Predicting the Conditions using RNN and Semantic Relationships between them

Sandesh Reddy Pendyala

Graduate Student, Department of Computer Science

Z1859176

Sindhusha Devi Parimi

Graduate Student, Department of Computer Science

Z1855951

ABSTRACT

In this project we will be working on Clinical Trials Dataset to build two Machine Learning Models that are able to take a new document as input containing Title and Abstract and then output the type of specialized research it belongs to. This Model would enable us to retrieve other related research without having to manually search for it from the corpus.

KEYWORDS AND PHRASES
Tensorflow, model building, NLP, Accuracy, RNN, LSTM, Unfreezing

1 INTRODUCTION

1.1 ULMFiT

ULMFiT stands for Universal Language Model Fine-tuning for Text Classification [1] [2] and is a transfer learning technique that involves creating a *Language Model* that is capable of predicting the next word in a sentence, based on unsupervised learning of the WikiText 103 corpus. The ULMFiT model uses multiple LSTM layers [3], with dropout applied to every layer. LSTM is a specific kind of Recurrent Neural Network [4] with Long Short-Term Memory units capable of learning long-term dependencies. Remembering information for long periods of time is practically their default behavior [5].Once we built a language model, we can now train this model on our corpus. This model once trained can now predict which class a new input document corresponds to.

1.2 Universal Sentence Encoder

This project implements Google’s Universal Sentence Encoder (USE) [6] [7], a model that is used to encode sentences onto an embedding vector which is then used for transfer learning for other NLP tasks. This is primarily done by building the model on a very large corpus of sentences which can form semantic relationships when returning the vector.

In the Encoder itself we would be using the Transformer Architecture [8] although the run time is high we would be better served by the corresponding increase in accuracy. This is a network architecture based solely on attention mechanisms. This network also has an encoder and a decoder where the encoder maps an input sequence of symbol representation to a sequence of continuous representations. For every element the decoder generates an output sequence. The attention function maps a query and a set of key-value pairs to an output where all the involved elements are vectors and the output is a weight sum of the vectors.

The model would be trained on Keras TensorFlow [9]. TensorFlow is a familiar tool for machine learning users. TensorFlow supports a variety of applications, with a focus on training and inference on deep neural networks. It has high performance matching the best in the industry. It provides unique approach allows monitoring the training progress of your models and tracking several metrics.

2 ULMFiT

2.1 Data Pre-Processing

For the first step we would extract all the Titles and Conditions from the dataset and store them in a csv. Moving this file onto a Pandas DataFrame [10] would help in speeding up the processing to a considerable degree. We will then tokenize the words , Pos tag them and lemmatize them.

ULMFiT's language building we need a data set that can be parsed by SKlearn's test_train_split function [11]. This meant that we have to first go through the data set manually and count the classes which have a statistically large enough chance that they will be split into train and test splits with at least one element of each class present in both sets.

For this we had to remove those classes which had less than 40 elements present in the data set. Any classes that had lesser than this value were all being put into just 1 data set thereby invalidating the split function.

2.2 FastAI data bunch

Creating a data bunch automatically results in pre-processing of text, including vocabulary formation and tokenization.

- TextLMDataBunch creates a data bunch for language modelling. In this, labels are completely ignored. Instead, data is processed so that the RNN [12] can learn what word comes next given a starting word.
- TextClasDataBunch sets up the data for classification. Labels play a key role here. We can also set the batch size for learning by changing the bs parameter.

2.3 Create and Train the Language Model

We create a language model based on the AWD-LSTM model [13]. This is done by passing the learning data data-bunch to language_model_learner function from FastAI. This trains the model on the training data. We also include dropout here as a regularization [14] so as to decrease overfitting of the model being trained.

