Report of Assessing multilingual news article similarity

Name: Raj Bhagat

1. Introduction

The task is to determine whether two news articles cover the same news story. The goal of this task is to create a system that identify multilingual news articles that contain similar information. This is a pairwise similarity task in the applied domain of news articles, rated on a 1 to 4-point scale from most to least similar. Here, we are interested in the real world-happenings covered in the news articles, not their style of writing, political spin, tone, or any other more subjective "design decision" imposed by a medium/outlet.

Every day, thousands of new news articles are published in various languages. Understanding which articles refer to the same story not only improves applications such as news aggregation, but also allows for cross-linguistic analysis of media consumption and attention. However, assessing the similarity of stories in news articles is difficult due to the various dimensions in which a story may differ; for example, two articles may have significant textual overlap but describe similar events that occurred years apart. To address this challenge, a new dataset of nearly 10,000 news article pairs annotated for seven dimensions of similarity spanning 18 language combinations.

This task's potential applications are limitless. The task reduces the mundane task of tracking the similarity of news coverage across different outlets or regions. Monitoring and containment of infectious disease outbreaks, for example, has remained a critical component of public health strategy to contain the diseases, whether previously with Ebola or recently with the COVID-19 pandemic. The ability to accurately track disease outbreaks is critical in the deployment of effective intervention measures. As a result, reports may not only be in English, but effective multilingual systems are also required. As a result, recent research has concentrated on identifying similarities between documents, phrases, stories, and so on.

The challenges faced in this task can be understood by the questions that necessitates knowledge of specific aspects of the events covered, such as what occurred, where and when it occurred, who was involved, and why and how it occurred. Solutions should also not overly rely on general language models, since firstly, stylistic similarity, generic phrases or other non-essential content might be misleading and secondly, dimensions like time and geolocation of the actual event (not the publication time or place) as wells as narrative are not straightforward to capture.

2. Problem Formulation

The task can be implemented by two types 1) Regression and 2) Multi-class classification. I’ve moved forward with Regression model.

The training data set contains 4,964 article pairs from multiple languages (English, German, Spanish, Arabic, Polish, Turkish, etc.) and gold standard similarity scores for six dimensions (Geography, Entities, Time, Narrative, Style, Tone), as well as the Overall score. Furthermore, the metadata contains the article titles, several specific topics and keywords, as well as links to representative images. The test data set contains 4,902 article pairs from multiple languages as well as surprise languages which are not included in the training data.

We fine-tune the SBERT model on multilingual article pairs in Regression based. We only use the Overall score as the target similarity score for fine-tuning. Because the similarity scores provided in the training data range from most to least similar.

3. Method

1. **Data Scrapping:**
   The Semeval_8_2022_ia_downloader is being used to scrap all the articles from the internet. The downloader is a script that scrapes news articles in the 2022 Semeval Task 8 format from the Internet Archive. A pair of articles with id 0123456789_9876543210 will be stored in output_dir/89/0123456789. {html|json} and output_dir/10/9876543210.{html|json} respectively. The downloader scraped 4710 article pairs as other article pair were not available to download for the training data and 3022 article pairs for the testing data.

2. **Data Preprocessing:**
   The firstly the data that is download using the downloader is in JSON or html format in the output directory, so using the Glob function for recursively retrieving path names as the downloader made a output directory which contained all the files in subfolders, data is extracted from the JSON files and important data like Title, Text, pair_id is stored in another dictionary.
   Then data from the training csv file and the dictionary is concatenated and final csv file is generated with pair_id, title, text and overall score. The title and text for same article is combined with a [SEP] token. Then the textual data in English language is then preprocessed and cleaned using the custom preprocess function.
   The idea is to generate embedding using a multilingual pre-trained model rather than converting other languages to English in order to avoid error compounding. Initially, I intended to obtain different embedding for each language. This approach was abandoned because it was difficult to find small models in other languages that did not deplete the resources.
   One of the most significant issues in current NLP research is that most resources are for English because it is a very easy language in comparison to other languages and is widely spoken. It is difficult to find comparable resources in other languages such as French, Danish, and Arabian. I chose to use a multilingual pre-trained similarity embedding generating model instead. This is a single model that has been pre-trained to recognize

multiple languages. The model I used is known as **'para-multilingual-mpnet-base-v2'**. The average performance of this model is **53.75** which the best among the Multi-Lingual Models.

