CSCE 479/879 Homework 2: Sentiment Analysis with Sequential Models

Mason Lien, Matthew Penne, Mrinal Rawool, Yves Shamavu

March 14, 2021

Abstract

This project focuses on sentiment analysis of the IMDB data set. The major techniques used are long short term memory (LSTM), attention layers, and regularizers. The project used manual tuning, grid search, randomized search and references for models. Sentiment analysis is fundamental for any machine learning task involving natural human speech. The best model found produced and accuracy of 92.56%.

1 Introduction

The IMDB dataset [3] is a commonly used data set for an introduction to sentiment analysis. The dataset consists of 50,000 reviews with 25,000 for training and 25,000 for testing. The learning goal is binary sentiment analysis, or more specifically to predict if a review is positive, producing a one, or negative, producing a zero.
The IMDB data set has input strings of varying lengths. These strings need to be converted to numerical vectors to be used for deep learning. This vectorization is done in a preprocessing layer that is discussed in Section 3.2. The output of the models in this assignment are a single neuron with a sigmoid activation function. This value of 0 to 1 represents how close the review is predicted to be negative, 0, or positive, 1.
Sentiment analysis is an integral part of natural language processing. This project categorizes the sentiment of a review into either positive or negative, or bianary classification. Sentiment analysis can be extended to detect the similarity between opinions. Sentiment analysis is important for any kind of human language feedback and is used in marketing, government, and medicine, etc.
The manual tuning the model resulted in the best validation accuracy of 92.56%. A ranking of literature results for this dataset is given in [1]. Out accuracy is similar to the first record paper [5], but falls short of the reported best accuracy of 97.4% in [5].

2 Problem Description

Th IMDB dataset is a large movie review dataset containing around 50,000 labeled movie reviews. [4]. The dataset is divided into 25,000 reviews for training and the rest for testing. This dataset is one of the widely used ones for sentiment analysis using a neural network architecture. The input to the model is the raw text from the movie review. The model processes each input to identify and determines whether the review is favourable/ positive/ 1 or unfavorable/ negative/ 0. Under the domain of Natural Language Processing, this particular learning problem falls under the category of sequence to vector transformation. That is, the model accepts an input sequence in the form of raw text and converts it into a vector representation that is used to determine the polarity of the review; i.e. the output.