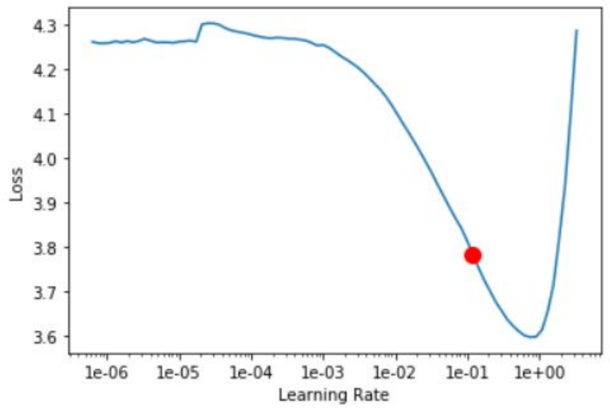


Figure 1: Learning Rate vs Loss for few epochs for language model

Now we also have to find the optimal learning rate at which the model has to run. This is done by calling the lr_find function from FastAI. It is a fast.ai built in function which runs a few epochs on the model to plot loss and inturn calculate the minimum gradient, which can be seen in Fig.1. Once we have this learning rate we then pass this as an argument to fit_one_cycle function to begin training of the language model. The language model encoder can be saved using *learn.save_encoder* so that we can load it later in our classifier.

2.4 Using the Language Model to Train the Classifier

Once we have completed training the language model we now move on to training the classifier. Creating and training the classifier is pretty similar to training the language model. First, we created a *text_classifier_learner* using the data bunch that was generated by the function *TextClasDataBunch.* We also pass the AWD_LSTM architecture and a dropout rate [15,16] of 0.5 as parameters into this. Then the language model encoder which is saved in the earlier step is loaded here.

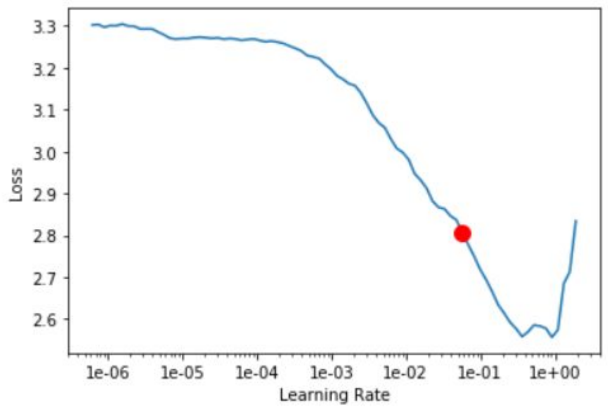


Figure 2: Learning Rate vs Loss for few epochs for classifier

The optimal learning rate at which model runs is to be found out again using the *lr_find* function. We use the concept of gradual unfreezing [17] to train the classifier. Initially, we start to train the last few layers , then go backwards and unfreeze and train the layer before those.
We used learn.recorder.plot_losses() to plot our loss functions of train and validation dataset. After unfreezing all the layers and training the model at low learning rate, our text classification model is built.

3 Universal Sentence Encoder

3.1 Data Pre-Processing

Similar to how we obtained the data for the previous model, here too we only need to consider the Conditions and the Title columns. But unlike ULMFiT we are able to work on those classes which have fewer than the necessary rows to train our model.

Once we had the necessary data we now have to modify it in such a way that the model we build later on can take it as input. Since we have categorical classes we have to modify them with 1-Hot encoding [18]. We have to encode both the training and test classes exactly the same so as to train and test successfully.

3.2 Encoder implementation

Now we have to explicitly declare every input we pass Embedder as a String. This is done so as it let the embedder return a 512 sized vector irrespective of the length and the data type of the input from the data set.

3.3 Semantic Relationship

When you convert the words “cat”, “animal”, “table” to their corresponding vectors, the words “cat” and “animal” will be located closer than the words “cat” and “table”.

Thus similar to the above example the vectors that are returned after embedding are closer to each other based on their meaning rather than just the euclidean distance between similar words in a sentence. This meant that those topics like cancer will automatically be clustered closer together instead with those topics whose wording is similar. This usage of Natural Language Processing [19] we hope increases the accuracy of the model significantly to justify its implementation.

3.4 Model Building

Once we have defined our embedder we can now build our model. For the task at hand , we have chosen to build a model that has dense layers with 256 nodes contained within them. These nodes have ‘relu’ as activation function [20].

After data processing we had calculated the number of unique classes which were 17 in total. Thus for the output layer we now need a dimensionality of 17 and since there are categorical classes we will use Softmax as the activation function for the output layer [21].

