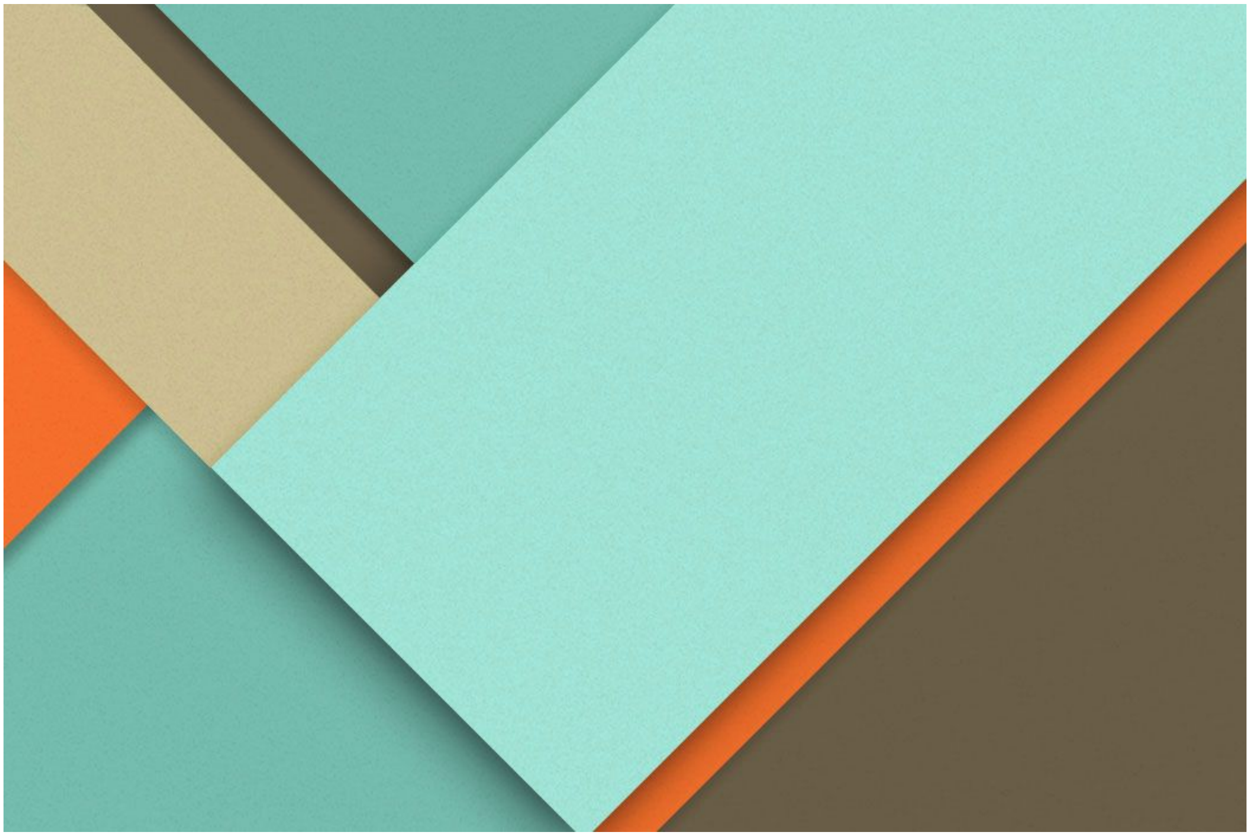


Machine Learning Nanodegree Capstone Project

Ekansh Gayakwad

22.03.2019

Short Answer Scoring(from Kaggle Competition)

Definition

Project Overview

The project selected is a real world problem faced by people which is evaluating essays or subjective type answers . I have tried to solve this problem using deep learning techniques which makes a smart model which can score subjective answers or essays when trained on some supervised data of essays which was already scored. The model make use of advance mathematics and computation to crack this problem.

For example I run a school and I am facing a problem in evaluation of students essays as they take a lot of time to evaluate and is a tedious job to do , if I want to correct some essay of this year's batch then I will take the corrected essays of previous year's batch and train a deep learning model on it then I will predict the scores of essays of this year's batch student which will save a lot of effort and time and will be out of any evaluation biases.

Teaching and evaluating is a challenge. Generally the critical thinking and analytical skills of student are evaluated using test where students get some score to measure these ability. For scoring subjective answers evaluators generally use two ways first searching keywords which do not asses the analytical writing of student and other being scoring according to the relatedness critical thinking of a batch. In both the ways of evaluating student's skills teachers use hand scored method which is tedious , commands considerable time and expense from public agencies .