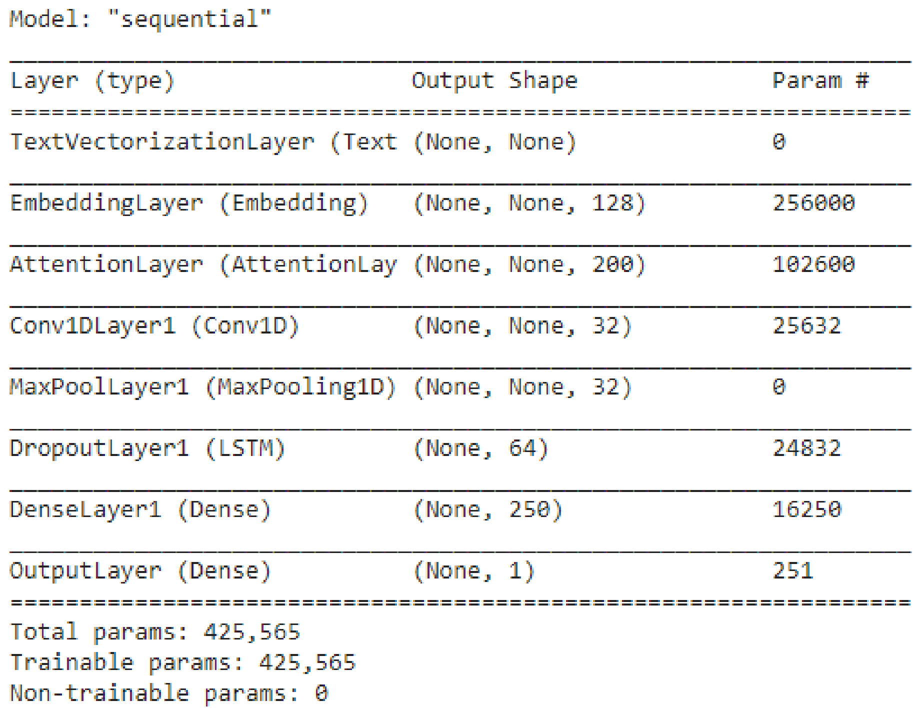
3 Approaches

The group used 2 methods for creating models: manual tuning and search methods. The manual tuning process involved change the layers and parameter ad hoc and then seeing if the changing improved the validation accuracy. The group also used search methods to explore a large combination of layers and hyperparameters to find the model with the best validation accuracy.

3.1 Manual Tuning

Below is an overview of the network architecture and design decisions undertaken for manual tuning. The model summary is shown in Fig. 1.

1. The Input layer
   The input layer has a shape of {} so it can take strings of different lengths.
2. The Vectorization layer
   Vectorization layer is created using Keras vectorization layer module with the vocabulary size set to 1000.
3. The Embedding layer
   The Keras embedding layer is used for this layer. The inputs are the sum of vocab size and num_oov_buckets, 1000 and 1000 respectively in this case, by the embedding size of 128.
4. The Attention layer
   The attention layer is similar to that derived in Hackathon 6. It uses a query layer with a 1D convolution of 100 filters and a kernel size of four. The value layer also uses a 1D convolution with 100 filters a with a kernel size of four.
5. Regularizer
   The regularizer takes the output of the attention layer through a 1D convolution layer with 32 filters and a kernel size of 4. This output then goes



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
TextVectorizationLayer (Text (None, None)              0