For compiling the model we had to choose a loss function which worked well with large number of classes in our data set and also accounted for the class imbalance. We had to choose between sparse_categorical_crossentropy and categorical_crossentropy. After extensive experimentation of various parameters we concluded that categorical_crossentropy [22] was better suited for the task at hand.

As for the optimizer we had option betweens Adam , RMSProp , SGD,Adagrad [23] [24]. And like earlier , we discovered in our experimentation that Adam gave the best results in the shortest time frame.

3.5 Training the Model

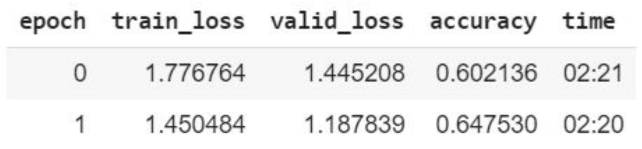
As with ULMFiT model training, we now pass all the parameters we have assembled so far and pass them into the model.fit function. From the hardware we had at our disposal we decided that training the model for 20 epochs gave us the results we wanted. As training the model for any longer was not giving any improvement in testing accuracy.

Once we are done training the model we save the weights of the model so that we can use them to test it out by predicting on information that the model has not seen before.

4 Results and Discussions

4.1 ULMFiT

Once we have our classification model, we will pass the number of epochs as parameter along with the minimum learning rate or optimal learning rate which was obtained with the function *recorder.min_grad_lr*.



epoch	train_loss	valid_loss	accuracy	time
0	1.776764	1.445208	0.602136	02:21
1	1.450484	1.187839	0.647530	02:20

Figure 3: Train and validation loss and accuracy for 2 epochs

In Fig. 3, we can see that our accuracy is 65% after 2 epochs for last few layers. The training and validation loss are comparable to each other at 1.45 and 1.18 respectively. Given that each epoch ran for approximately 2 minutes, the model ran pretty faster.

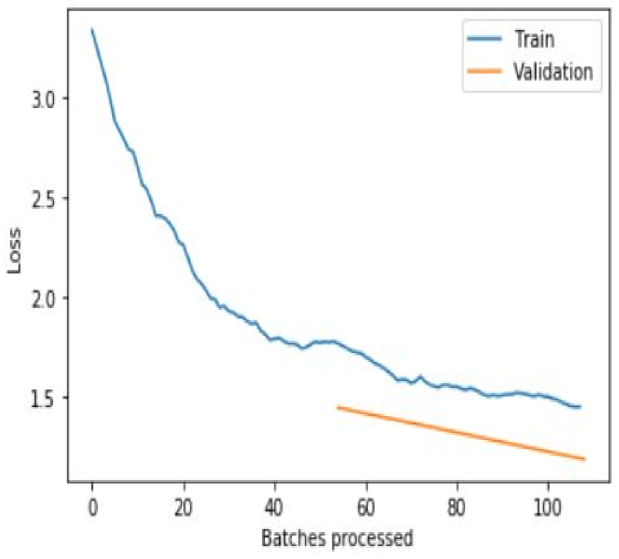
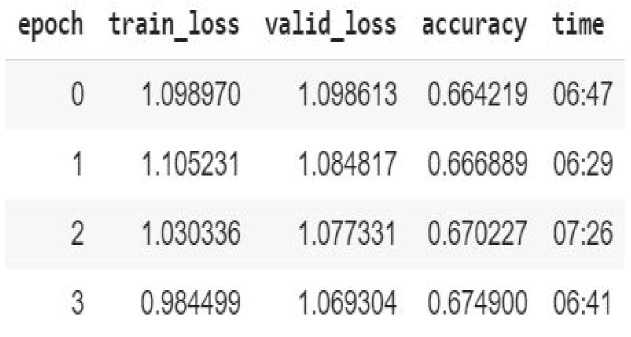


Figure 4: Loss vs Batches processed for Train and validation data before unfreeze

The Training and validation loss curves per batches processed can be seen in Fig. 4. The train and validation loss curves go almost parallel. There is no diverge happening in the curve which shows that no overfitting [25] has occurred.



epoch	train_loss	valid_loss	accuracy	time
0	1.098970	1.098613	0.664219	06:47
1	1.105231	1.084817	0.666889	06:29
2	1.030336	1.077331	0.670227	07:26
3	0.984499	1.069304	0.674900	06:41

Figure 5: Train and validation loss and accuracy for 2 epochs

Finally, we unfreezed all the layers and using *one_fit_cycle,* trained the model at low learning rate. In Fig. 5, we can see that our accuracy is 67% after 4 epochs for all the layers. The training and validation loss are comparable to each other at 0.98 and 1.06 respectively. Each epoch ran for approximately 7 minutes for all the layers when compared to 2 minutes per epoch before unfreeze of all layers.

