[Team F2] Text-based Emotion Recognition

Vignesh Dhanasekaran (vdhanas) Bhavana Vijay Lalwani (blalwan) Mallika Sinha (msinha2)

***Abstract*—Affective computing is the study of identifying emotional labels to a given snippet of text. Applications of text emotion recognition include monitoring social media platforms for suicidal thoughts or understanding customer experience through reviews and marketing. However, language complexity and ambiguity makes it difficult for computers to parse and understand. We selected a support vector machine (SVM) as our baseline for emotion recognition in text. To improve on the baseline, we propose a deep neural network consisting of a combination of convolutional and recurrent layers. We compared different pre-trained word embeddings (GloVe and Word2Vec) as well as different recurrent architectures (LSTM, BiLSTM, GRU, BiGRU). The model and baseline performances were evaluated against the ISEAR dataset, which contains seven emotion labels. This dataset was collected by Wallbot and Scherer [1] in the form of self-reported emotional experiences of the respondents in a questionnaire. The best performing model architecture used LSTM with frozen Word2Vec embeddings which saw approximately a 7% increase in the F1 score and accuracy over the SVM baseline.**

I. Model Training and Selection

A. The Model

We used the basic structure from the work done by Kratzwald et al [2] for our model. We used a set of convolutional layers to better extract features from the dataset and enhance the model. A high level overview of the model architecture is shown in Figure 1. The model is outlined in more detail in terms the number of parameters and shapes in Table I.

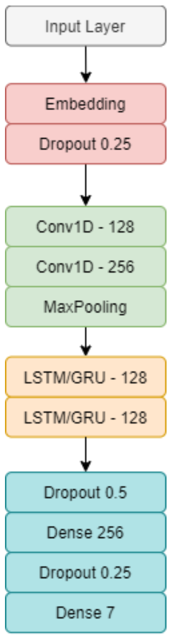


Fig. 1: Model Architecture for Affective Computing

Layer	Parameters	Output Shape
Embedding	3 000 000	200 × 300
Dropout 0.25		
Conv1D 128	115 328	198 × 128
Conv1D 256	98 560	196 × 256
MaxPooling 5		39 × 256
LSTM 128	197 120	39 × 128
LSTM 128	131 584	1 × 128
Dropout 0.5		
Dense 256	33 024	1 × 256
Dropout 0.25		
Dense 7	1799	1 × 7

TABLE I: Detailed Model Information

Our selected model uses a combination of recurrent and convolutional layers. Commonly, CNNs are used in image processing applications such as for classification on the ImageNet database. We exploit the features of convolutional layers by applying it to the output of our embedding layer, which is a two dimensional vector of size 200×300 containing high dimensional information about the words present in the data. Next, we used a stack of two recurrent layers in the model to learn about sequential relationships in the data. We note that any sequential data at this stage of the model will be abstract due to passing through the convolutional layers. In this study, we experiment with the LSTM and GRU as well as the bidirectional variants of both. Both LSTM and GRUs are common recurrent architectures used to combat the vanishing gradient problem present in long recurrent layers. The bidirectional versions help the model look forward in the sequence to anticipate data instead of relying on only past data. After our recurrent layers, we sent the data to a fully connected network for final classification. To combat overfitting, we employed dropout and max-pooling at several points in the model architecture.

B. Baseline

We created the baseline for our project using a Support Vector Machine (SVM) employing a Radial Basis Function kernel (RBF). SVMs are widely used for text sentiment analysis applications [3]. The RBF kernel is a non-linear kernel and is good for general-purpose when there is no prior knowledge about the data [4]. We used Term Frequency Inverse Document Frequency (TF-IDF) vectorization to convert text into vectors compatible with SVM classifier [5]. SVM classification on the

ISEAR dataset gave an F1 score of 0.567, which we used as the baseline for our project.

In addition, we wished to compare the model performance against the model trained by Kratzwald et al. on the ISEAR dataset. Their model used a RNN trained with transfer learning from a sentiment analysis task. Their best model run against the ISEAR dataset was an F1 score of 0.577 [2], which we use as a secondary baseline.