Also the evaluation may be bias or depends on the mood of evaluator. So, because of those costs, standardized examinations have increasingly been limited to using "bubble tests" that deny us opportunities to challenge our students with more sophisticated measures of ability.Recent developments in innovative software to evaluate student written responses and other response types are promising. There are situations where computer has surpassed the manual evaluation.

(II reference)

Also on a personal note I think that evaluation is a critical job it sometimes decides the future humans are very likely to make mistake when give a long task like answer correction for example if an evaluator has to correct 100 different essays he/she might feel tired after evaluating some initial essays that can have good or bad impact on the evaluation of further essays but a software will be consistent throughout the evaluation.

Problem Statement

The problem is to classify typed essays (text format) into category of scores such that the scores are according to the question set given.This problem is classified as classification problem, input is an essay written by student and the goal is to score(integer score) it.

The main objective is to build a deep learning model which can do the task and on the same time also beat the provided benchmarks.

Metrics

Score predictions are evaluated based on objective criteria, and specifically using the quadratic weighted kappa error metric, which measures the agreement between two raters. This metric typically varies from 0 (only random agreement between raters) to 1 (complete agreement between raters). In the event that there is less agreement between the raters than expected by chance, this metric may go below 0. The quadratic weighted kappa is calculated between the automated scores for the responses and the resolved score for human raters on each set of responses. The mean of the quadratic weighted kappa is then taken across all sets of responses.

The kappa score is mentioned by the kaggle competition so I will use the same and I will be creating a deep neural net as the winner mentioned it performed well on cross-validation.

Kappa -score :

A set of essay responses *E* has *N* possible ratings, 1,2,...,*N*, and two raters, *Rater A* and *Rater B*. Each essay response e is characterized by a tuple *(e,e)*, which corresponds to its scores by *Rater A* (resolved human score) and *Rater B* (automated score). The quadratic weighted kappa is calculated as follows. First, and N-by-N histogram matrix *O* is constructed over the essay ratings, such that *O* corresponds to the number of essays that received a rating *i* by *Rater A* and a rating *j* by *Rater B*.

An *N-by-N* matrix of weights, *w*, is calculated based on the difference between raters' scores:

An *N-by-N* histogram matrix of expected ratings, *E*, is calculated, assuming that there is no correlation between rating scores. This is calculated as the outer product between each rater's histogram vector of ratings, normalized such that *E* and *O* have the same sum.

From these three matrices, the quadratic weighted kappa is calculated:

The Fisher Transformation is approximately a variance-stabilizing transformation and is defined:

$$z = \frac{1}{2}\ln\frac{1+\kappa}{1-\kappa}$$

Since this transformation approaches infinity as kappa approaches 1, the maximum kappa value is capped at 0.999. Next the mean of the transformed kappa values is calculated in the z-space. Finally, the reverse transformation is applied to get the average kappa value:

$$\kappa = \frac{e^{2z} - 1}{e^{2z} + 1}$$

If you have questions regarding the evaluation criteria, please refer to the help page.

(III reference)

Analysis

Data Exploration

For this competition, there are ten data sets. Each of the data sets was generated from a single prompt. Selected respones have an average length of 50 words per response. Some of the essays are dependent upon source information and others are not. All responses were written by students primarily in Grade 10. All responses were hand graded and were double-scored. Each of the eight data sets has its own unique characteristics. The variability is intended to test the limits of your scoring engine's capabilities.

The training data is provided in a tab-separated value (TSV) file containing the following columns:

- **Id**: A unique identifier for each individual student essay.
- **EssaySet**: 1-10, an id for each set of essays.
- **Score1**: The human rater's score for the answer. This is the final score for the answer and the score that you are trying to predict.
- **Score2**: A second human rater's score for the answer. This is provided as a measure of reliability, but had no bearing on the score the essay received.
- **EssayText**: The ascii text of a student's response.

