Data Wrangling Coursework

40087141

Abstract. Sentiment analysis is a widely covered topic in the study of machine learning, and state of the art models are often evaluated on simple binary classification problems. The goal of this project is to outline the key decisions and their merits, in pursuit of reasonably performing machine learning algorithm for text classification. The model will be motivated by the literature covered in the field, and the results evaluated by the local successes and improvements. State-of-the-art models will not be replicated, but considered (as decision trees are not state-of-the-art, they will not be considered).

Initial Considerations

Natural language and sentiment analysis problems are largely domain specific. As such the 'best' solution or combination of solutions will be different from domain to domain.

From an analytical point-of-view, aspects of the input(s) should be considered on their importance to learning. For instance, two popular neural network choices of CNN and RNN apply different learning mechanics. Where convolutions nets will detect and extract features, like negative or abusive terms; and recurrent networks identify text in length, hoping to 'understand' context[1]. As in sentiment classification, the whole length of text is as important as its parts, it makes sense, in theory, to utilise this learning structure for this task. Results indicate this to be true[2].

Although still considered a recurrent neural network, an LSTM is a modified version of an RNN purposed to better remember past data, and solving some issues RNNs had like vanishing gradients[3]. It is also accepted that LSTM gives the most controllability and thus, better results. but also comes with more complexity and operating cost[4]. A further enhancement on traditional unidirectional LSTM models, is to introduce an ability to also process future inputs, allowing contexts to be learned from forward and reverse. Bidirectional layers are found in recent state-of-the-art algorithms, and can increase accuracy[5].

Considering the model architecture, CNNs are much more complex than LSTMs, with much more options of layer types as well as number of layers and neurons. Not only is it more difficult from a design perspective but given optimisation in its many facets is of NP-Hard complexity, it adds more search considerations, therefore for this task using an (Bidirectional) LSTM makes the most sense.

Pre-Process

Text Representation

Word embeddings consistently outperform other methods of representation, and larger dimensionality in those word embeddings are more useful in tasks that require semantic analysis[7]. Given the magnitude of parameters, and the relatively small corpus size, pre-trained word embeddings can greatly enhance the successful learning process, and overall accuracy of models[8]. Both Word2vec and GloVe were evaluated after a best model was found, with Glove being better by ~3% See **Fig 6**.

Pre-trained weights

A common tactic for increasing accuracy and reducing learning time is the use of pretrained vectors on large corpuses of text, this encodes some association into the words and the network "understands" them, as the sequences are fed in. These vectors' dimensionalities are typically of a magnitude larger. One issue with using pretrained vectors is the occurrence of unknown tokens. There is no outlying best practice for dealing with these tokens. For this algorithm, thy are removed from the vocabulary. Taking an exert shows that the tokens not found in the wordvectors model are not semantically inclined and often nonsensical. Where sense can be derived, it is often that of an actor or director's name, in which bias to these words being positive or negative should be discouraged. See **Table 1** in appendix for sample output.

Model Preparation

To keep the model architecture somewhat simple and searchable, decisions can be made before any model is compiled. For instance,

- **Activation:** Using an activation function that ensures the final value will be valid, the sigmoid function outputs between 0 and 1 [9].
- **Loss:** calculating the loss between two possible values with binary_crossentropy.
- **Optimiser:** adam is often the most stable and accurate[10].

Searchable Hyperparameters

Using the Keras tuner library and a Bayesian Optimization search function, we can search the selected hyperparameters to see which yields the greatest output. Bayesian Optimization was chosen for its efficiency of manoeuvring the search space[11].

- **Depth:** the number of layers both in Dense and Bidirectional has an impact on the performance of the model. Where in particular, stacking LSTM layers, each layer performs some part of the task to be solved[12]. The search allows for 1 to 4 of each layer.
- **Neurons:** there is unfortunately no way of knowing the correct number of neurons to have in any layer, to allow a more reasonable search space, the tuner looks only searches in the LSTM neurons at between *5* and *90.* The number of dense layer neurons is not searched, but calculated in conjunction with this as $\frac{n}{i+2}$ where n is the number of LSTM neurons and I is the number of additional dense layers (starting at zero). This is to allow the architecture to converge and not have any cases of extreme loss of data (often seen when there is only a single dense layer and neuron).
- **Other:** epochs, batch size, and learning rate may be explored after the best performing model is selected, it is noted that learning rate does have an impact on not only the learning speed but the performance/accuracy of the model.