Sentence transformers is a Python framework for creating cutting-edge vector representations of sentences. We can find the most similar sentences based on their semantic meaning by placing the sentences in space and computing the distance between them.

Now using **'para-multilingual-mpnet-base-v2'** sentence transformer model which maps sentences & paragraphs to a 768-dimensional dense vector space and can be used for tasks like clustering or semantic search, I transformed all the articles into feature vectors and stored all the values.

Then the transformed features vectors are for each article is combined with its respect pair making it one feature vector instead of two which makes the computing easier.

3. **Model Design:**

The model is based on TensorFlow 2. It's type of Sequential Model which is linear layer of stacks. The model is created using Dense, Dropout, Batch Normalization and ReLU layers. The Dropout Layer is used to avoid the overfitting as we can see some of the experimental models suffer from overfitting. The Batch Normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1 and it has two parameters momentum and epsilon which are defined in the training phase. The ReLU layer helps with bounding the predictions. The Dense layer implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument. As the dataset is large in total seven dense layer, two dropout layers, six batch normalization and relu layers. All the dense layers have relu activation, normal kernel initializer and, L2 as kernel regularizer while the final dense layer has linear activation with other parameters same. The model used the Adam optimizer as it's much faster computationally than other optimizer's with the parameters learning rate and epsilon which is a small constant for numerical stability .The whole model is showed in the figure below with trainable and non-trainable parameters.

Model: "sequential"

Layer (type)	Output Shape	Param #
dropout (Dropout)	(None, 769)	0
dense (Dense)	(None, 600)	462000

batch_normalization (BatchN ormalization) (None, 600) 2400

re_lu (ReLU) (None, 600) 0

dense_1 (Dense) (None, 400) 240400

batch_normalization_1 (Batc hNormalization) (None, 400) 1600

re_lu_1 (ReLU) (None, 400) 0

dense_2 (Dense) (None, 400) 160400

batch_normalization_2 (Batc hNormalization) (None, 400) 1600

re_lu_2 (ReLU) (None, 400) 0

dropout_1 (Dropout) (None, 400) 0

dense_3 (Dense) (None, 200) 80200

batch_normalization_3 (Batc hNormalization) (None, 200) 800

re_lu_3 (ReLU) (None, 200) 0

dense_4 (Dense) (None, 100) 20100

batch_normalization_4 (Batc hNormalization) (None, 100) 400

re_lu_4 (ReLU) (None, 100) 0

dense_5 (Dense) (None, 10) 1010

batch_normalization_5 (Batc hNormalization) (None, 10) 40

re_lu_5 (ReLU) (None, 10) 0

dense_6 (Dense) (None, 1) 11

==========

Total params: 970,961
Trainable params: 967,541
Non-trainable params: 3,420

_____

4. **Loss Function Design:**

   There are some of the loss function choices in the Regression model

   1) Mean Absolute Error Loss
   2) Mean Squared Error Loss
   3) Mean Squared Logarithmic Error Loss
   4) Huber Loss

   So, I've selected Huber loss is a loss function that is used to solve regression problems. This function is a combination of the mean squared error (MSE) and mean absolute error (MAE). Huber loss function is quadratic (MSE) when difference between actual and predicted values is small, otherwise function is linear (MAE). Mean Absolute Error is the Evaluation Metric. As a regression predictive modeling problem involves predicting a real-valued quantity and we are going predict between 1 to 4.

5. **Training:**

   The training was done on multiple experimental model design and values as you can in the experimental section of the report. The training data is already preprocessed and ready for training phase. The final training was done on the model as shown in the model design which generated the minimal loss on the training and testing data. The training process was indeed a tough job as I'd have to train model on multiple experimental parameters . The training data consisted of 4710 pairs of articles with all the scores like Geography, Overall, Narrative, Time, etc. The training data was encoded using the sentence transformer **'para-multilingual-mpnet-base-v2'** and the embeddings of size (4710, 768) were stored into a csv file for both the articles. Then the articles are already paired with each other, so the embeddings are combined for the article pairs. And the labels are already available as the Overall score in training dataset. The features and labels are separated using sklearn.model_selection.train_test_split and parameters for the train_test _split are test_ratio is 0.3 as keeping a small validation set may not produce desired results to tune the hyperparameters in the model. The train and validation set are converted to tensor of dtype float32. The parameter used in the model are input_dimension = 769 as the feature vector has the shape of 769, hidden_dimension = 600, hidden_dimension_1 = 400, hidden_dimension_2 = 400,