C. Implementation

Our main toolset in the project was Keras with Tensorflow as the backend [6] [7]. We utilized various machine learning functions from Keras such as Conv1D, Dense, as well as preprocessing packages such as `Tokenizer` and `fit_on_text` for tokenization. For classification reports, metrics, and our baseline creation, we used scikit-learn; specifically, we used the `classifiation_report`, `TfIdfVectorizer`, and `svm` from this library [8]. Additionally, we used Matplotlib and Seaborn to generate our plots and graphics [9] [10].

II. Experimental Section

A. Metrics

The main metric used for evaluation of our model and comparison with baseline was the F1 score. We also monitored other metrics like precision, recall and accuracy to compare different deep learning architectures.

B. Model Selection

After selecting the base form of our model, we tuned the hyperparmeters of the model. Our model's major hyperparameters were: the optimizer, the learning rate, the batch size, and the number of epochs for training. In Figure 2, we show the comparison of the effect of different optimizers while trained on our model. For each optimizer and learning rate, we trained the model for 5 epochs. Each data point in the plot represents the lowest validation loss the model was able to achieve for a given optimizer at the specified learning rate on the x-axis. We observed that the best optimizers were Adam, RMSProp, and Adam with Nesterov momentum, all with a learning rate of `1e-3`. We chose the Adam optimizer with a learning rate of `1e-3` as our final hyperparameter for learning due to its ubiquity in other applications.

Next, we compared the performance of the model with different batch sizes. Figure 3 shows the comparison of validation loss across different batch sizes while training the model. Note that we tuned the batch size after selecting the learning optimizer. We found that batch sizes of 8 and 16 had the most stable performance during training. We note that although a batch size of 32 also had relatively stable performance compared to higher batch sizes, its validation loss did not reach as low as with batch sizes of 8 and 16. We also found that the model started to overfit the data after 5 epochs, evidenced by the increasing validation loss.

After determining our base model and its hyperparameters, we experimented several variants of recurrent layers as well

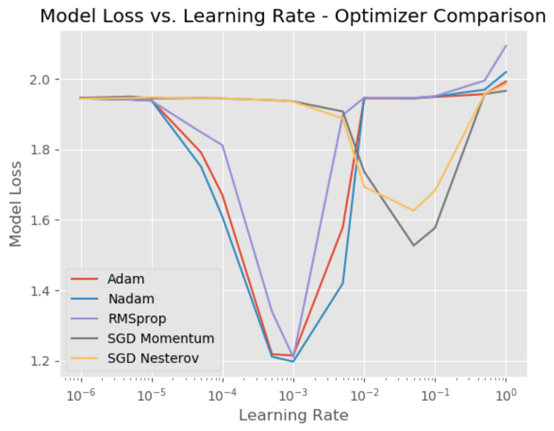


Fig. 2: Learning Rate selection.

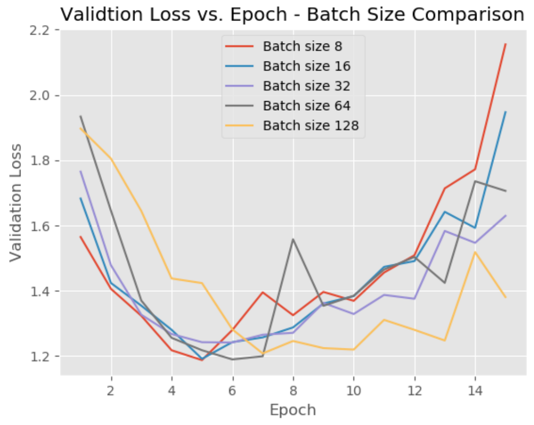


Fig. 3: Batch size tuning.

as different pre-trained word embedding layers. We compared the performances of LSTM, Bidirectional LSTM, GRU, Bidirectional GRU with GloVe (frozen and unfrozen) trained on Wikipedia 2014 [11] and Word2Vec (frozen and unfrozen) trained on GoogleNews [12]. When training unfrozen embedding layers, we loaded the pre-trained embedding and the model was allowed to fine-tune those parameters during backpropagation. Both embedding layers had a dimension of 300. We trained each model variant for 20 epochs and selected the model with the best validation loss in each variant as a candidate for evaluation on the test set. In Figure 4, we show a comparison matrix of the F1 scores of the variants when evaluated against the test set. The models with a higher F1 score are colored in darker blue while models with a lower F1 score are colored lighter. We found that the model with best F1 score was an LSTM with Word2Vec with a frozen pre-trained embedding layer; the model with least F1 score was the LSTM with Word2Vec unfrozen embeddings. We also observed that the BiLSTM had the most consistent performance across the different embedding variants. Another notable observation we found was that the model performed better when allowing the