For the purposes of the search, the number of epochs, batch size, and learning rate is kept consistent, at *10*, 16, and 0.01. Early stopping is also used throughout as a mitigation of over and underfitting. The search monitors a score of validation accuracy, the use of a validation set in this case and in general is a mitigation on overfitting. The results are documented in the appendix as **Table 2** (note: as the layer is bidirectional, the number of neurons is doubled).

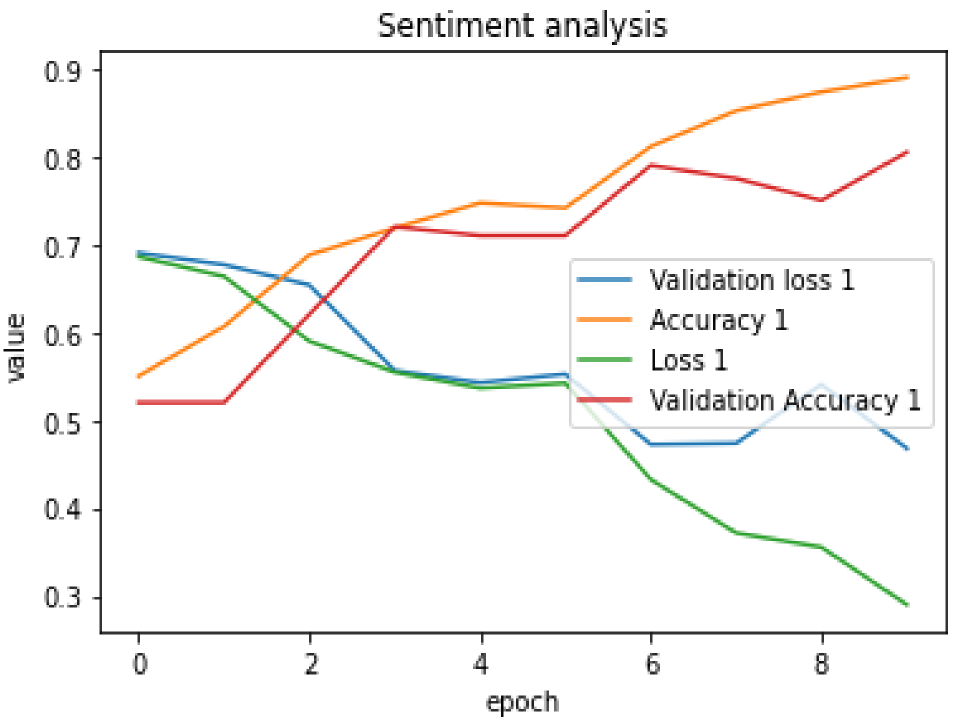
Results

Some local running and tweaking resulted in a model (displayed in F**ig. 2 and 3**) final best test accuracy score of 84.25%.

```
train_accuracy = 0.9242857098579407
train_loss = 0.21353447437286377

val_accuracy = 0.8050000071525574
val_loss = 0.4677537679672241

test_accuracy = 0.8424999713897705
test_loss = 0.42621028423309326
```



Conclusion

Given the resources of time computation and brain power, it is unfeasible to expect outstanding results from a simple model architecture. The best process is to plan, analyse the technology that is both, at hand, and computationally reasonable.

Also considering the effect random distribution plays an important part in the performance of a model, and each iteration is in pursuit of improving the previous, success is based on the final accuracy value. Considering the list of papers published has 87.15% as the next highest, 84.25%, it's much closer than would have been expected[14]. A major criticism is how temperamental each train of the same model is, unlike a CNN counterpart, it seems like this model relies on a large amount of chance to find a high scoring result, furthermore, tweaks to the architecture sometimes have a drastic results of stunting the learning totally (it is not known if this is avoidable or standard practice), or making it too prone to overfitting, which can be somewhat mitigated but changing hyperparameters.

References

[1] Ian Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[2] W. Yin, K. Kann, M. Yu, and H. Schütze, “Comparative study of CNN and RNN for natural language processing,” *arXiv*. 2017.

[3] “Understanding RNN and LSTM. What is Neural Network? | by Aditi Mittal | Medium.” https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e (accessed Apr. 09, 2021).