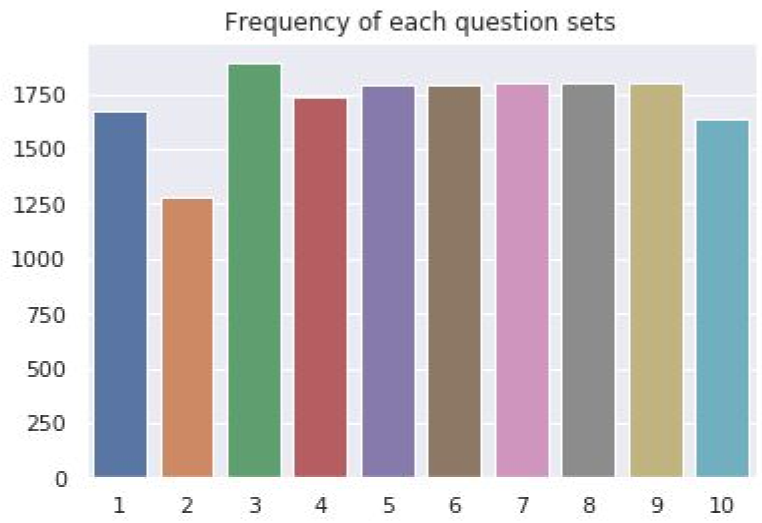
The total number of entries in training set is 17207 and in testing set is 5224

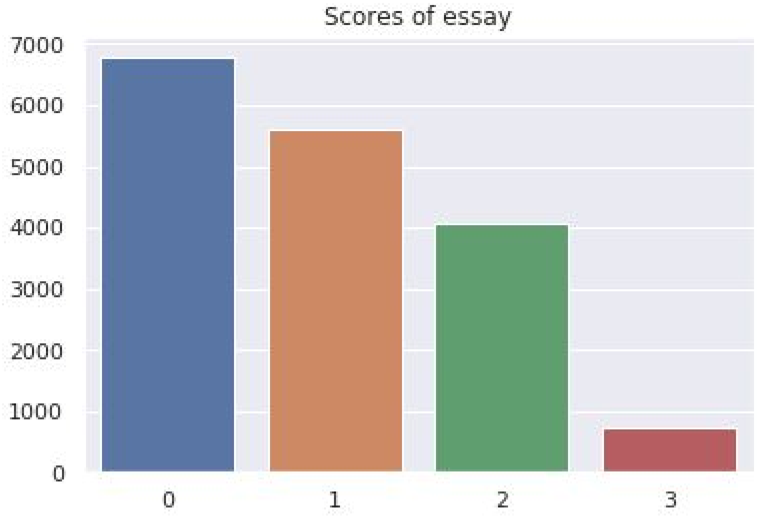
SET	TRAIN SIZE	TEST SIZE
1	1672	558
2	1278	426
3	1891	631
4	1738	580
5	1795	599
6	1797	599
7	1799	601
8	1799	601
9	1798	600
10	1640	584

Each of the set correspond to a different question for which a different answer is expected

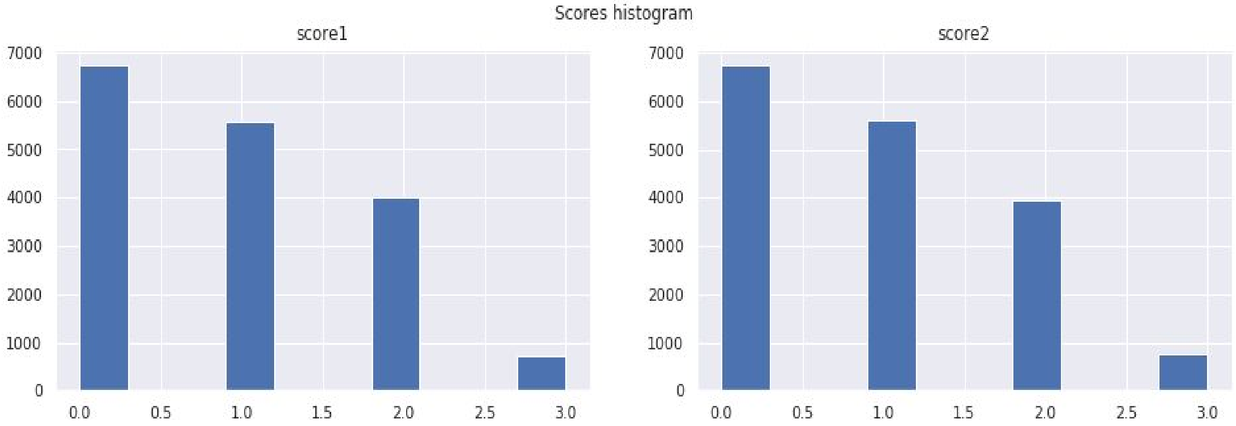
The scores of data or the classes(0,1,2,3) to be classified is highly unbalanced.