_________________________________________________________________
EmbeddingLayer (Embedding)   (None, None, 128)         256000
_________________________________________________________________
AttentionLayer (AttentionLay (None, None, 200)         102600
_________________________________________________________________
Conv1DLayer1 (Conv1D)        (None, None, 32)          25632
_________________________________________________________________
MaxPoolLayer1 (MaxPooling1D) (None, None, 32)          0
_________________________________________________________________
DropoutLayer1 (LSTM)         (None, 64)                24832
_________________________________________________________________
DenseLayer1 (Dense)          (None, 250)               16250
_________________________________________________________________
OutputLayer (Dense)          (None, 1)                 251
=================================================================
Total params: 425,565
Trainable params: 425,565
Non-trainable params: 0
```

Figure 1: Summary of Manual Tuning model

through a max pooling layer with a pool size of 2. This then goes through a LSTM layer with 64 units and a drop out rate of 0.2.

6. Output
   The output of the regularizer is then input into a dense layer with 250 neurons and relu activation. The output of this layer is then input into a single neuron with sigmoid activation. The output of this last layer can be rounded to determine if the prediction is 0, a negative review, or 1, a positive review.

7. Hyperparameters
   Adam optimization is used with a base learning rate of 0.001. The loss is calculated using binary crossentropy.

3.2 Search method

Below is an overview of the network architecture and design decisions undertaken for the search space for models that use Attention mechanisms and LSTM's.

1. The Input layer
   The input layer is declared with shape as {} since the model accepts input of variable dimensions.

2. The Vectorization layer
   Text vectorization is performed using **TextVectorization** module from Keras. The vocabulary size is set to 1024.

3. The Embedding layer
   This layer accepts vectorized representation of the raw input and transforms them into vectors of shape 1024×128 where 128 is the embedding size. The **mask_zero** attribute is set to allow the recurrent layers to ignore padding.

4. The Attention layer
   This model uses attention layer to evaluate the input (movie review) and map it to the output (sentiment). This is done by calculating a score between a query vector and and a value vector (which also doubles as the key vector). This score is converted into a linear scale using a softmax function. This process allows the input to be expressed in terms of the relative importance of its constituent terms. The query vector accepts the output of the embedding layer as its input. The query and value layers are individually made up of Conv1D layers of 100 units of filters of shape 4 each. Attention is calculated as the dot product of the query and the key.

5. Other Layers
   The attention layer is followed by at least one Conv1D, LSTM, and Dense