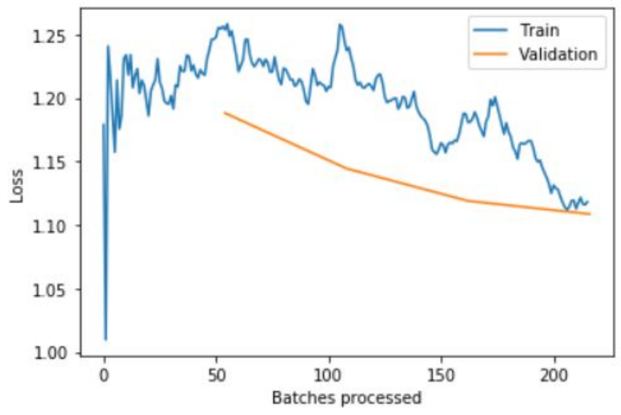
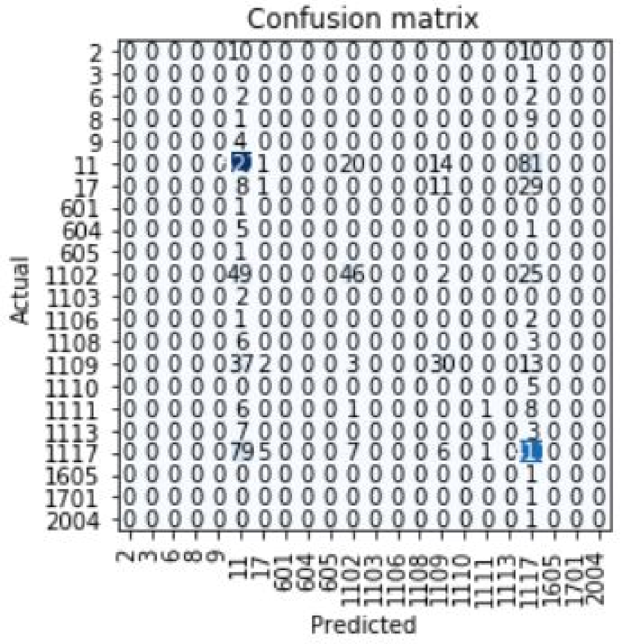


Figure 6: Loss vs Batches processed for Train and validation data after unfreeze

Confusion Matrix

We ran the text classification model on our unseen test data and predicted the results using *get_preds*. We plotted the confusion matrix [26] with the actual test labels and predicted test labels using *plot_confusion_matrix()*.



	2	3	6	8	9	11	17	601	604	605	1102	1103	1106	1108	1109	1110	1111	1113	1117	1605	1701	2004
2	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
8	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0
9	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	[illegible]	1	0	0	0	20	0	0	0	14	0	0	0	81	0	0	0
17	0	0	0	0	0	8	1	0	0	0	0	0	0	0	11	0	0	0	29	0	0	0
601	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
604	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
605	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1102	0	0	0	0	0	49	0	0	0	0	46	0	0	0	2	0	0	0	25	0	0	0
1103	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1106	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
1108	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
1109	0	0	0	0	0	37	2	0	0	0	3	0	0	0	30	0	0	0	13	0	0	0
1110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0
1111	0	0	0	0	0	6	0	0	0	0	1	0	0	0	0	0	1	0	8	0	0	0
1113	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
1117	0	0	0	0	0	79	5	0	0	0	7	0	0	0	6	0	1	0	[illegible]	0	0	0
1605	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1701	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
2004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 7: Confusion Matrix

The diagonal represents the correct predictions. Playing around the hyperparameters while training the model gave better results.

