Project 2 - Emotion

Henrik Daniel Christensen[*hench13@student.sdu.dk*]
Frode Engtoft Johansen[*fjoha21@student.sdu.dk*]

DM873: Deep Learning
University of Southern Denmark, SDU
Department of Mathematics and Computer Science

1 Introduction

The aim of this project is to design and implement two deep learning models for text/sentiment classification, capable of identifying the emotion conveyed in a given text. The model is trained on the *dair-ai/emotion* dataset, a widely used collection of 20,000 labeled tweets available through Hugging Face. This dataset is divided into six emotion categories: sadness (0), joy (1), love (2), anger (3), fear (4), and surprise (5). Of the total dataset, 2,000 tweets are reserved exclusively for final testing.

2 Label Distribution

We started by analyzing the distribution of the labels in the dataset, see Figure 1.

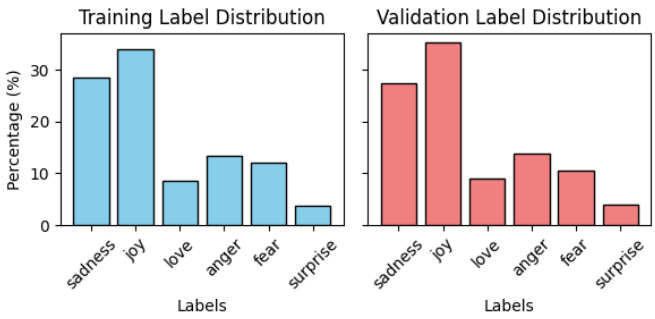

Fig. 1. Distribution of the labels in the dataset.

From the figure, it is evident that the dataset is highly imbalanced, with the majority of tweets labeled as either joy or sadness. However, the validation set exhibits a roughly similar distribution of labels, suggesting that the test set is likely to follow a comparable pattern.

This imbalance in the training data is expected to impact the model's performance, particularly for less represented labels, as sentences labeled with joy are less likely to be classified correctly. This challenge must be carefully considered when evaluating the model's performance and interpreting the results.

3 Preprocessing

To enable a machine learning model to process textual data effectively, it is essential to convert the text into numerical representations. The first step in this process is tokenizing the text, which involves splitting sentences into smaller units, such as words. For this task, we utilize a custom RegexpTokenizer from the NLTK library, which filters the text to retain only words, numbers, and a limited set of special characters. This ensures that irrelevant symbols are excluded while preserving the meaningful components of the text.

Following tokenization, the text is further refined by removing stopwords and applying stemming. Stopwords, which are frequently used words such as "the," "is," and "and," are removed as they are unlikely to contribute significant value to sentiment analysis. This is particularly important when using a limited sequence length for input sentences, as it allows the model to focus on more informative features while reducing the size of the vocabulary. Consequently, the model learns fewer parameters, which can improve its generalization and efficiency.

In addition, we apply stemming using the PorterStemmer from the NLTK library, which reduces words to their base or root forms (e.g., "feeling" becomes "feel"). This process further reduces the vocabulary size by grouping inflected or derived word forms together, helping the model to recognize similar meanings and improve its performance.

To better understand the text data after preprocessing, we visualize the most common words using a WordCloud, as shown in Figure 2. This provides an intuitive way to analyze the prominent terms in the dataset and assess whether the preprocessing aligns with the goals of the sentiment analysis task.

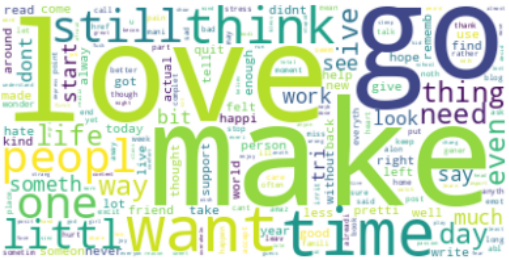

Fig. 2. WordCloud of the vocabulary.

As expected many emotional words are present in the vocabulary, e.g. 'love', 'friend', 'hate', 'never', 'go', etc.

To be able to process the text data, we need to choose a maximum sentence length. A boxplot of the distribution of the lengths of the sequences can be seen in Figure 3.

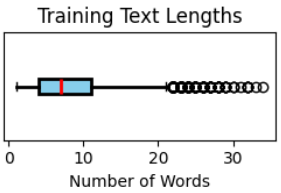

Fig. 3. Distribution of the lengths of the sequences.