Table 1: Network Layer combinations search space

Type of Layer	Number of Layers	Number of units per layer
Dense	1	256
	2	256, 128
	3	256, 128, 64
	4	512, 256, 128, 64
LSTM	1	128
	2	128, 64
	3	512, 128, 64
Conv1D	1	32
	2	32, 32
	3	32, 32, 32

Table 2: Other Network Parameters for Search Space

Hyperparameters	Values
Optimizer learning rate	1e-5, 1e-4, 1e-3, 1e-2, 1e-1
Drop-out rate for LSTM	0.1, 0.2, 0.3, 0.4, 0.5
Vocabulary size**	1000, 1024, 2048
Embedding dimensions	20, 70, 128, 256

layer. Regularization is implemented by adding a MaxPooling layer between the convolution and LSTM layer(s). Please refer the Grid Search below for more details on these layers.

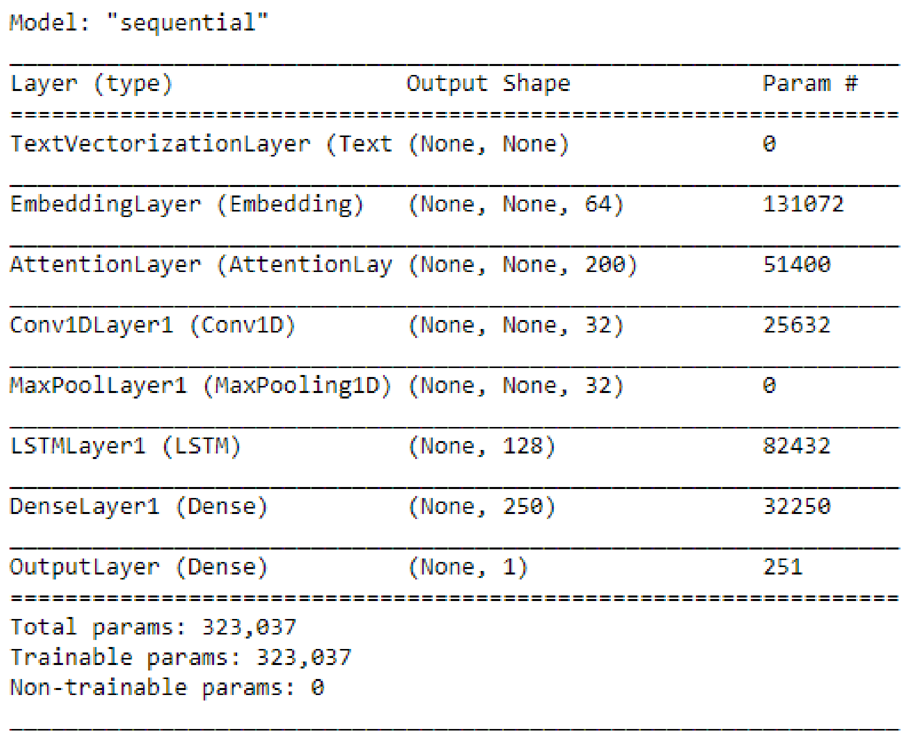
6. Model Hyperparameters
The model uses Adam optimizer with a base learning rate of 0.001. The loss is measured in terms of binary cross-entropy. The performance metric is accuracy.

7. Hyperparameter Tuning Techniques
To allow for tuning of hyperparameters, three methods have been implemented. The first one is using GridSearchCV from SkLearn. The second one is Random Search. The third one is inspired from the paper 'Attention is all you need " [6]. Grid Search and Randomized search use the same options for phyperparameters. The tables 1 and 2 summarize the different options in place for different network parameters.

The architectures of the three variations of models using the search method we developed for this task are represented in Figures 2-4.

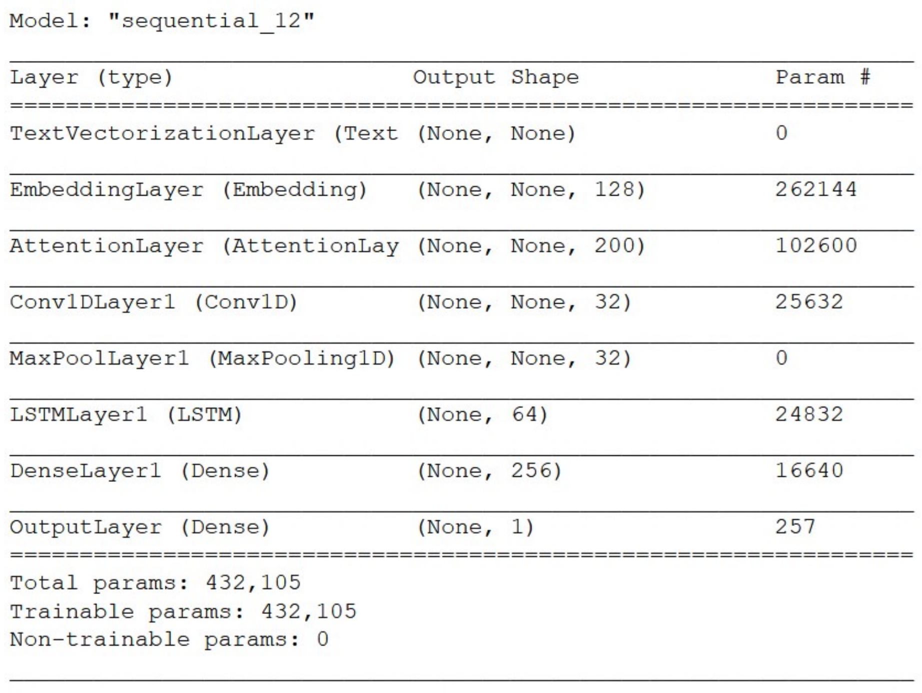
4 Experimental Setup

The **Train** split of the IMDB dataset is loaded using Tensorflow Datasets with the as_supervised attribute set to **True**. The dataset is further divided into



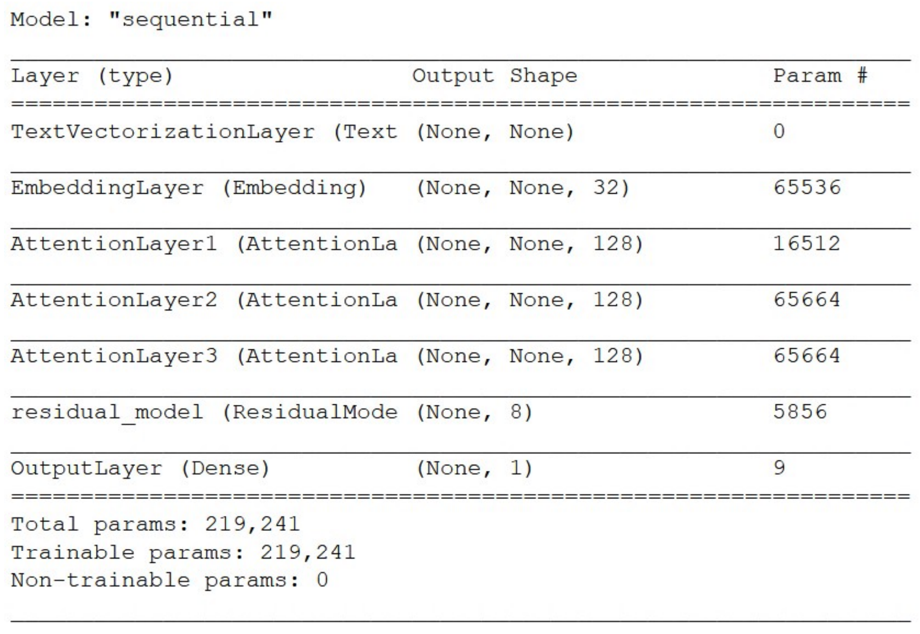
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
TextVectorizationLayer (Text (None, None)              0

_________________________________________________________________
EmbeddingLayer (Embedding)   (None, None, 64)          131072
_________________________________________________________________
AttentionLayer (AttentionLay (None, None, 200)         51400
_________________________________________________________________
Conv1DLayer1 (Conv1D)        (None, None, 32)          25632
_________________________________________________________________
MaxPoolLayer1 (MaxPooling1D) (None, None, 32)          0
_________________________________________________________________
LSTMLayer1 (LSTM)            (None, 128)               82432
_________________________________________________________________
DenseLayer1 (Dense)          (None, 250)               32250
_________________________________________________________________
OutputLayer (Dense)          (None, 1)                 251
=================================================================
Total params: 323,037
Trainable params: 323,037
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: This model was reported as a result of grid search by trying out a combination of parameters represented in Table 2, as we explained in homework one. It has 323,037 parameters, still with the custom attention layer.