4.2 Universal Sentence Encoder

Once we have our classification model with Keras Tensorflow using universal sentence encoder hub, we ran the model on train data of 19111 and validation data of 4778 samples respectively for 20 epochs.

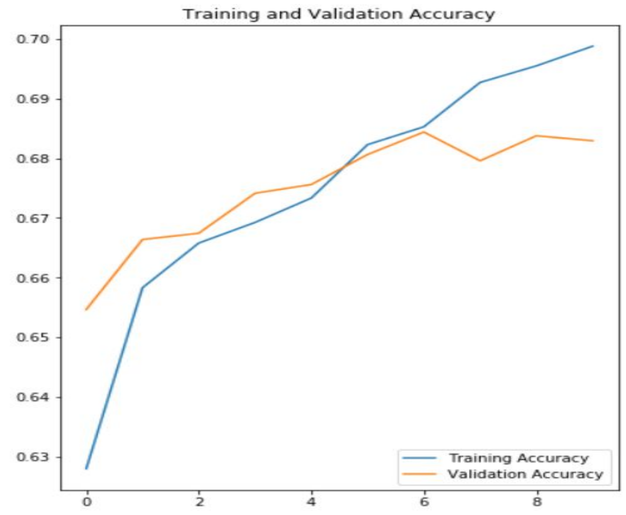


Figure 8: Training and Validation accuracy curves for different epochs

After the model has been trained, we get accuracy for both train and test data for each epoch which we plotted as shown in Fig. 8

It has taken approximately 7 minutes for each epoch and the training accuracy is 70% with a validation accuracy of 68%. This shows that the model doesn't overfit the data as the accuracies are in comparable range. The train loss is 0.85 and validation loss is 0.92 as seen in the below Fig. 9.

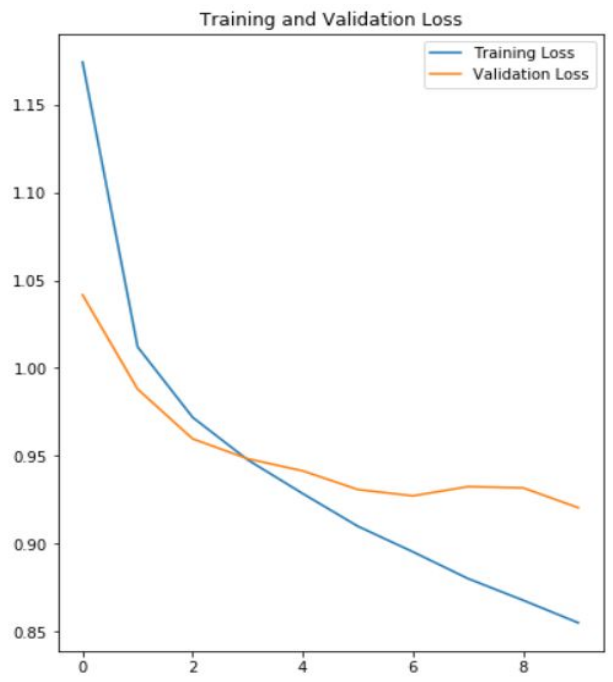


Figure 9: Training and Validation loss curves for different epochs

4 CONCLUSIONS

To conclude, using the FastAI library to train the text classifier was a good way. We can improve and get the best out of the model by playing around the training hyperparameters such as dropout rate, epochs and optimal learning rate. We achieved an accuracy of 67% with the ULMFiT model while the accuracy is 70% with the Universal sentence encoder model. The accuracy with USE can be improved with increasing the number of layers, playing with the dropout rate and check for better loss function or optimizers that suits the best for the model.

Our future work includes oversampling of the minority classes as one of the classes has high majority. This may improve the training of the model and in turn the accuracy.

Currently we used small sentences as input to our Universal Sentence Encoder to form vectors, we can try working with paragraphs in the future to increase the input scope of our model.

5 REFERENCES

1. Howard J, Ruder S. Universal Language Model Fine-tuning for Text Classification. arXiv [cs.CL]. 2018. Available: http://arxiv.org/abs/1801.06146

2. Hepburn J. Universal Language Model Fine-tuning for Patent Classification. Proceedings of the Australasian Language Technology Association Workshop 2018. 2018. pp. 93–96.