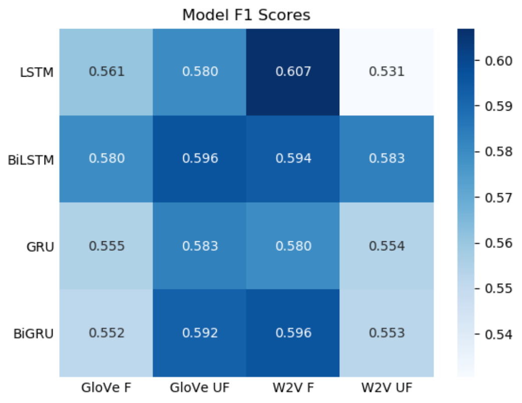


Fig. 4: Comparison of DNN model performance to the SVM.

GloVe embedding to be trained while the opposite was true when training with Word2Vec. The best performing model was the LSTM with frozen Word2Vec pre-trained embeddings, with an F1 score of 0.607. We used this configuration as our selected model for comparison against the baseline.

C. Performance and Comparison to Baseline

The model loss history during training for our selected model is shown in Figure 5. We can see that the validation loss

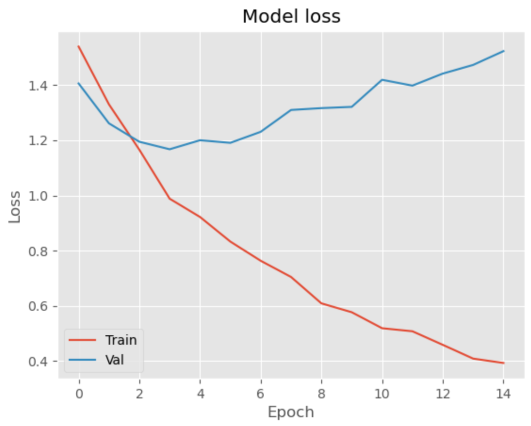


Fig. 5: Model loss history.

reaches its minimum after only 3 epochs. After this point, the model begins to overfit on the training data. We implemented early stopping and used the weights of the model with lowest validation loss even though the F1 score continued to increase for several more epochs, as shown in Figure 6. The model with the lowest validation loss however yielded the most confident model for the configuration. We note that in the F1 history plot, the score peaks at roughly 0.6 against the validation set and does not decrease even as the model begins to overfit the training data.

After selecting our model architecture and training, we compared the performance of our model against the baseline.

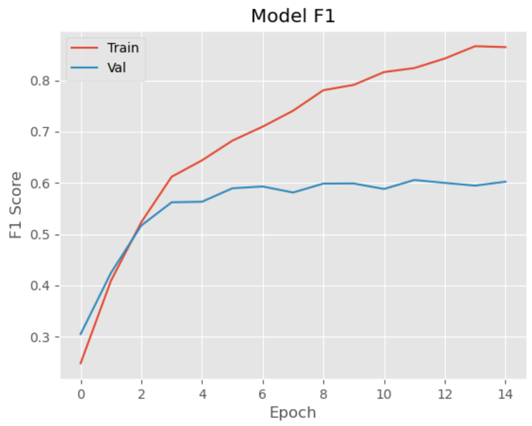


Fig. 6: Model F1 history.

The results of our model against our SVM baseline are shown in Table II. We also noted the relative change of our model over the baseline in the last column. We achieved a 7.05% improvement over the baseline in the F1 score metric. We also saw an improvement across accuracy, precision, and recall over the baseline. Additionally, we saw a 5.2% increase from

Metric	Baseline	Model	Change (%)
Accuracy	0.564	0.605	7.27
F1 Score	0.567	0.607	7.05
Precision	0.576	0.617	7.12
Recall	0.566	0.605	6.89

TABLE II: Baseline Comparison of the best model

our secondary baseline (0.577) taken from the model trained by Kratzwald et al [2].

