Prof. Simon Clematide

Exercise 3

Emotion Recognition

1 Preprocessing

After loading the data line-by-line from GitHub, and before filtering out the selected emotion classes, we inspected some samples to identify which preprocessing steps we would need to perform. Since the data originate from tweets, there were a lot of hashtags present as well as a lot of chat words and emojis. User mentions also appear frequently and most of them are already anonymized. Therefore we decided on the following preprocessing steps:

1. Convert Unicode emojis to emoji names and make sure there's a space between consecutive emoji names,
2. Decode HTML characters, such as "&amp;",
3. Replace some common chat word abbreviations with their full form,
4. Handle special cases like removing extra new line characters and adding space before hashtags such that they can be treated separately,
5. Lowercase text, and
6. Tokenize and lematize all words while also removing digits, punctuation and stop words.

We then build our word-to-index dictionary, by combining the data from all splits, and adding the special padding token and unknown token. We use the dictionary we generated to encode all tweets as arrays of the same size (equal to the maximum available size, in our case 75), adding padding at the end of each one if necessary and replacing the actual tokens by their dictionary IDs.

We choose *anger* and *joy* as the emotion classes to predict in our first dataset and filter out the rest. At this point it needs to be mentioned that we interpreted the instructions as performing binary classification for each set, **not** making predictions over all 4 classes, since the data contain only two of them each time. Anger and joy correspond to labels 0 and 1 so there's no need to also encode the labels.

For the second dataset, we use the *anger* and *sadness* classes with labels 0 and 3, respectively. In this case, we manually change the sadness label to 1, otherwise the model would expect 4 classes in total.

2 Choosing & Training a model

We use the CNN class from the tutorial session and experiment with different hyperparameters in the model and optimizer initialization step. The parameters we played around with, as well as the results they yielded for the validation/development set, can be found in Table 1, where we've highlighted the best-performing one.

Model	Hyperparameters		Accuracy	F1-macro
1	Embedding dimension	300	0.7082	0.6674
	Filter sizes	[3, 4, 5]		
	Number of filters	[100, 100, 100]		
	Dropout	0.5		
	Optimizer	Adadelta		
	Learning rate	0.01		
2	Embedding dimension	500	**0.7588**	**0.7242**
	Filter sizes	[3, 5, 7]		
	Number of filters	[120, 120, 120]		
	Dropout	0.5		
	Optimizer	Adadelta		
	Learning rate	0.01		
3	Embedding dimension	500	0.7160	0.6653
	Filter sizes	[3, 5, 7]		
	Number of filters	[200, 100, 200]		
	Dropout	0.1		
	Optimizer	Adadelta		
	Learning rate	0.05		
4	Embedding dimension	300	0.7315	0.6836
	Filter sizes	[3, 5, 7]		
	Number of filters	[150, 150, 150]		
	Dropout	0.5		
	Optimizer	Adadelta		
	Learning rate	0.005		
5	Embedding dimension	500	0.7432	0.6984
	Filter sizes	[3, 5, 7]		
	Number of filters	[200, 200, 200]		
	Dropout	0.5		
	Optimizer	Adadelta		
	Learning rate	0.05		
	Weight decay	10^{-4}		

Table 1: Dataset 1, *Anger & Joy*. Accuracy and F1-macro scores on the development (validation) set for the different configurations of hyperparamaters of the CNN model.

3 Results

We re-train the best-performing model on the second dataset, this time detecting either anger or sadness. The model performance (accuracy and F1-macro) on the test set of both datasets can be found in Table 2.

Dataset	**Accuracy**	**F1-macro**
Anger & Joy	0.7762	0.7479
Anger & Sadness	0.8032	0.7881

Table 2: Dataset 1, *Anger & Joy*, and Dataset 2, *Anger & Sadness*. Accuracy and F1-macro scores on the test set for the best performing CNN model.

4 Conclusions

Our first observation was that slowing down the learning did not particularly improve the quality of the prediction. It actually seemed like the model was overfitting the train data in this case, since during our experiments the validation loss for each for each batch was increasing after one point. On the other hand, increasing the learning rate did not help much either since we noticed that the validation loss was jumping from higher to lower values and vice versa between batches. Therefore the best learning rate was 0.01.

Regarding the filter size and the number of the filters, we realized that making the model too complex by increasing these parameters too much did not add any value to the prediction quality. This likely happens because the task at hand is simple and straightforward enough, such that it doesn't require a very complicated architecture.