hidden_dimension_3 = 200, hidden_dimension_4 = 100, hidden_dimension_5 = 10 output_dimension = 1, dropout_rate = 0.01, learning rate = 1e-5, momentum =0.99 and, epsilon = 1e-3. Then a sequential model is created as shown in figure. The Optimizer used for the training and testing is Adam as it's computationally fast than other optimizers available. The loss function is Huber loss as it's a regression model. The model is then compiled with the optimizer and loss function. The compiled model is then fitted with batch size =1 for full batch, epochs =500 as I've implemented early stopping criterion for better results, and validation data. The fitted model can then predict values for the training data as well as the validation data. The Evaluation Metric used in the task is Mean Absolute Error and, the model is evaluated using the TensorFlow 2 evaluation method, but for second check MAE is imported using sklearn.metrics which predicted the final scores on the training data and validation data which is 0.96 on the training data and 0.974 on the validation data.

6. **Inference:**
   The model built in the training phase is saved and used in the Inference phase. The model is saved in the H5 format, and it is loaded in the run_prediction file. The model is already built and fitted so direct prediction is performed using the model. The testing data had 3034 pair of articles which goes through the scrapping and preprocessing phase and the data is encoded by sentence transformer **'para-multilingual-mpnet-base-v2'** which generated embeddings of size (3034, 768) for the testing data. The data is then converted to tensor to match the training data format. Then the data is evaluated and predicted which generated the MAE of 1.0.

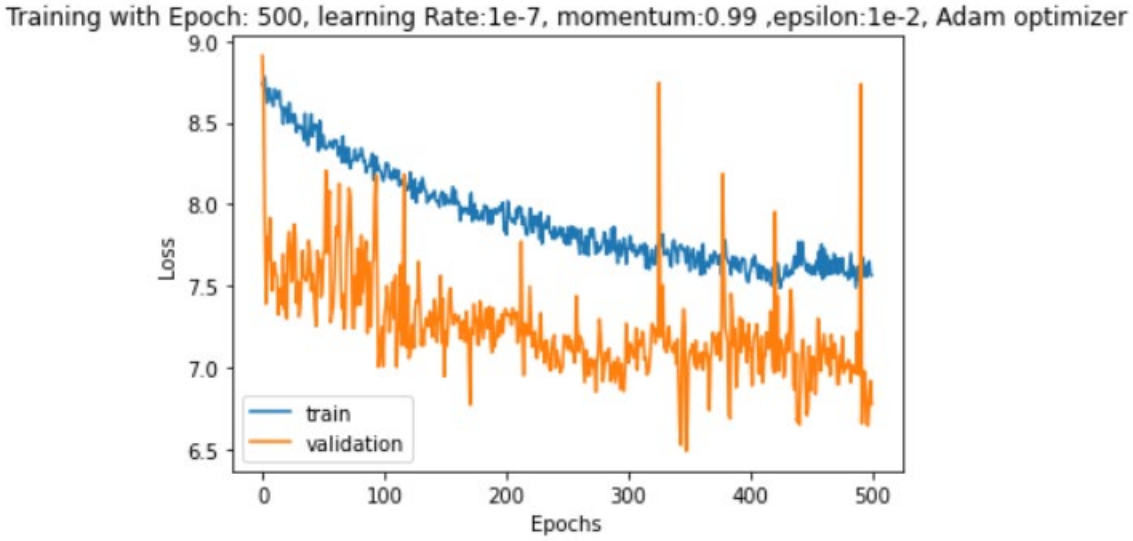
4. Experiments

There are 4 types of Multi-Lingual Sentence Transformer available which are distiluse-base-multilingual-cased-v1, distiluse-base-multilingual-cased-v2, paraphrase-multilingual-MiniLM-L12-v2, paraphrase-multilingual-mpnet-base-v2.