[4] “Simple RNN vs GRU vs LSTM :- Difference lies in More Flexible control | by Saurabh Rathor | Medium.” https://medium.com/@saurabh.rathor092/simple-rnn-vs-gru-vs-lstm-difference-lies-in-more-flexible-control-5f33e07b1e57 (accessed Apr. 09, 2021).

[5] B. Jang, M. Kim, G. Harerimana, S. U. Kang, and J. W. Kim, “Bi-LSTM model to increase accuracy in text classification: Combining word2vec CNN and attention mechanism,” *Appl. Sci.*, vol. 10, no. 17, 2020, doi: 10.3390/app10175841.

[6] Google, “Text classification with an RNN | TensorFlow Core,” 2020. https://www.tensorflow.org/tutorials/text/text_classification_rnn (accessed Apr. 09, 2021).

[7] S. Ruder, P. Ghaffari, and J. G. Breslin, “A hierarchical model of reviews for aspect-based sentiment analysis,” in *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, 2016, pp. 999–1005, doi: 10.18653/v1/d16-1103.

[8] “Pretrained Word Embeddings | Word Embedding NLP.” https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/ (accessed Apr. 09, 2021).

[9] Keras, “Layer activation functions,” 2020. https://keras.io/api/layers/activations/ (accessed Apr. 09, 2021).

[10] D. Mack, “How to pick the best learning rate for your machine learning project,” *Medium*, 2018. https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2 (accessed Apr. 09, 2021).

[11] “Comparison of Hyperparameter Tuning algorithms: Grid search, Random search, Bayesian optimization | by Lavanya Gupta | Analytics Vidhya | Medium.” https://medium.com/analytics-vidhya/comparison-of-hyperparameter-tuning-algorithms-grid-search-random-search-bayesian-optimization-5326aaef1bd1 (accessed Apr. 09, 2021).

[12] M. Hermans and B. Schrauwen, “Training and analyzing deep recurrent neural networks,” in *Advances in Neural Information Processing Systems*, 2013.

[13] A. Bäuerle, C. Van Onzenoodt, and T. Ropinski, “Net2Vis: Transforming deep convolutional networks into publication-ready visualizations,” *arXiv*. 2019, Accessed: Apr. 09, 2021. [Online]. Available: https://a13x.io.

[14] “IMDb Benchmark (Text Classification) | Papers With Code” https://paperswithcode.com/sota/text-classification-on-imdb (accessed Apr. 09, 2021).

Appendix

Figure 1 Keras/Tensorflow Example of Bidirectional LSTM[6]