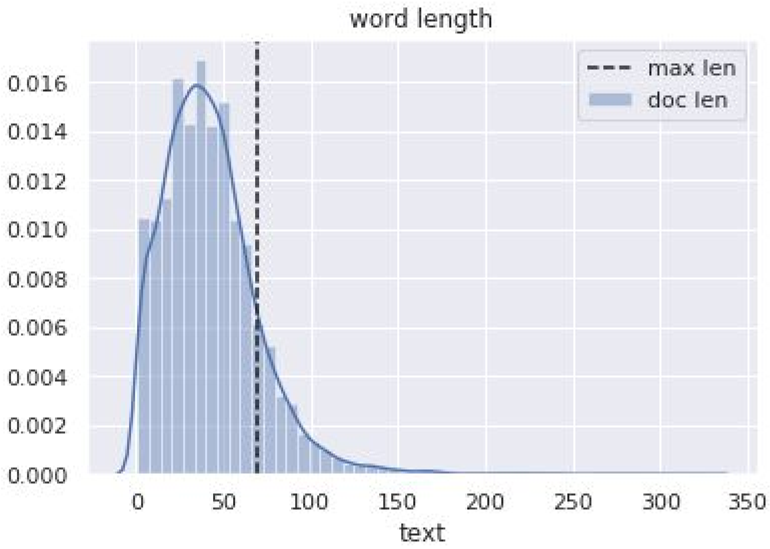
Exploratory Visualization





Note: The above score is the final score which is score 1 as mentioned in the kaggle competition





Algorithms and Techniques

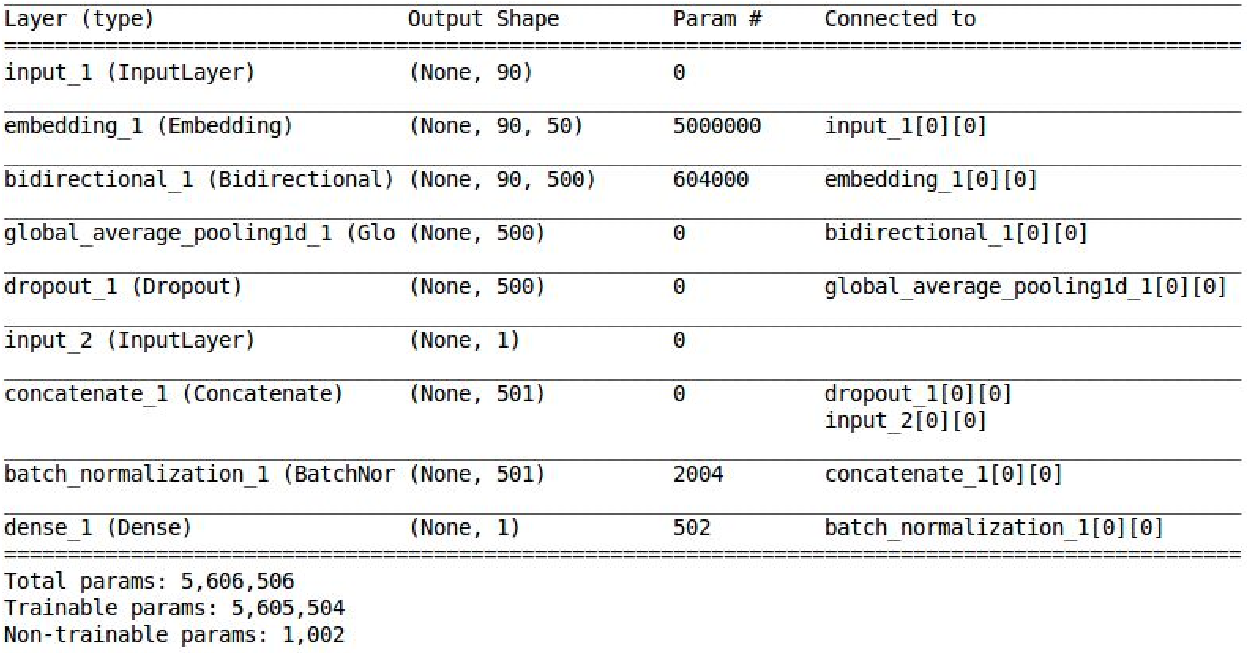
The classification model is a custom deep learning non-sequential model made using keras library on tensorflow backend

The model uses an embedding layer initially to vectorize the text data and then it is passed into a bidirectional LSTM(IV reference) layer as it is long essay so LSTM helps to retain the previous used sentences as well then the next layer is global average pooling layer followed by a dropout layer to avoid overfitting risk

Then the main part is to have a concatenation of the output of average pooling and set value by using this the model differentiates between different types of set question and rate them accordingly.