From the boxplot, we see that the majority of the sequences have a length of around 8 words. We choose to set the maximum sequence length to 10 words. This means that all sequences longer than 10 words are truncated, and all sequences shorter than 23 words are padded with a special token, '<PAD>', to make them all the same length. In addition, we discovered that some sentences are very short, therefore we choose to remove all sentences with a length less than 3 words.

Having our cleaned and tokenized text data, we build a vocabulary from the training dataset. The vocabulary is built by mapping each unique word to an integer. The vocabulary for the training dataset ended up being 10,336 words.

Then, we convert the text data to integers by mapping each word in the text data to the corresponding integer in the vocabulary and pad the sequences to the maximum sequence length. This is done for both the training, validation and test datasets. The text data is now ready to be used for a model.

4 Models

Two different models were developed for the sentiment analysis task. The first model is a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) cells, and the second model is a Transformer model.

4.1 Model 1: RNN-LSTM

Model Architecture We chose to begin with a Recurrent Neural Network (RNN) model using LSTM (Long Short-Term Memory) cells due to their proven effectiveness in capturing long-term dependencies in sequential data, such as text. LSTMs are particularly suited for this task because they address the vanishing gradient problem often encountered in standard RNNs.

The architecture of the model, shown in Table **??**, was designed to balance complexity and performance, resulting in a total of 1,117,734 trainable parameters. Key considerations behind the design are as follows:

– Embedding Layer: The vocabulary size of 10,336 and embedding dimension of 75 were chosen to compactly represent the input text while capturing semantic relationships between words effectively. This dimensionality ensures the embeddings are rich enough for the sentiment classification task without being computationally excessive. - LSTM Layer: A single LSTM layer with 256 hidden units was selected to provide sufficient capacity to capture sequential patterns in the text. This configuration was chosen to avoid overfitting while retaining the ability to model complex dependencies in the data. - Dropout Layer: A dropout rate of 50% was applied to the LSTM outputs to reduce the risk of overfitting by preventing the model from becoming overly reliant on specific neurons. This regularization technique ensures better generalization to unseen data. - Output Layer: The final linear layer maps the 256 features to the 6 output classes, corresponding to the emotions in the dataset.

The specific parameters mentioned above was found using a combination of empirical testing (grid search) and theoretical considerations to achieve a balance between model complexity and performance.

Training and Evaluation The model training was conducted using the AdamW optimizer with a learning rate of 0.001 and a batch size of 16. AdamW was chosen due to its effectiveness in handling sparse gradients and its built-in weight decay mechanism, which helps prevent overfitting. The batch size of 16 strikes a balance between computational efficiency and the stability of gradient updates.

For the loss function, we used CrossEntropyLoss, which is well-suited for multi-class classification problems like this one, where the task involves predicting one of six emotion categories. To further regularize the model and reduce the risk of overfitting, an L2-norm penalty (weight decay) of 0.0001 was applied to the weights, encouraging smaller parameter values and promoting a simpler model. The model was trained for 4 epochs, as using more epochs lead to overfitting. The final test accuracy and F1-score is summarized in Table **??**.

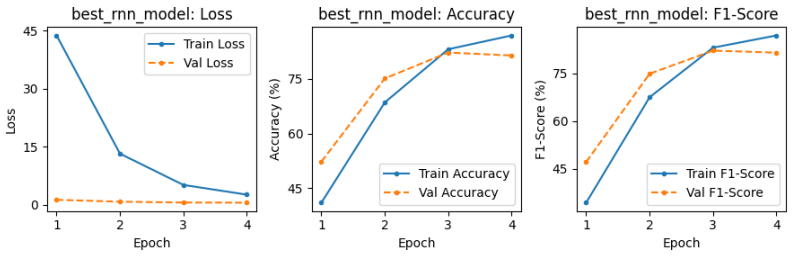

Fig. 4. Training and validation loss, accuracy, and F1-score for the RNN-LSTM model.

Label	Precision	Recall	F1-Score
sadness	0.91	0.86	0.89
joy	0.85	0.86	0.85
love	0.62	0.69	0.65
anger	0.84	0.81	0.82
fear	0.81	0.85	0.83
surprise	0.65	0.65	0.65
accuracy			0.83
weighted avg	0.84	0.83	0.83

Table 1. RNN-LSTM model performance on test set.

From the above table, we can see that the model performs well on sadness, joy, anger and fear, but not as well on love and surprise. This is likely due to the imbalanced distribution of the classes in the dataset as discussed in Section 2, see Figure 1.