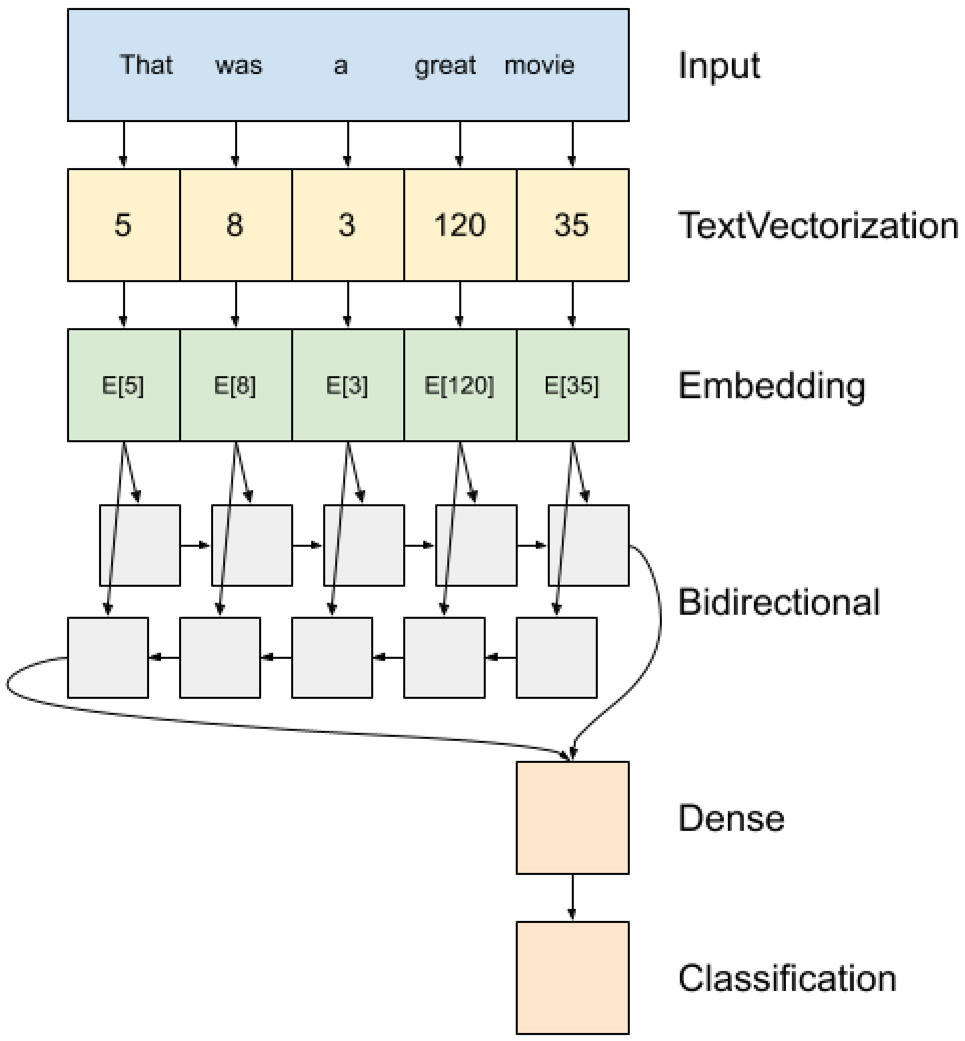


Figure 2 Sample of words not in word vector

"ru'afro"	'xref'	"kersey's"	"bromides'"	"hirst's"
'fallingout'	'requisitive'	'armynavyair'	"bronx's"	'simpithize'
'ultraserious'	"gugino's"	'typecasted'	"jason's"	'rougly'
'sweetfaced'	'honsou'	'atkine'	"strangers'"	"necklace's"
"beatles'"	'braindamaged'	'gyllenhall'	"gattaca's"	'firetaming'
"unabomber's"	'japanimation'	'evars'	"atlanta's"	'illadvised'
'wellpublicized'	'maleegostroking'	'climact'	'matteroffact'	"welsh's"
"moresco's"	'demandingly'	'popin'	'mtvinfluenced'	'ooky'
'oneupped'	"huston's"	'alreadybeautiful'	"huston's"	"buehler's"

Figure 3 Results summary of Search optimisation

```
neurons =  25
LSTMlayers =  2
Denselayers =  1
Best Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 220, 300)          600300
_________________________________________________________________
bidirectional (Bidirectional (None, 220, 50)           65200
_________________________________________________________________
bidirectional_1 (Bidirection (None, 220, 50)           15200
_________________________________________________________________
bidirectional_2 (Bidirection (None, 50)                15200
_________________________________________________________________
dense (Dense)                (None, 12)                612
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 13
=================================================================
Total params: 696,525
Trainable params: 96,225
Non-trainable params: 600,300
```

```
Results summary
|-Results in S:\keras_tuning_biLSTM\kerastuner_bayesian_poc
|-Showing 10 best trials
|-Objective(name='val_accuracy', direction='max')
Trial summary
|-Trial ID: 121da12673f75ef6f482f8af709a1f53
|-Score: 0.7950000166893005
|-Best step: 0
Hyperparameters:
|-Denselayers: 1
|-LSTMlayers: 2
|-neurons: 25
Trial summary
|-Trial ID: 37fffd45c1c4eb1d91eb063d113435b5
|-Score: 0.7649999856948853
|-Best step: 0
Hyperparameters:
|-Denselayers: 2
|-LSTMlayers: 2
|-neurons: 5
Trial summary
|-Trial ID: afa214d2819fa56b61f98dbb91a00034
|-Score: 0.7649999856948853
|-Best step: 0
Hyperparameters:
|-Denselayers: 2
|-LSTMlayers: 1
```

Figure 4 Keras output of plot_model (with dropout layers)