```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
TextVectorizationLayer (Text (None, None)              0
_________________________________________________________________
EmbeddingLayer (Embedding)   (None, None, 128)         262144
_________________________________________________________________
AttentionLayer (AttentionLay (None, None, 200)         102600
_________________________________________________________________
Conv1DLayer1 (Conv1D)        (None, None, 32)          25632
_________________________________________________________________
MaxPoolLayer1 (MaxPooling1D) (None, None, 32)          0
_________________________________________________________________
LSTMLayer1 (LSTM)            (None, 64)                24832
_________________________________________________________________
DenseLayer1 (Dense)          (None, 256)               16640
_________________________________________________________________
OutputLayer (Dense)          (None, 1)                 257
=================================================================
Total params: 432,105
Trainable params: 432,105
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: This model was reported as a result of randomized search by randomly trying out a combination of parameters represented in Table 2, as we explained in homework one. It has 432,105 parameters.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
TextVectorizationLayer (Text (None, None)              0
_________________________________________________________________
EmbeddingLayer (Embedding)   (None, None, 32)          65536
_________________________________________________________________
AttentionLayer1 (AttentionLa (None, None, 128)         16512
_________________________________________________________________
AttentionLayer2 (AttentionLa (None, None, 128)         65664
_________________________________________________________________
AttentionLayer3 (AttentionLa (None, None, 128)         65664
_________________________________________________________________
residual_model (ResidualMode (None, 8)                 5856
_________________________________________________________________
OutputLayer (Dense)          (None, 1)                 9
=================================================================
Total params: 219,241
Trainable params: 219,241
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: This model was developed with the intention to test whether an adapted version of the residual network introduced in chapter 14 of the textbook by [2] could achieve better results, if combined with a stack of attention layers. The idea was inspired from the ”Attention is All You Need” [6] paper, in which Google researchers demonstrated that a model with a stack of attention layers could outperform one based on recurrent and convolutional layers. Our adaptation has 219,241 parameters and we kept a couple of LSTM layers (within the residual model custom layer), albeit with a very small number of units.