4.2 Model 2: Transformer

The model is made from the solution to exercise sheet 5. Very few elements have been changed since it follows the pages in the Bishop book closely. The only big changes that has been made to the model itself is added dropout layers and positional encoding. We also experimented with randomly removing 2 tokens from each sentence to increase generalisation but it didn't work. WE also tried adding 2 more linear layers to the transformer block, but this was detrimental to the accuracy so we removed them again. We chose to use a transformer model as it seems like a good architecture for working with text since it makes words weigh differently depending on which other words are present in the sentence, and with positional encoding also where they are in that sentence. This is important since the meaning of words change depending on other words, and position in sentence. We used sinusoidal position embedding since this is the positional embedding used in the Bishop book. The L in the sinusoidal position embedding is 10000. The example in the Bishop book had an L of 30, and some other examples we encountered had an L of 10000. We chose 10000 as the value for L since we encountered it multiple places.

We used cross entropy loss as a our loss function since this is a good loss function for classification tasks. This is the number of heads in the multi head self attention mechanism. d_model should preferably be divisible by this number so each head computes the same size of input. We chose 8 heads since having too many heads can be inefficient and since we have d_model of 128, choosing more heads would lead to quite few dimensions per head, which would make it hard for them to capture any useful features. The mlp factor controls the amount of nodes in the linear layers of the transformer block. We use an mlp factor of 4. This was the default value and decreasing this value hit the performance of the model. Increasing the mlp factor would increase the training time so we decided to not do that. d_model is the dimensions of each token. We have a vocabulary of 10336 so we made this pretty big to ensure that each word is able to be uniquely represented.

We have 6 layers of transformer blocks. According to the slides, 6 layers is a normal amount of layers and 12 layers is for very large models. We experimented with 6 and 12 layers and found only a small performance increase with 12 layers, but a significant increase in training time, so we decided that we would rather train quicker to test values for other parameters. When training the transformer models with different parameters we found out that they would quickly overfit, so we needed to generalise the model. To do this we added two dropout layers on the linear layers of the transformer blocks. The dropout chance for the first layer is 50% and for the second layer is 25%. We tested with both higher and lower dropout chances and this worked the best. We chose adam as the optimizer. From the last project our and other peoples takeaway were that adam was a good optimizer in general, and since were not using weight decay, we don't need to use adamW. We used a learning rate of 0,0003. This was discussed as a good learning rate from the last project, and our experimenting also led us to this learning rate. We experimented a bit with using weight decay but it did not work well so we decided not to use it in our final model. When testing different parameters, we used 50 epochs. The transformer models quickly overfitted so we didn't need more epochs. We experimented with both L1 and L2 regularisation. L1 regularisation worked the best, and that is probably because it makes some weights go to 0, which indicates those words aren't important. In sentences, some words tells more than other words, so it makes sense that this would be good.

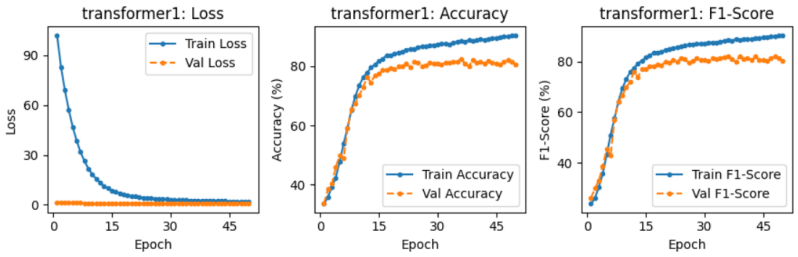

Fig. 5. Training and validation loss, accuracy, and F1-score for the transformer model.

Label	Precision	Recall	F1-Score
sadness	0.89	0.82	0.85
joy	0.81	0.88	0.85
love	0.68	0.59	0.63
anger	0.77	0.81	0.79
fear	0.80	0.80	0.80
surprise	0.67	0.56	0.61
accuracy			0.81
weighted avg	0.81	0.81	0.81

Table 2. Transformer model performance on test set.

Despite a lot of parameter tuning, we could not get the transformer model to get a better performance. We even tried adding extra linear layers in the transformer block, and removing random words from the sentences for better generalisation.

5 Analysis and Final Prediction

Of the two models, the performance of the recurrent neural network were better with an accuracy of 83% to the transformer models accuracy of 81%. Both architectures are good at predicting based on text, so its no big surprise that they are close in performance.