From Section II-B, we found that our best performing model used frozen Word2Vec pretrained embeddings and an LSTM recurrent layer. A detailed classification report is shown in Table III. We observed that the model's accuracy and F1 scores were close together. This behavior implies that the model was able to learn some features of the data instead of randomly guessing and being correct.

Class	Precision	Recall	F1-score	Support
Anger	0.527	0.616	0.568	112
Disgust	0.776	0.590	0.670	100
Fear	0.693	0.725	0.709	109
Guilt	0.471	0.554	0.509	101
Joy	0.753	0.700	0.725	100
Sadness	0.581	0.626	0.603	115
Shame	0.517	0.422	0.465	109
Accuracy			0.605	746
Macro Avg.	0.617	0.605	0.607	746
Weighted Avg.	0.614	0.605	0.606	746

TABLE III: Classification Report of Selected Model

We generated a confusion matrix to get an insight into the accuracy achieved for the test dataset for each label as well as commonly confused labels. Figure 7 shows that the model predicted most of the emotions with around 60–70% accuracy, except for the emotion "shame", which had lowest accuracy of 42%. The model predicted the label "fear" most accurately at 72% with "joy" being a close second at 70%. One

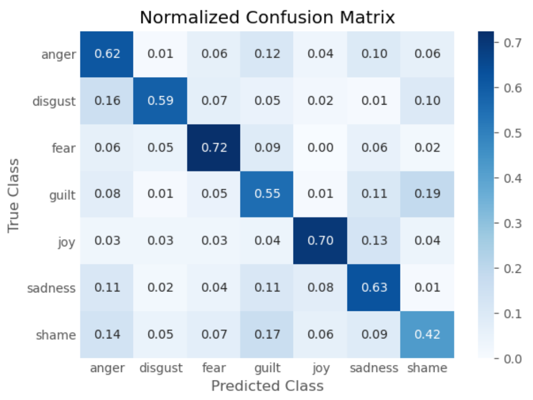
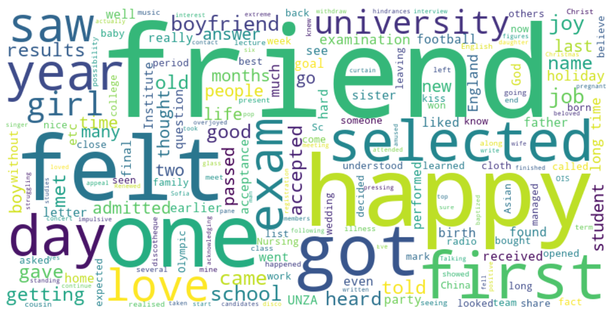


Fig. 7: Confusion matrix.

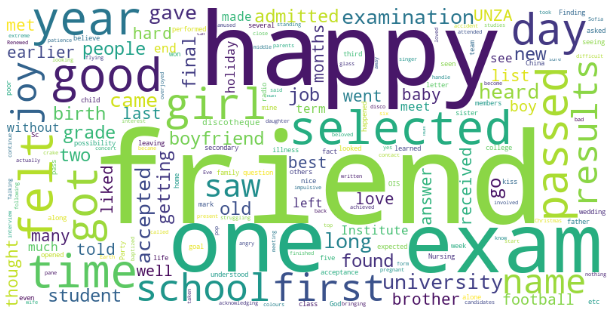
interesting observation we noticed from the confusion matrix was that the model often confused the emotion "sadness" with "anger" and "guilt". This was not surprising as even humans can get confused between these emotions, especially without context. Additionally, emotions are rarely so simple that a single emotion can be assigned to it especially in cases such as "sadness" and "guilt" or "guilt" and "shame"; humans will often feel both at the same time. Due to ambiguity present in language, "guilt" and "shame" are often similar and it is difficult to distinguish the two.