Table 3: Other Network Parameters for Search Method

Option	Description
1	Runs the base model. Currently, this model has hyper parameter values corresponding to the best performing combination that was discovered through hyper parameter tuning.
2	Performs grid search using Table 2
3	Runs a model with stacked attention layers
4	Performs randomized search using Table 2

train and validation dataset using a 90-10 split. The train data size is 22500 and the validation data size is 2500. Data is then shuffled and grouped into batches of 32.
The model is wrapped inside a Keras classifier that is used by GridSearchCV while searching for the best performing set of hyperparameters. The grid search is run for 30 epochs with the best performing model saved using ModelCheckpoint callback. Please refer tables 1 and 2 of section 3.2 for information about grid search parameter combinations. K-fold cross validation is implemented with K = 3. The code has provisions to run architecture variations by specifying options given in Table 3

4.1 Visualization of the Embedding Layer

To obtain a good understanding of how the embedding layer operates within a sequential model and how it trains an embedding for a given word in natural language, a visualization of the embedding space is helpful to explain classification of the IMDB movie review data. After the preprocessing of the movie review data discussed in 3.2, each word will be associated to a dimensional vector (embedding) that will be trained by the model. Word embedding gives the user the ability to represent words in a dataset that have similar encoding, where the values of the embedding are trainable parameters. For this sentiment analysis, The Keras embedding layer was selected for training. This layer can be understood as a "lookup" table that maps specific words to their embeddings. To visualize the embeddings, the embedding projector from tensorboard was used, where key words like "terrible" and "great" were selected to identify their closest neighbors as in Fig. 5.

This projector allows the user to generate graphical representations of high dimensional embeddings, through the use of principal component analysis (PCA), where PCA reduces dimensionality that is necessary for visualization. To understand which words are closest and furthest in proximity to each other, euclidean distance is calculated from the Cartesian coordinates of the euclidean space using Pythagorean theorem. The relationship can then be explained by the distance between a specific word and it's nearest neighbors. The smaller the

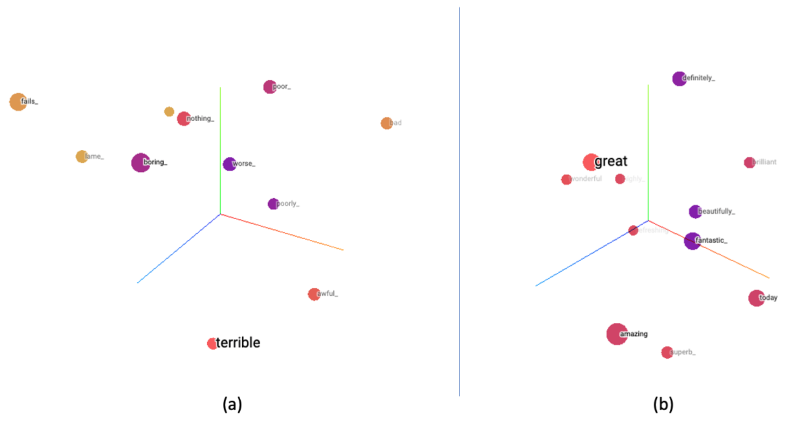


Figure 5: Principal Component Analysis (PCA) of Embedding Layer. The PCA (a) of the negative text ”terrible” shows the ten nearest points which signifies association. The PCA (b) of the positive text ”great” also shows the ten nearest points of words with association. The euclidean distance between the key word and the ten associative words is color coded, where purple (closest), red (middle), yellow (furthest) signify length of distance.

Table 4: Top Five Nearest Neighbors (NN) in Keras Word Embedding for ”Terrible” and ”Great”.

Rank	NN - Terrible	Euclidean Distance	NN - Great	Euclidean Distance
1	awful	0.208	powerful	0.142
2	poorly	0.340	beautifully	0.151
3	poor	0.362	rare	0.154
4	worse	0.417	recommended	0.155
5	horrible	0.425	gem	0.161

distance, the stronger the association between the two words. Table 4 gives an example of the 5 most closely related terms for both negative and positive reviews.