Since the transformer model is the worst performing, we decided to take a closer look at some failed classification cases to see why it performed as it did. These are the 7 sentences we looked at.

Number	Sentence	Transformer prediction	Actual classification
1	im updating my blog because i feel shitty	Joy	Sadness
2	i never make her separate from me because i don t ever want her to feel like i m ashamed with her	Joy	Sadness
3	i left with my bouquet of red and yellow tulips under my arm feeling slightly more optimistic than when i arrived	Sadness	Joy
4	i cant walk into a shop anywhere where i do not feel uncomfortable	Joy	Fear
5	i explain why i clung to a relationship with a boy who was in many ways immature and uncommitted despite the excitement i should have been feeling for getting accepted into the masters program at the university of virginia	Anger	Joy
6	i jest i feel grumpy tired and pre menstrual which i probably am but then again its only been a week and im about as fit as a walrus on vacation for the summer	Joy	Anger
7	i find myself in the odd position of feeling supportive of	Anger	Love

Table 3. Sentence Classifications with Transformer prediction and Actual classification

Looking at these sentences my intuition would be that the model looks at keywords to decide which category it belongs to. Some of the sentences lack any keywords that indicate what feeling they are associated with. The first sentence has a keyword 'shitty', but it might be a rare word in the training set. Sentence 1 also contains the word updating which is probably a positive word for the model, which is why it predicts joy. Sentence 6 contain the word 'vacation' which is probably why the model predicts joy for that sentence. This could indicate it is looking for keywords. If you were looking to deliberately make the model fail, we predict that missspellings, irony and figurative language would be some good techniques to do so.

To confirm our suspicions we tried to edit some of the sentences to get them correctly classified. Changing sentence 1 to "im updating my blog because i feel sad" changed the classification to sadness which is the correct one. This indicates that keywords are important. On the other hand, changing sentence 5 to "i explain why i clung to a relationship with a boy who was in many ways despite the excitement i should have been feeling for getting accepted into the masters program at the university of virginia" did not change the prediction or even the confidence which is a big surprise since the only change in that sentence is removing "immature and uncommitted" which seems like the most negative words in that sentence, so maybe the model is less keyword focused than anticipated.

6 Pretrained Model (DistilBERT)

To evaluate how our model compares to a larger pretrained model, we utilized the DistilBERT base model (uncased) from Hugging Face. DistilBERT is a distilled version of BERT, pretrained on a large corpus comprising 11,038 books and the English Wikipedia. It features a vocabulary size of 30,000 tokens and contains approximately 67 million parameters, making it significantly larger and more complex than our models. To adapt DistilBERT to our sentiment analysis task, we fine-tuned the model for 5 epochs using a small initial learning rate of 5×10^{-6}. This low learning rate ensures stable and effective fine-tuning of the pretrained weights without overfitting to our dataset. The performance of DistilBERT on our task is summarized in Table 4.

Label	Precision	Recall	F1-Score
sadness	0.92	0.93	0.92
joy	0.90	0.92	0.91
love	0.74	0.71	0.72
anger	0.90	0.89	0.89
fear	0.87	0.89	0.88
surprise	0.82	0.61	0.70
accuracy			0.89
weighted avg	0.89	0.89	0.89

Table 4. DistilBERT model performance on test set.

From the above results, we see that the pretrained model performs generally better than our two models. Notice that the performance on the less presents classes love and surprise is also lower as seen in the previous models.

7 Conclusion

We constructed two different models to complete the task, a recurrent neural network with LSTM and 256 hidden units, and a transformer model with 6 transformer layers. The recurrent neural network had the best test accuracy of 83%, while the transformer model had a test accuracy of 81%. We tried using a pretrained model distilBERT on our task to see how it would fare. It got a test accuracy of 89%. As expected our models were worse. This can be explained by the fact that distilBERT has more parameters and a much bigger training set than our models.

Overall we are satisfied with our result but it could have been improved. Initially an accuracy of 83% seems quite bad, but even the pretrained model had a test accuracy of less than 90% so compared to that, it does not seem too bad.

7.1 Individual Contributions

	Henrik Daniel Christensen	**Frode Engtoft Johansen**
Code	Task 1, 2, 5	Task 3, 4
Report	Section 1, 2, 3, 4.1, 6	Section 4.2, 5, 7

Table 5. Individual contributions.