Then the concatenation is normalised and last dense layer with softmax activation is added to predict the score of the text.

To compile this model rmsprop optimizer is used and to avoid any kind of overfitting or undefitting I have used some part of training data as cross validation data and used it to save the best weights for the model using keras callbacks



```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 90)           0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 90, 50)       5000000     input_1[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 90, 500)      604000      embedding_1[0][0]
__________________________________________________________________________________________________
global_average_pooling1d_1 (Glo (None, 500)          0           bidirectional_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 500)          0           global_average_pooling1d_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 501)          0           dropout_1[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 501)          2004        concatenate_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)            502         batch_normalization_1[0][0]
==================================================================================================
Total params: 5,606,506
Trainable params: 5,605,504
Non-trainable params: 1,002
```

Benchmark

The benchmark for this problem was produced by the winner of the problem that was Random forest model and Gradient Boosting Machine.

The winner paper also mention that neural network also worked and gave high cross validation score but did generated winning metric for the competition. I have tried to use LSTM deep neural network here

(V reference)

The competition was of $100,00 prize which is a huge amount so considering that I will take the benchmark of the model to be kappa score of 0.60903 which was generated by 45th rank in the competition.

My model fairly generated 0.64904 kappa-score which beats the threshold.

Methodology

Data Preprocessing

Data preprocessing was one of the important steps performed in this project as it showed significant changes in the final kappa-score .

The text data has been preprocessed as follows:

- Using keras preprocessor to remove punctuations and making the text to lowercase
- Unnecessary words are remove which are not present in “stopwords” NLTK corpus.
- The words are tokenized and again basic punctuation check is made.
- The tokenizer is also saved using jobli python library for easy and fast use
- Then the text is converted into a sequence using keras preprocessing text to sequence for tokenizer
- Then the texts are post padded to match the length of the text with maximum length

Implementation

There are 7 files(main.py, models.py, config.json, preprocess.py, test.py, train.py and evaluate.py) and data folder which are required to run this project the project can be ran either in the provided jupyter notebook or using command line interface using command line interface is recommended

Running project :

Just create a new environment and install the dependencies provided in requirement.txt and directly run all the cells of jupyter notebook

To run the project in cli first of all run the initial cells of jupyter notebook which creates train and test csv then on cli type “python main.py” to train the model and to test the model you need to type “python main.py --test” then an output file will be generated in the folder then just run the evaluate.py file to get the kappa-scores by typing “python evaluate.py”

Project workflow:

First of all create train.csv and test.csv using the initial commands provided in jupyter notebook these commands will format the train and test files

Then when we go for training first we preprocess the input data (train data) by tokenizing it and removing punctuations also a padding is added such that full data have consistent length of text

After that model is created using the model.py file and the data is trained and the best weights are stored in the checkpoints folder the best weights are only stored when the cross validation loss is decreased to prevent overfitting , after the model gets trained we run the test.py

During test .py execution the saved model is loaded and the test data is passed over the model to predict its classes(score) its saved in the output file then further we want to know the kappa score of the test so we run evaluate.py which gives us the kappa score using the output file.

Initially the base model was a sequential LSTM model which did not gave a good kappa score the reason for that may be is that the data from all the sets were considered similar and scored accordingly but after some critical thinking I cam to a conclusion that the model should be non sequential and the set value should be tagged to the text while training this was one of the complication that I faced during the implementation of the project.

Refinement

As discussed earlier the base model was sequential and did not consider the set of the question the further model gave importance to the set of the questions which increase the kappa score of the model also twitching the model initial embedding parameter brought drastic changes to the kappa score initially the embedding dimensions were high around 300 when it was brought down to 50 increased the kappa value

The initial use of lstm was normal some significant changes in kappa score were observed when bidirectional LSTM was used.