5 Experimental Results

5.1 Manual Tuning: Results

The manually tuned model reported a test accuracy of 92.56% and took roughly 12 minutes to train. The outputs of the model need to be rounded to 0 or 1 to determine if the review was negative or positive. The model produced a test accuracy of 92.56%. A confidence interval of 95% give an accuracy interval of ± 0.51%. So with 95% confidence the accuracy of the model is 92.09%-93.11%.

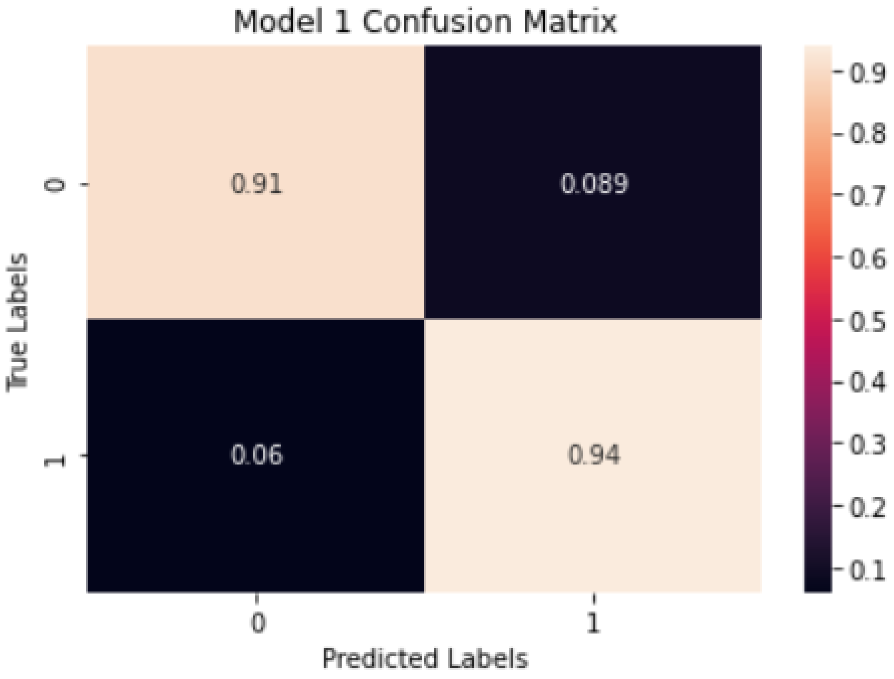


Figure 6: Confusion Matrix for manually tuned model

Table 5: Accuracy interval with 95% confidence.

Model	Accuracy	Range ±	Confidence Interval Accuracy
Grid Search	88.20%	0.63%	87.57%-88.83%
Randomized Search	57.84%	0.97%	56.87%-58.81%
Attention is all you Need	87.60%	0.65%	86.95%-88.24%

Table 6: Results of the four models ranked based on their performance on the validation set. error.

Model	Number of Parameters	Accuracy
Manually Tuned	425,565	92.6%
Grid Search	323,037	88.20%
Attention is all you Need	219,241	87.60%
Randomized Search	432,105	57.84%

5.2 Search Method: Results

The 95% confidence of the models are shown in Table 5. The grid search was the best model produced by grid search, but it is slightly worse than the manually tuned model. This shows that while grid search is a powerful tool, hand tuning still can produce better results. The rankings of the four models by validation accuracy is given in Table 6.

6 Discussion

6.1 Evaluation

The validation set comprised 2,500 records; it thus consisted of 10% of the entire training set. We conducted cross-validation through both grid search and randomized search. The parameters that were objects of the search are presented in Table 2. Grid search ran for more than eight hours. It was repeatedly interrupted before exhausting the search; however, it saved the best model after each epoch based on its performance on the validation set, thanks to the checkpoint callback. Randomized search was quick and exhausted the search in less than eight hours.