The First choice was distiluse-base-multilingual-cased-v2 model which supports 50+ languages but it had one draw back that the model could only generate embeddings on size (4710, 512) and because of that the loss and evaluation were not satisfactory. So, the second choice which is used in the final model is paraphrase-multilingual-mpnet-base-v2 which also supports 50+ languages and could generate embeddings of size 768 which helps a lot in the model training and tuning the hyper-parameter.

There are multiple models with different epochs, batch size, optimizers, loss, momentum, epsilon etc. All the experiments are listed below with their respective parameters. The models had different number of layers, epochs, optimizer as well as changed hyperparameters as well as learning rate scheduler was also used on some models.

The first figure is of the training with respective parameters and the second is of the evaluation based on that model.



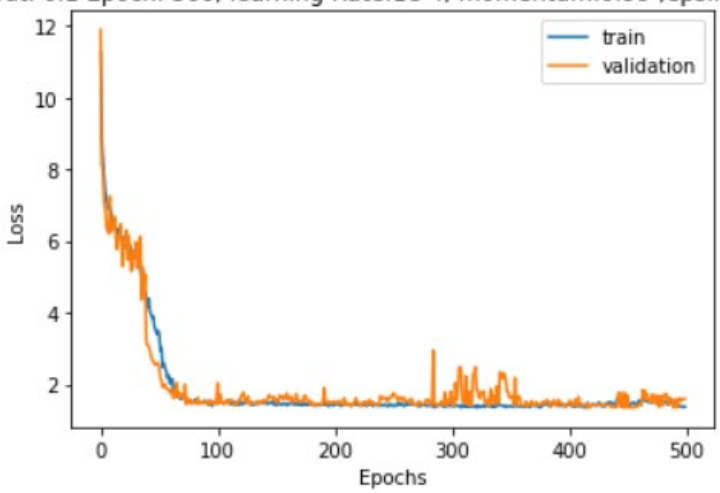
Using the Mean Absolute Error Evaluation Metric from sklearn.

```
[189] from sklearn.metrics import mean_absolute_error

      train_score = mean_absolute_error(train_y, y_pred_train)
      test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[190] print(train_score)
      print(test_score)

      1.5710361
      2.2653975
```

Training with Dropout: 0.1 Epoch: 500, learning Rate:1e-4, momentum:0.99 ,epsilon:1e-2, Adam optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

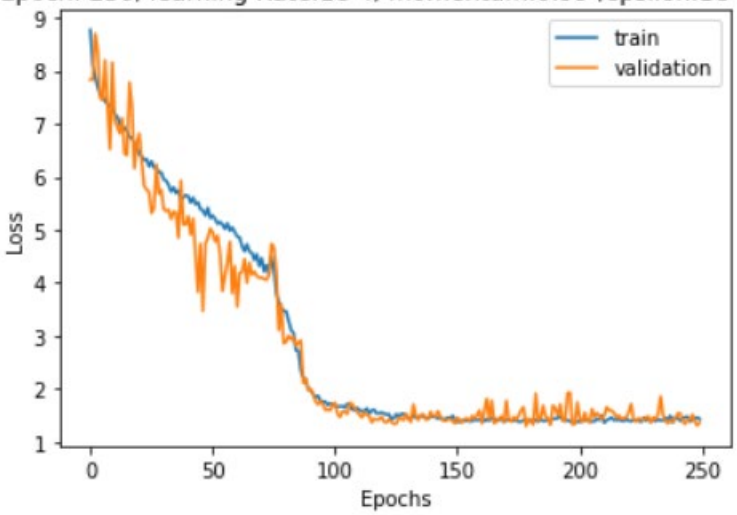
```
[229] from sklearn.metrics import mean_absolute_error

      train_score = mean_absolute_error(train_y, y_pred_train)
      test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[230] print(train_score)
      print(test_score)
```

```
      1.0740688
      1.1019831
```

Training with Epoch: 250, learning Rate:1e-4, momentum:0.99 ,epsilon:1e-2,Adam optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