In order to visualize any relationships between sentences that the model predicts, we generated two word clouds for the text documents that were labeled "joy" from sentences in our selected test set. We generated the first word cloud, shown in Figure 8a using the ground truth labels. We then generated the second word cloud shown in Figure 8b, from the text documents for the selected test data that our model classified as "joy". The word clouds show the most common words found in the ground truth and the inference sentences. We observed that some words, such as "happy", "one", etc., appear in both the word clouds. Many of the larger labels are the same between both word clouds. This relationship means that the sentences that were predicted as "joy" and the sentences with a ground truth label as "joy" both had high similarity. However, there are some other words like "girl", "exam", and "good" which appear as different sizes between the two clouds. This finding implies that the model misclassified some of the predictions and may be assigning certain words an incorrect weight towards the label "joy".

We created a bar plot for a selected input, showing the probabilities of our output layer SoftMax activation . The outputs of each neuron act as the confidence of the model for the given label. One such relationship is shown in Figure 9, the input sentence is shown in the title. From the plot, we



(a) Ground truths



(b) Model inference

Fig. 8: Word cloud.

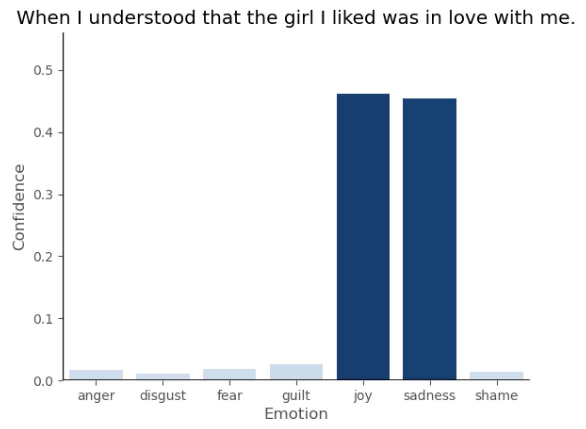


Fig. 9: Probabilities of Output for Selected "joy" Input.

saw that the model accurately predicted this input as "joy". However, we noticed that it was not confident in its assessment as the "sadness" output was quite close to the "joy" output. We initially suspected that this sentence would be easy to classify as "joy" because of the words "girl", "liked", and "loved". However, from the results, we determined that the model is looking at more than just what words are present in the input.

We then altered the input slightly to read "When I understood that the girl I liked **wasn't** in love with me", with the changed part in bold. By changing just one word, we expected that the model would predict sadness. The probabilities for this sentence are shown in Figure 10. In the new predictions, the model predicted that the sentence was associated with

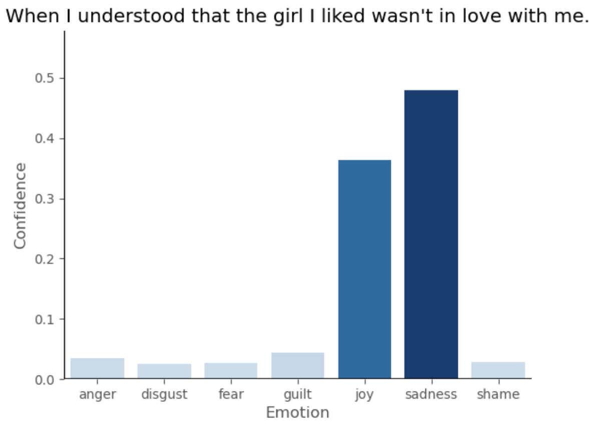


Fig. 10: Probabilities of Output for Altered Input.

"sadness", as expected. Again, we saw that the probability of the "joy" label was similar to "sadness" output. Both sentence structures were nearly identical and as a result, the model was not entirely certain if the sentence is describing "joy" or "sadness". Clearly, altering a single word or its placement in the input text can change the meaning of the sentence. The model will often confuse "joy" for "sadness" because it is easy to change the emotion from one to the other by adding a negation in the text or changing the order of some words. As a result of this semantic proximity, the model is uncertain about which one to pick and so shows similar probability outputs for both labels. These results are consistent with our confusion matrix and explains why the "joy" ground truth labels were sometimes predicted as "sadness". We saw similar behavior while comparing the output layer probabilities for a sentence with a ground truth label of "shame", shown in Figure 11. From our confusion matrix in Figure 7, we found that the