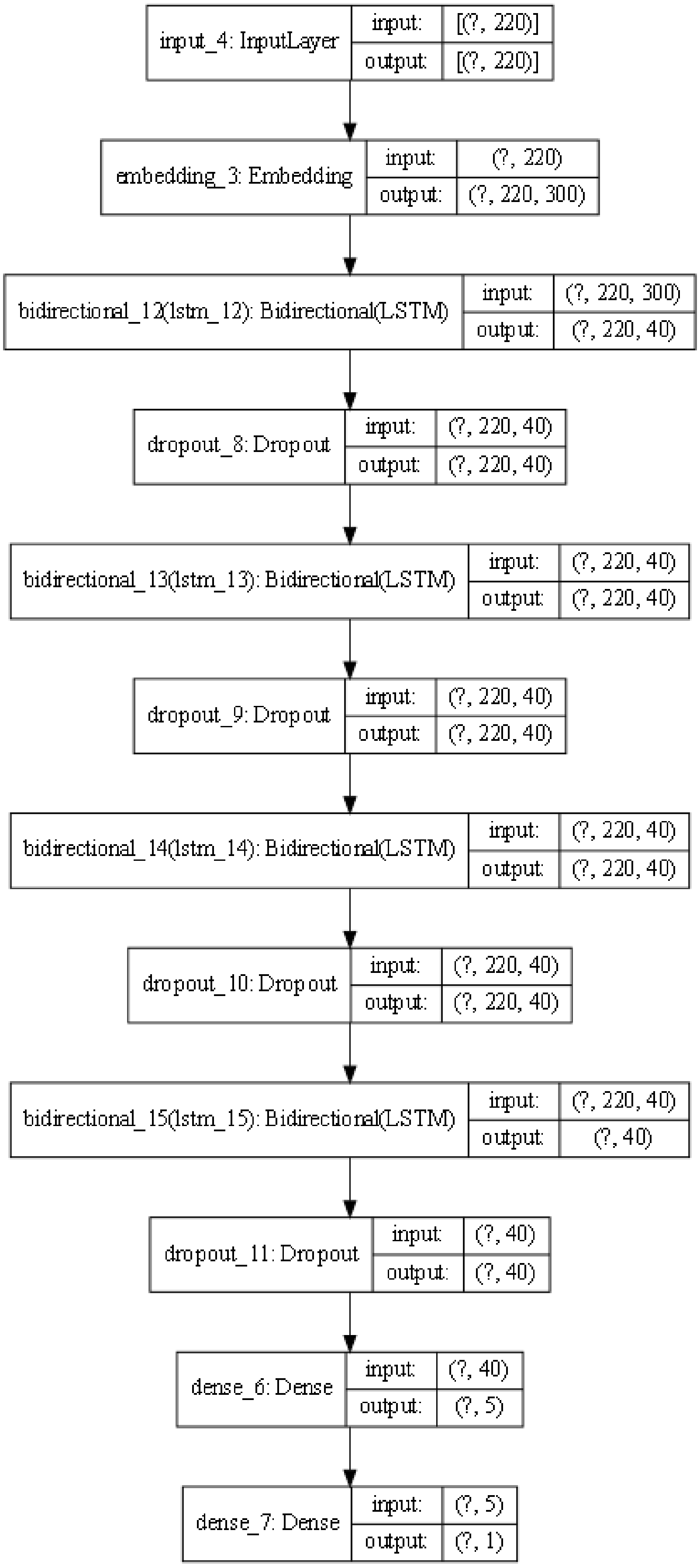


Figure 5 Net2Vis NN graph, and accompanying key[13]