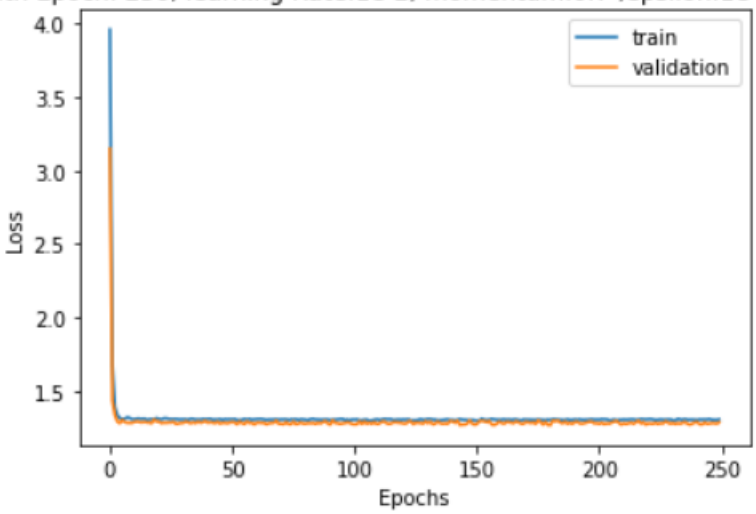
```
from sklearn.metrics import mean_absolute_error

train_score = mean_absolute_error(train_y, y_pred_train)
test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[77] print(train_score)
     print(test_score)
```

```
1.0284564
1.0285405
```

Training with Epoch: 250, learning Rate:1e-2, momentum:0.7 ,epsilon:1e-3,SGD optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

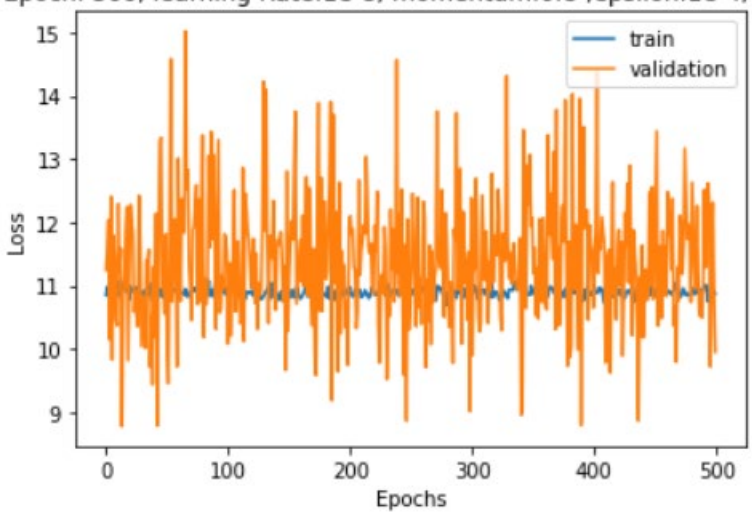
```
from sklearn.metrics import mean_absolute_error

train_score = mean_absolute_error(train_y, y_pred_train)
test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[90] print(train_score)
     print(test_score)
```

```
1.0181367
1.0066012
```

Training with Epoch: 500, learning Rate:1e-8, momentum:0.8 ,epsilon:1e-4, Adagrad optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

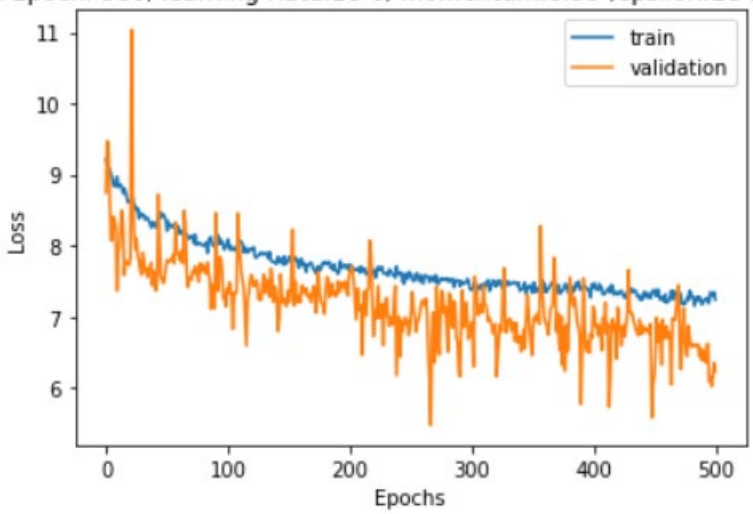
```
[102] from sklearn.metrics import mean_absolute_error

      train_score = mean_absolute_error(train_y, y_pred_train)
      test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[103] print(train_score)
      print(test_score)
```

```
      1.7987541
      2.8676927
```

