensemble of micro f1 >.7	.7366
Ensemble of micro f1 >.71	.7386
Emoji replacement model	.7386

Table 1: Micro f1 score on the final evaluation set for each of my submissions.

The first submission on the evaluation dataset was an ensemble of 118 networks. In total, the 118 networks came from 100 different training cycles: 50 runs using causal padding as the padding method in the dilated residual convolutional stack, and 50 runs using the "same" padding method. Each run was trained against the full dataset for nine epochs and tested against the test dataset, so the checkpoints chosen were the ones that performed the best on the test dataset. The remaining 18 networks came from checkpoints that were not the best checkpoint of the training run but still had a micro f1 score above 0.7 on the test set.

To compare two different convolutional padding methods, an ensemble of all networks using "same" padding (56 in total), and an ensemble of all causal padding runs (62 in total) was submitted. While the causal padding models performed better on average, the difference was not significant. Two non-ensemble runs were submitted - one of which was the best run of all 100 runs (which happened to be a causal padding model) and the other of which was the second best run of all 100 runs (which happened to be a "same" padding model).

The sixth submission was an ensemble where every network in the ensemble had to have a micro f1 score of .7 or higher and the seventh submission was an ensemble where every network had to have a micro f1 score of .71 or higher respectively.

Unsurprisingly, the ensemble methods ourperform the best single model runs on micro f1 score. Interestingly though, the 118 network ensemble performed worse than the causal padding ensemble. This could represent opposing learning biases present in "casual" versus "same" padding types. That is, instead of the two different types of models agreeing with each other and removing bad predictions, the two models instead mostly disagree over certain portions of the data, leading to larger uncertainty and therefore lower overall performance.

A rule based model was also tested where every conversation was labeled based on whether or not it contained an emoji. More specifically, for every conversation the first and third turns of the text were concatenated and then if that had a happy emoji it was labeled happy, if it had a sad emoji it was labeled sad, if it had an angry emoji it was labeled angry, and if it didn't have any of those emojis then it was labeled using the output of the seventh submission. This model is interesting because applying the algorithm changed zero labels from the seventh submission. This means that the network learned the different emojis and used them in order to predict the emotion. This also agrees with the intuition that emojis should represent how a user is feeling (or perhaps that the dataset was created or biased by this idea). To see all of the results, view Table 1.

Submitter	F1 Score
1st Place	.7959
2nd Place	.7947
3rd Place	.7765
My Best	**.7386**
Median	.6947
Average	.6605

Table 2: Micro f1 score of the first, second, and third place submissions as well as the average and median scores of all submissions as compared to my best submission.

To see a comparison to other people's submissions in the competition, view Table 2. Additionally, I show the micro f1 score as well as the f1 score for the happy, angry, and sad labels for a few of my models in Table 3.

Model	F1	Angry	Happy	Sad
BSM	.7255	.7382	.6718	.7653
f1 >.7	.7366	.7481	.6812	.7803
f1 >.71	.7386	.7536	.6790	.7818

Table 3: Micro f1 score as well as the f1 score for the angry, happy, and sad labels. Best single model is abbreviated as BSM, as well as the ensemble of all models that scored higher than a certain f1 score as f1 >score.

4 Conclusion

A language understanding model and an emotion prediction model was trained in an end to end fashion for understanding and predicting the emotions of conversations. The previous work was adapted in order to leverage the ability to use any sized inputs with convolutional neural networks, and showed that such a model can be fast and accurate. Finally, it was shown that augmenting language learning using clustering loss can help augment training and improve results.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Jacob Anderson. 2018. Fully convolutional networks for text classification. *arXiv preprint arXiv:1902.05575*.

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.

François Chollet et al. 2015. Keras.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. 2016. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*.

Benjamin Heinzerling and Michael Strube. 2018. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *CoRR*, abs/1808.06226.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832.