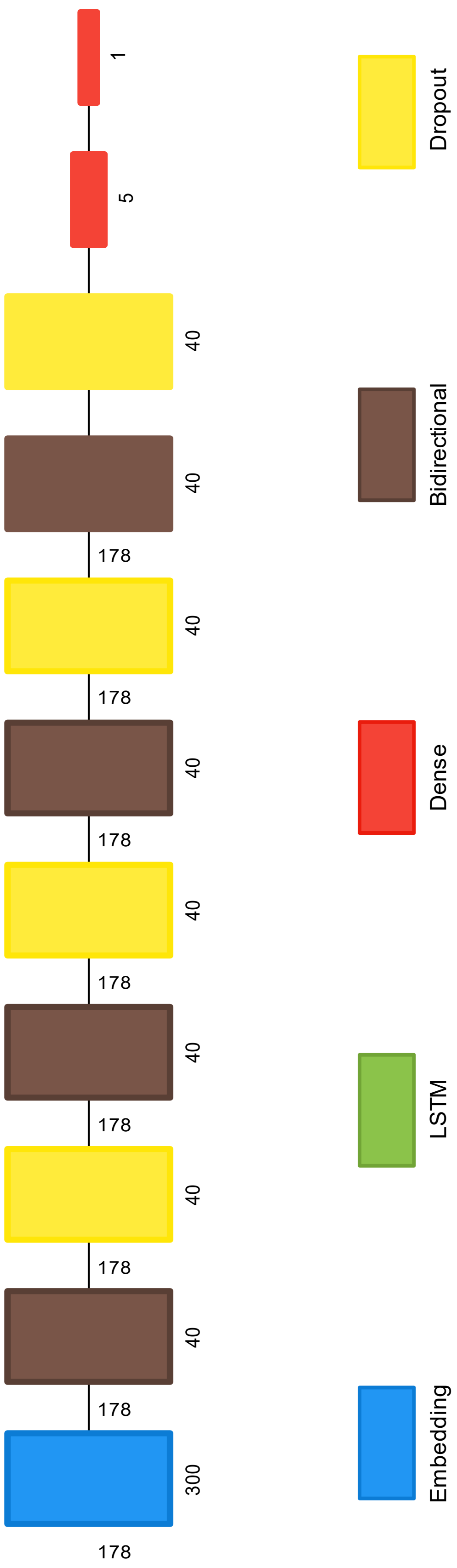


Figure 6 Comparison of GloVe and Word2Vec

GloVe = 1
Word2Vec = 2

train_accuracy1 = 0.9900000095367432
train_accuracy2 = 0.9785714149475098

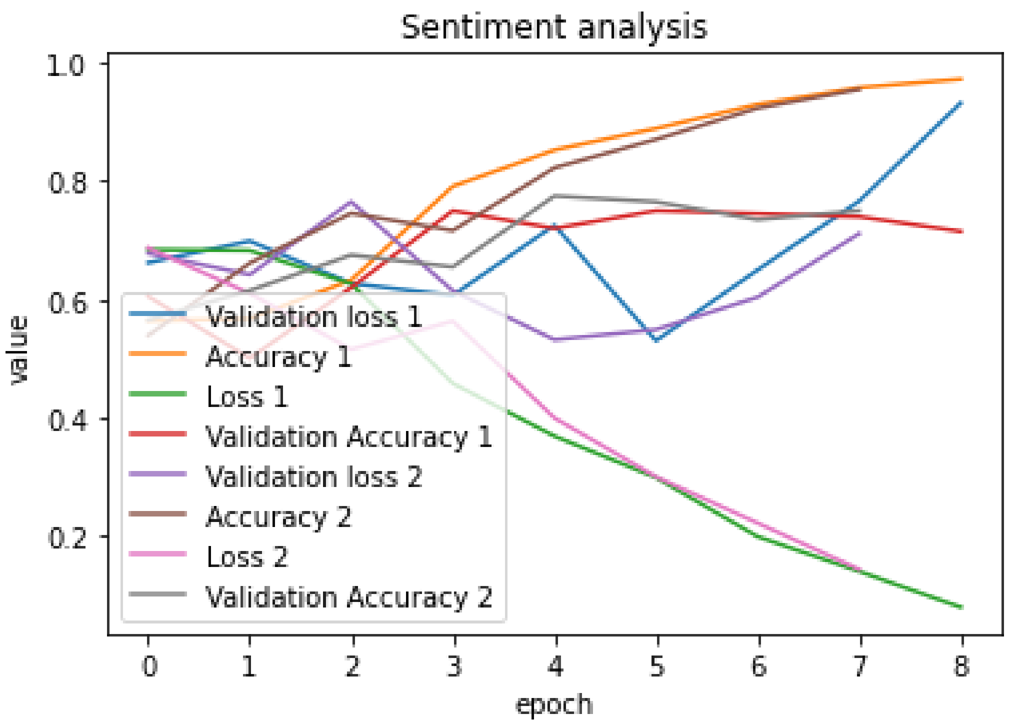
train_loss1 = 0.03643043711781502
train_loss2 = 0.0844436064362526

val_accuracy1 = 0.7149999737739563
val_accuracy2 = 0.75

val_loss1 = 0.9325019121170044
val_loss2 = 0.7110413908958435

test_accuracy1 = 0.8374999761581421
test_accuracy2 = 0.8050000071525574

test_loss1 = 0.6189182996749878
test_loss2 = 0.5440778732299805


Sentiment analysis
value
epoch
Validation loss 1
Accuracy 1
Loss 1
Validation Accuracy 1
Validation loss 2
Accuracy 2
Loss 2
Validation Accuracy 2