3. Sundermeyer M, Schlüter R, Ney H. LSTM neural networks for language modeling. Thirteenth annual conference of the international speech communication association. 2012. Available: https://www.isca-speech.org/archive/interspeech_2012/i12_0194.html

4. Mikolov T, Karafiát M, Burget L, Černock\`y J, Khudanpur S. Recurrent neural network based language model. Eleventh annual conference of the international speech communication association. 2010. Available: https://www.isca-speech.org/archive/interspeech_2010/i10_1045.html

5. Nowak J, Taspinar A, Scherer R. LSTM Recurrent Neural Networks for Short Text and Sentiment Classification. Artificial Intelligence and Soft Computing. Springer International Publishing; 2017. pp. 553–562.

6. Cer D, Yang Y, Kong S-Y, Hua N, Limtiaco N, St. John R, et al. Universal Sentence Encoder. arXiv [cs.CL]. 2018. Available: http://arxiv.org/abs/1803.11175

7. Conneau A, Kiela D, Schwenk H, Barrault L,

Bordes A. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. arXiv [cs.CL]. 2017. Available: http://arxiv.org/abs/1705.02364

8. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is All you Need. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, et al., editors. Advances in Neural Information Processing Systems 30. Curran Associates, Inc.; 2017. pp. 5998–6008.

9. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, et al. Tensorflow: A system for large-scale machine learning. 12th ${USENIX} Symposium on Operating Systems Design and Implementation ({OSDI}$ 16). 2016. pp. 265–283.

10. McKinney W. pandas: a foundational Python library for data analysis and statistics. Python for High Performance and Scientific Computing. 2011;14. Available: https://www.dlr.de/sc/portaldata/15/resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf

11. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. J Mach Learn Res. 2011;12: 2825–2830.

12. Lai S, Xu L, Liu K, Zhao J. Recurrent convolutional neural networks for text classification. Twenty-ninth AAAI conference on artificial intelligence. 2015. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/viewPaper/9745

13. Merity S, Keskar NS, Socher R. Regularizing and Optimizing LSTM Language Models. arXiv [cs.CL]. 2017. Available: http://arxiv.org/abs/1708.02182

14. Zaremba W, Sutskever I, Vinyals O. Recurrent Neural Network Regularization. arXiv [cs.NE]. 2014. Available: http://arxiv.org/abs/1409.2329

15. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15: 1929–1958.

16. Xiong J, Zhang K, Zhang H. A Vibrating Mechanism to Prevent Neural Networks from Overfitting. 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC). 2019. doi:10.1109/iwcmc.2019.8766500

17. Shleifer S. Low Resource Text Classification with ULMFit and Backtranslation. arXiv [cs.CL]. 2019. Available: http://arxiv.org/abs/1903.09244

18. Balikas G, Amini M-R. An empirical study on large scale text classification with skip-gram embeddings. arXiv [cs.CL]. 2016. Available: http://arxiv.org/abs/1606.06623

19. Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P. Natural Language Processing (Almost) from Scratch. J Mach Learn Res. 2011;12: 2493–2537.

20. Le QV, Jaitly N, Hinton GE. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. arXiv [cs.NE]. 2015. Available: http://arxiv.org/abs/1504.00941

21. Do CB, Ng AY. Transfer learning for text classification. In: Weiss Y, Schölkopf B, Platt JC, editors. Advances in Neural Information Processing Systems 18. MIT Press; 2006. pp. 299–306.

22. Zhang Z, Sabuncu M. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, editors. Advances in Neural Information Processing Systems 31. Curran Associates, Inc.; 2018. pp. 8778–8788.

23. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. arXiv [cs.LG]. 2014.

Available: http://arxiv.org/abs/1412.6980

24. Zou F, Shen L, Jie Z, Sun J, Liu W. Weighted AdaGrad with Unified Momentum. arXiv [cs.LG]. 2018. Available: http://arxiv.org/abs/1808.03408

25. Hawkins DM. The problem of overfitting. J Chem Inf Comput Sci. 2004;44: 1–12.

26. Ting KM. Confusion Matrix. In: Sammut C, Webb GI, editors. Encyclopedia of Machine Learning and Data Mining. Boston, MA: Springer US; 2017. pp. 260–260.