Training with Epoch: 500, learning Rate:1e-6, momentum:0.99 ,epsilon:1e-2, Adam optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

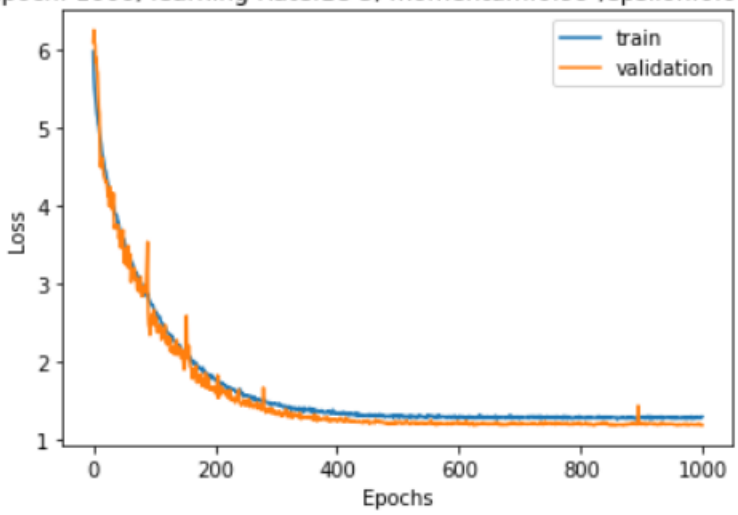
```
from sklearn.metrics import mean_absolute_error

train_score = mean_absolute_error(train_y, y_pred_train)
test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[116] print(train_score)
      print(test_score)
```

```
1.5313035
2.1566565
```

Training with Epoch: 1000, learning Rate:1e-5, momentum:0.99 ,epsilon:0.0001, SGD optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

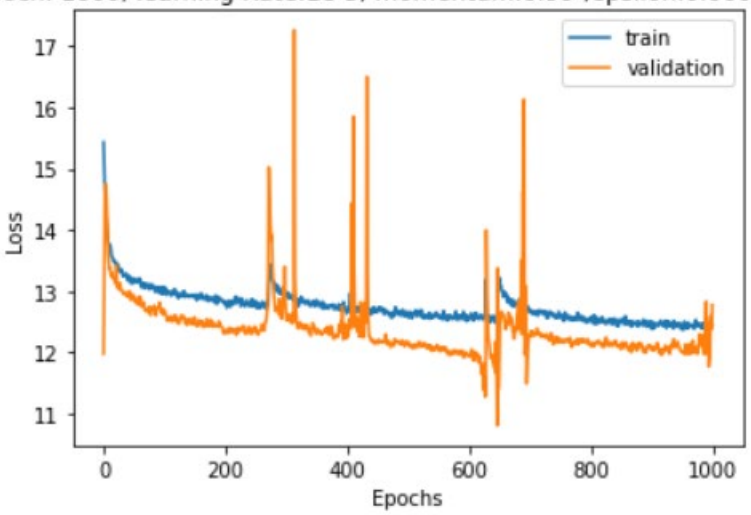
```
[97] from sklearn.metrics import mean_absolute_error

     train_score = mean_absolute_error(train_y, y_pred_train)
     test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
print(train_score)
print(test_score)
```

```
0.9940979
0.96682656
```

Training with Epoch: 1000, learning Rate:1e-5, momentum:0.99 ,epsilon:0.0001, Adagrad optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

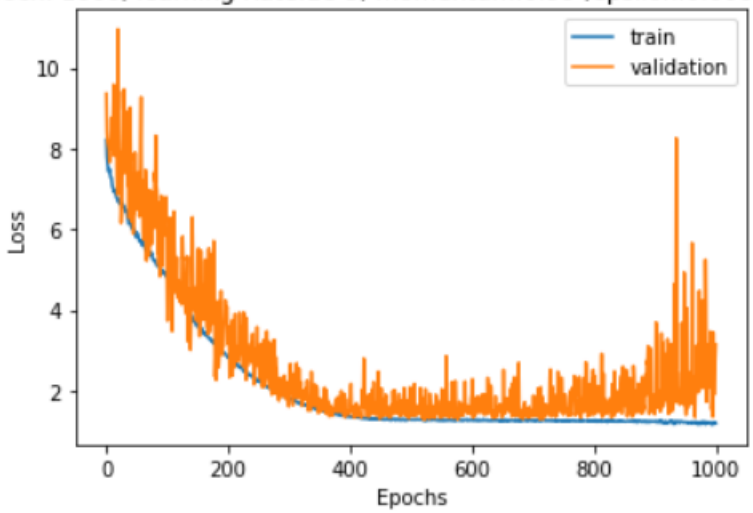
```
[112] from sklearn.metrics import mean_absolute_error

      train_score = mean_absolute_error(train_y, y_pred_train)
      test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[113] print(train_score)
      print(test_score)
```

```
      1.8246421
      3.3657959
```