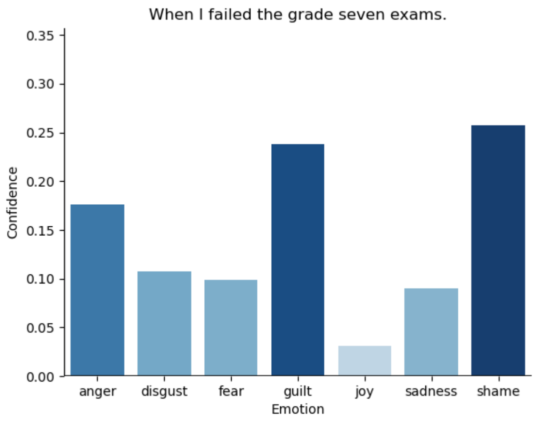


Fig. 11: Probabilities of Output for Selected "shame" Input.

ground truth "shame" was often confused with "guilt" and "anger". We can see from the bar plot that for a sentence labeled "shame", the labels "anger", "guilt", and "shame" all had high probabilities.

III. Conclusion

Affective computing allows computer programs to identify and label emotions from text sequences. This capability has the ability to provide support for marketing teams seeking to understand customers or to social media platforms monitoring for hate speech or suicidal posts. Natural language however has a complex structure that makes it difficult for naive machine learning approaches, such as the SVM, to perform well. We propose a modification on basic recurrent neural networks by adding a set of convolutional layers to the beginning of the model architecture.

Our project experimented with different recurrent structures as well as different pre-trained embedding layers to determine the best structure for affective computing. Both the baseline and deep neural network models were trained and evaluated on the ISEAR dataset, consisting of self-reported experiences by participants. Overall, we found that the LSTM recurrent structure with frozen Word2Vec embeddings gave the highest increase in our selected baseline. Our results showed a 5.2% increase in the deep neural network trained by Kratzwald et al and a 7.2% increase in the SVM baseline.

References

[1] H. G. Wallbott and K. R. Scherer, "How universal and specific is emotional experience? Evidence from 27 countries on five continents," *Social Science Information*, vol. 25, no. 4, pp. 763–795, 12 1986. [Online]. Available: http://ssi.sagepub.com/cgi/doi/10.1177/053901886025004001

[2] B. Kratzwald *et al.*, "Deep learning for affective computing: Text-based emotion recognition in decision support," *Decision Support Systems*, vol. 115, pp. 24–35, 2018. [Online]. Available: http://arxiv.org/abs/1803.06397

[3] H. A. Bin Sulaiman *et al.*, *2014 I4CT : 1st International Conference on Computer, Communications, and Control Technology : proceedings : Fave Hotel, Langkawi, Kedah, Malaysia, 2-4 Sept 2014*.

[4] Vasista Reddy, "Sentiment Analysis using SVM." [Online]. Available: https://medium.com/@vasista/sentiment-analysis-using-svm-338d418e3ff1

[5] D. Jurafsky and J. H. Martin, *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*, ser. Prentice Hall series in artificial intelligence. Upper Saddle River, N.J.: Prentice Hall, 2000. [Online]. Available: //catalog.hathitrust.org/Record/004076860http://hdl.handle.net/2027/mdp.39015047846814

[6] F. Chollet *et al.*, "Keras," Tech. Rep., 2015. [Online]. Available: https://keras.io

[7] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," Tech. Rep. [Online]. Available: www.tensorflow.org.

[8] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," Tech. Rep., 2011. [Online]. Available: http://scikit-learn.sourceforge.net.

[9] M. Waskom *et al.*, "seaborn: version 0.8.1," 11 2014. [Online]. Available: https://zenodo.org/record/12710#.XX-8CmlKhhEhttps://doi.org/10.5281/zenodo.12710https://zenodo.org/record/12710#.XX-8CmlKhhE%0Ahttps://doi.org/10.5281/zenodo.12710

[10] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 5 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4160265/

[11] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," Tech. Rep., 2014. [Online]. Available: https://nlp.stanford.edu/projects/glove/

[12] T. Mikolov *et al.*, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*. International Conference on Learning Representations, ICLR, 2013.