Results

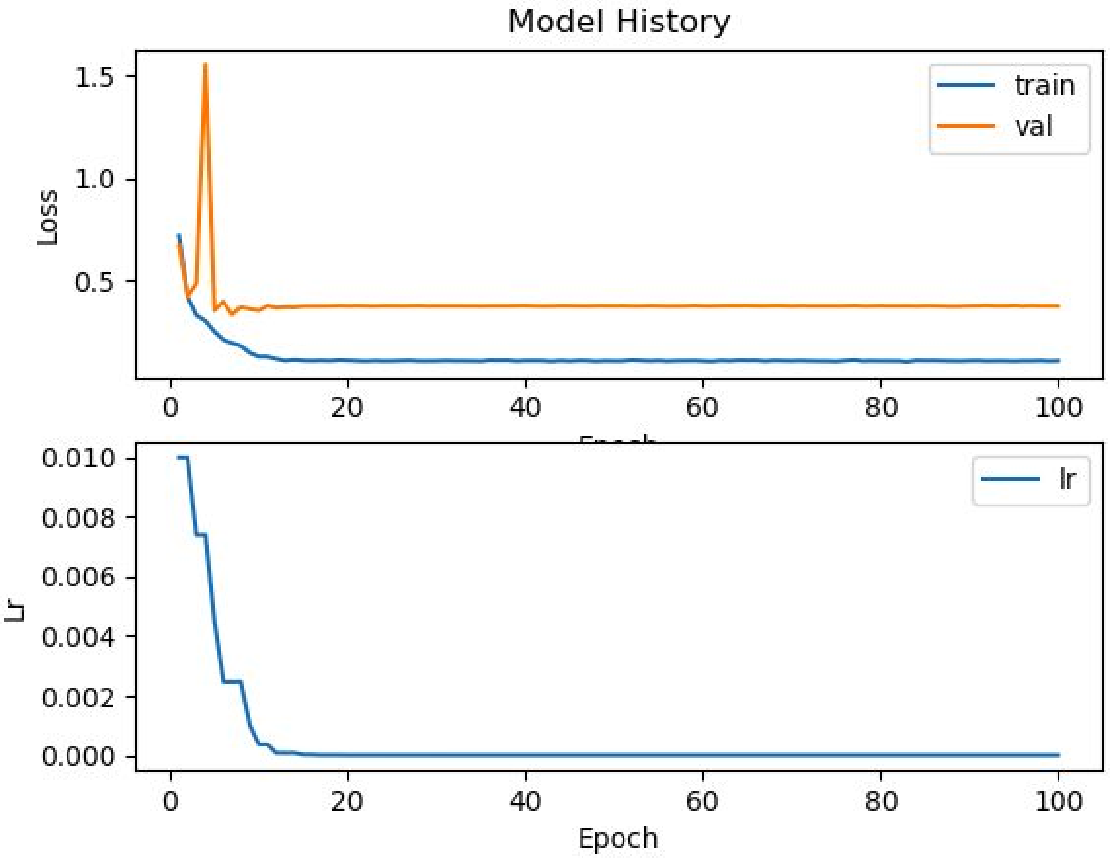
Model Evaluation and Validation

As discussed earlier the parameter such as embedding dimensions affected the final kappa score making it equal to 50 gave the best results in 1-300 range

The training data was splitted for cross validating the training process and also to save the best checkpoints it was 80:20 training to cross validation split which gave 65% accuracy for the cross validation data in the best epoch which was okay according to me and the training accuracy was around 80% which also justifies that the model is not overfitted. This was around 8th epoch after that the cross validation accuracy did not decreased.

The model trained very fast and yielded good kappa score

Below is the history plot of the training process



Justification

The model on the testing produced kappa score of 0.6490 which is more than the proposed value (i.e 0.60903) it justifies that the model did pretty well than expectations but still can be made much better .

Conclusion

The project was pretty much challenging on time to time it judged my conginition , after a lot of trial and errors with some intuitions I was able to beat the benchmark score and was very happy after solving the challenge.

Future Improvements:

I would like to add some parallel training architecture such as BERT(VI reference) to the model and make it an ensemble deep learning model such that small parallel models vote for score and we get the best possible score for the essays.

References

I. https://www.kaggle.com/c/asap-sas

II. https://www.tandfonline.com/doi/abs/10.1080/00220973.1994.9943835

III. https://www.kaggle.com/c/asap-sas#evaluation

IV. https://www.bioinf.jku.at/publications/older/2604.pdf

V. https://storage.googleapis.com/kaggle-competitions/kaggle/2959/media/TechnicalMethodsPaper.pdf

VI. https://www.lyrn.ai/2018/11/07/explained-bert-state-of-the-art-language-model-for-nlp/