Training with Epoch: 1000, learning Rate:1e-5, momentum:0.99 ,epsilon:0.0001, RMSprop optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

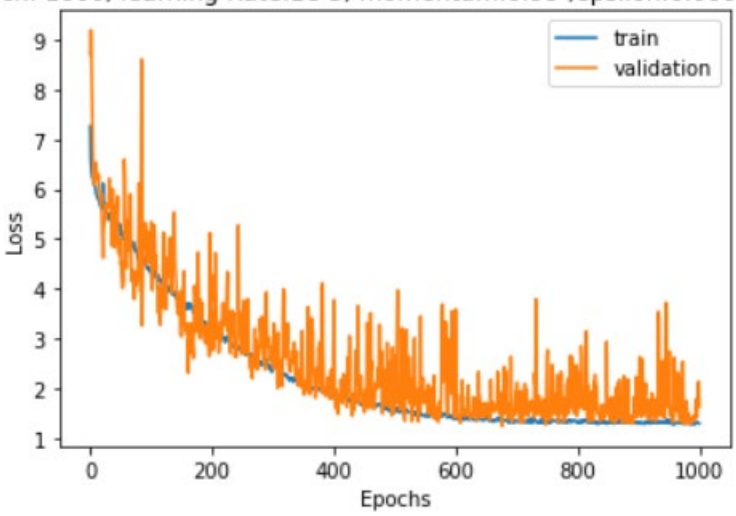
```
from sklearn.metrics import mean_absolute_error

train_score = mean_absolute_error(train_y, y_pred_train)
test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[130] print(train_score)
      print(test_score)
```

```
1.0887009
1.3962023
```

Training with Epoch: 1000, learning Rate:1e-5, momentum:0.99 ,epsilon:0.0001, RMSprop optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

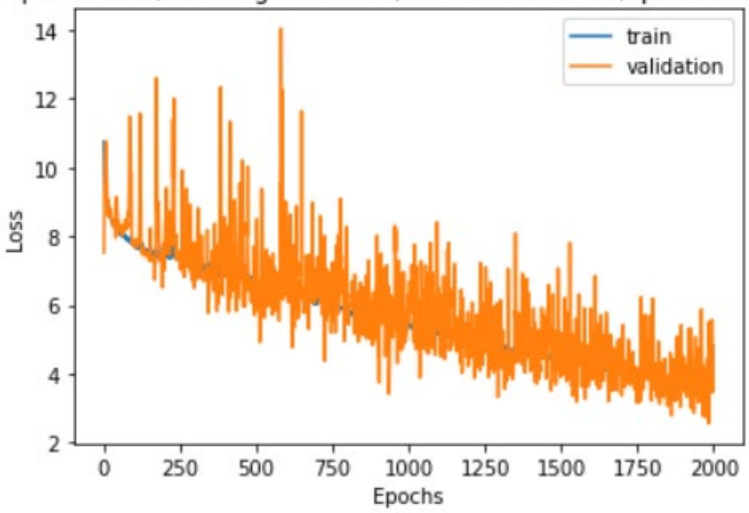
```
[152] from sklearn.metrics import mean_absolute_error

      train_score = mean_absolute_error(train_y, y_pred_train)
      test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[153] print(train_score)
      print(test_score)
```

```
      1.0832264
      1.0732574
```

Training with Epoch: 2000, learning Rate:1e-6, momentum:0.99 ,epsilon:1e-5,Adam optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