6.2 Performance

The model found through grid search (shown in Figure 2) outperformed by far the one found through randomized search (shown in Figure 3). Also, Table 6 shows the models ranked based on their performance on the validation set. Neither of the models were overfitting the training set, and the number of parameters did not correlate with the performance. The manually tuned model and the randomized search in Table 6 have almost the same number of parame-

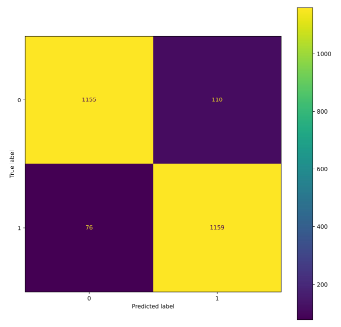


Figure 7: Confusion matrix for manually tuned model.

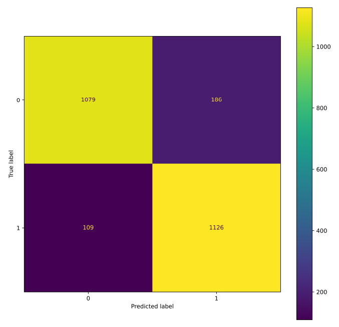


Figure 8: Confusion matrix for the best model found by grid search.

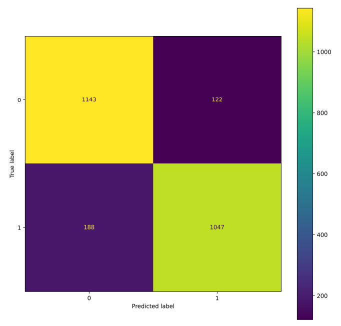


Figure 9: Confusion matrix for the model from "Attention is all you Need."

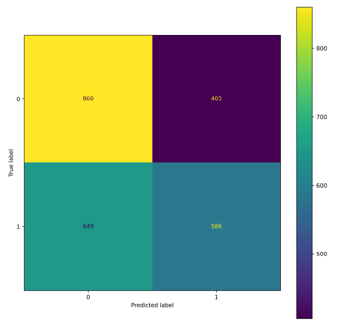


Figure 10: Confusion matrix for the best Randomized Search model.

ters and consist of the same layers. However, the difference in hyperparameters led to the randomized search poorest performance, yet it had more parameters overall. The confusion matrices for the four models are represented in Figures 7-10.

As shown in the confusion matrices, in a total of 2,500 records the manually tuned model correctly classified 2,314. The Grid Search model correctly classified 2,205. The "Attention is all you Need" correctly classified 2,190. And, lastly, the Randomized Search only got 1,446 correctly. We can see that the Randomized Search model mainly struggled with the positive label (1), just getting close to 50% accurately. This contrasts with the manually tuned model, which had more false positives compared to true positives.

7 Conclusions

In this assignment the group used attention layers, regularizers, and LSTMs for sentiment analysis of the IMDB dataset. Sentiment analysis is an import part of natural language processing and has many uses from business analytic to the medical field. This assignment showed that manually tuning a model still can produce the best results. Grid search and possibly randomized search are useful if enough time and computation power are available for a given project.

The group is doing a natural language processing project for the final, so this assignment was a good introduction to many of the problems that can occur. Preprocessing natural human speech can be difficult at times and this assignment made us understand some of the issues that can arise. Using embedding layers, attention mechanisms, and LSTM modules helped the group plan out how they will try to solve the natural language inference learning problem for the final project.

References

[1] Papers with code - imdb benchmark (sentiment analysis). URL: `https://paperswithcode.com/sota/sentiment-analysis-on-imdb`.

[2] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, 2019.

[3] IMDb. Imdb datasets. URL: `https://www.imdb.com/interfaces/`.

[4] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL: `http://www.aclweb.org/anthology/P11-1015`.

[5] Tan Thongtan and Tanasanee Phienthrakul. Sentiment classification using document embeddings trained with cosine similarity. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, 2019. https://doi.org/10.18653/v1/p19-2057 `doi:10.18653/v1/p19-2057`.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.