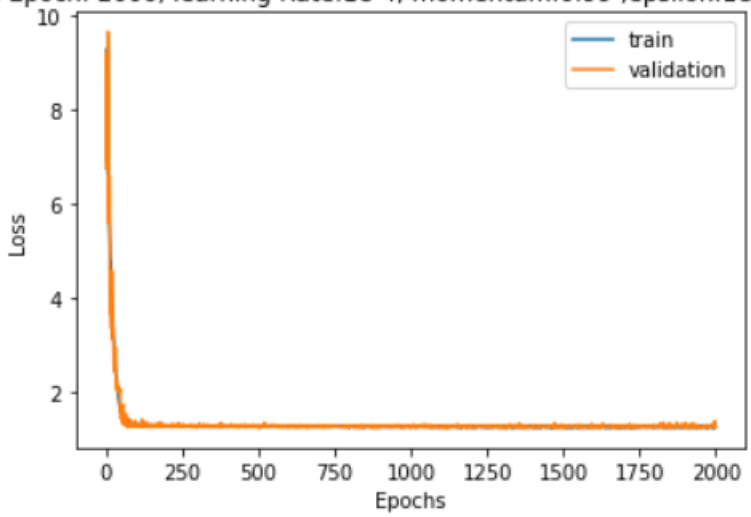
```
from sklearn.metrics import mean_absolute_error

train_score = mean_absolute_error(train_y, y_pred_train)
test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[166] print(train_score)
      print(test_score)
```

```
1.6187263
1.663944
```

Training with Epoch: 2000, learning Rate:1e-4, momentum:0.99 ,epsilon:1e-2,Adam optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

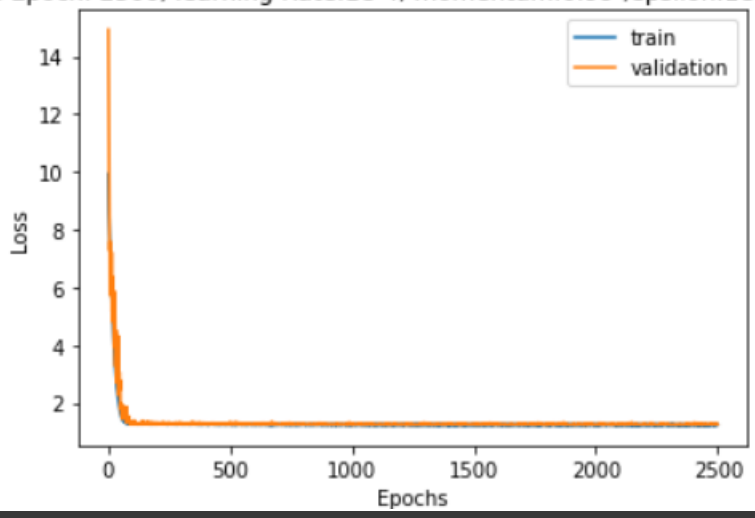
```
[183] from sklearn.metrics import mean_absolute_error

      train_score = mean_absolute_error(train_y, y_pred_train)
      test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
[184] print(train_score)
      print(test_score)
```

```
      0.97152764
      0.9784189
```

Training with Epoch: 2500, learning Rate:1e-4, momentum:0.99 ,epsilon:1e-2,Adam optimizer



Using the Mean Absolute Error Evaluation Metric from sklearn.

```
[183] from sklearn.metrics import mean_absolute_error

      train_score = mean_absolute_error(train_y, y_pred_train)
      test_score = mean_absolute_error(test_y, y_pred_test)
```

The MAE scores the training and validation set.

```
print(train_score)
print(test_score)
```

```
0.97152764
0.9784189
```

5. Conclusion

Multilingual news article similarity is a challenging problem despite sharing some characteristics with Semantic Text Similarity. By deploying, the model with multiple layers and optimizers and loss functions, I've tried to build a model that has minimalistic loss and high evaluation value. With somewhat complex model design like showed in the report, good results can be achieved. The training portion of the task is very tricky and the options to train and build the model are in an abundance. Nevertheless, the current state of the system leaves many things to be improved: Currently only Title and Text of the article are used for the training and evaluation, but to improve the model we can include images, meta data and much more to create more robust and efficient model. The model was tried with different sentence transformers and multiple model designs, but nonetheless, there was no clear consensus as to the best

architectures, embedding models, or preprocessing to perform on the data as it's an ongoing task in NLP.