Assessing multilingual news article similarity

Student Name: Paruchuri Ratan Chowdary **Student ID:** 20011581

Introduction:

In the real world, there will be news articles written in multiple languages. At any instance, news agencies tend to author articles on similar topics to reach their targeted audiences. The set of audience for different news agencies can collide because of geography or audience knowing multiple languages. In this project, we calculate the similarity between the content of the news articles which are written in different languages. We score the similarity of the articles between 1 - 4.

The multilingual news article similarity model has various applications. We will review two of them. Firstly, Machine translation. The model can be used to translate news articles from one language to another while preserving the content of the original article. Second, sentiment analysis. The model can be used to analyze the sentiment of news articles in different languages and provide insights into public opinion on several topics.

It is difficult to model multilingual news article similarity because different languages have different syntactic and semantic structures. Language contact [1] can be a problem as languages lack words and phrases. Then the model must borrow words from other languages. There are a few other difficulties which will not go through.

Problem Formulation:

Given the train and test datasets, containing the links to news articles of various languages (English, Spanish, Arabic...). The dataset itself does not have the article text. Download the text using the **semeval_8_2022_ia_downloader** **.** After downloading, assign a percentage of train data for validation. Develop a model that predicts the similarity score in range 1 to 4. The model can use techniques such as optimizers, regularizer etc. Implement Evalution metrics and calculate MAE of the overall score on the test set.

In this project, we implement the task as regression model. We take a pair of articles along with their titles as inputs. Encoding is performed with sentence transformer. The similarity score is the output of the model. The target column of the dataset is “overall” column.

Method:

Download dataset:

To download the text of the news articles links given in the dataset, we use semeval_8_2022_ia_downloader. We follow the below steps to download the given dataset into google collab /context/ folder,

Install semeval_8_2022_ia_downloader:

```
!pip install semeval_8_2022_ia_downloader
```

Scrape train dataset:

```
!python -m semeval_8_2022_ia_downloader.cli --links_file='/content/semeval-2022_task8_train-data_batch.csv' --dump_dir='/content/my_folder'
```

Scrape test dataset:

```
!python -m semeval_8_2022_ia_downloader.cli --links_file='/content/final_evaluation_data.csv' --dump_dir='/content/downloads_test_data’
```

Data Preprocessing:

We start by walking through the dataset and collecting the Json files. Then we extract title and text from the Json files. We concatenate and separate them with the keyword [SEP]. This helps the model understand the relation between title and text and can generate better embeddings. To keep code simple, we choose separator over passing tile and text as separate inputs. Title and text, along with data in rest of the columns of the dataset are added to the new file. For train and test, we create new files with name train.csv and test.csv respectively.

We then clean the data by removing punctuation, URL and numbers in the concatenated text. After that, we assigned 20 percent of the train data for validation.

Embeddings:

Note: Our first choice was XLMRobertaTokenizer and TFRobertaModel for tokenization and word embeddings. Due to limited computational resources, we shifted to other methods. You can find the details on this in the experimentation and results section.

We finally went on with Sentence transformers. This creates embeddings for a sentence, not individual words. It generates vector of size 768. When we pass a sentence to the sentence transformer, it tokenizes the sentence and passes it to a pre-trained BERT. To get the sentence embeddings, pooling operation is applied on output of BERT [3]. We use ‘paraphrase-multilingual-mpnet-base-v2' version, which is a multi-lingual variant of sentence transformers. It was trained on corpus with 50+ languages.

Model:

The model is partially inspired from the article [4]. We used Siamese neural architecture to find the similarities between multilingual news articles. Fun fact, Siamese means twin. This architecture has two branches. Two articles' embeddings are passed through each of the branches. Each branch consists of a dense layer. The hidden unit's size is 256. These layers are enforced with ReLU activation, l2 kernel_regularizer.

The output of these two branches is passed to dot layer, where the dot product to two vectors is calculated. The parameter normalized=True, ensures the normalization of the product. The output of this layer is then passed to a dense layer with linear activation. This outputs a single scalar value.

To train the model, we use Adam optimizer. The parameters, that we set for training purposes are the following,

Number of epochs = 100

Learning rate = 0.001

Size of the batch = 32

In the experimentation and results section, we will observe the results of different optimizers and epoch settings. We will also observe the performance by adding additional layers, along with batch normalization and early stopping

Please find the model summary below,

```
Model: "model_15"

__________________________________________________________________________________________
__________________________________

Layer (type)              Output Shape        Param #    Connected to

==========================================================================================
==================================

input_25 (InputLayer)        [(None, 768)]       0         []


input_26 (InputLayer)        [(None, 768)]       0         []


dense_101 (Dense)          (None, 256)         196864     ['input_25[0][0]']


dense_102 (Dense)          (None, 256)         196864     ['input_26[0][0]']


dot_15 (Dot)                 (None, 1)           0         ['dense_101[0][0]',
                                                          'dense_102[0][0]']


dense_103 (Dense)          (None, 1)          2         ['dot_15[0][0]']


==========================================================================================
==================================

Total params: 393,730

Trainable params: 393,730

Non-trainable params: 0

__________________________________________________________________________________________
__________________________________
```

Loss Function:

We designed the assessing multilingual news article similarity as regression model. So, we can use Mean Squared Error, Mean Absolute Error, Huber loss. The architecture of the model is Siamese. So, we could use contrastive loss.

In this project, we have used Mean Squared Error as the primary loss function. In experimentation and results, we could find results for contrastive loss as well.

As evaluation metric, we used Mean absolute error.

Training:

The model has Siamese neural architecture. To build the model, we used TensorFlow framework. We have a data set that contains text from multiple languages. For training we use 4965 pairs of news articles. The data is preprocessed before moving to the next steps. We train the model by converting the strings of article title and text separated with [SEP] into sentence embeddings, using sentence transformer. This will create 768 sized vectors. We assigned 20 percent of train data for validation.

We train the network in such a way that it predicts the similarity score with minimal loss. Our target column in the data set is 'Overall'. The number of epochs used for this model is 100. For every step of epoch, we evaluate on validation data. This will help the model to generalize for unseen data. We use Adam optimizer to adjust weights during back propagation. The loss function we use is mean squared error, as it's a regression model.

The output of the mode will be a scalar value between 1-4. If we get a floating-point number, we will round it off to the nearest number. The evaluation metric used in this project meant absolute error. We will observe the loss and MAE value for various configurations in the experimentation and results section.

Inference:

For inference, we use 4903 pairs of news articles. This is preprocessed before encoded into sentences embedding. This is done by a paraphrase-multilingual-mpnet-base-v2, which is a version of sentence transformers. We then use the model that was saved in h5 format during the training phase, to predict on test examples. The result is evaluated by Mean Absolute Error. The MAE we obtained was 1.11

Experimentation and Results:

Encoded the sentences with XLM-RoBERTa models:

Roberta Model:

We use a variant of 'Robustly optimized BERT approach'. We will discuss why we chose RoBERTa over BERT. RoBERTa is a transformer-based model, which is built on the BERT architecture. It is trained with dynamic masking, Full sentences without NSP (Next Sentence Prediction), large mini batches and a larger byte level BPE (Byte Pair Encoding) [2]. Also, it was trained with larger datasets than BERT was ever trained. This gives an advantage to RoBERTa over BERT. We chose XML-Roberta which is a multilingual variant of RoBERTa.

Tokenizing dataset:

We chose XLMRobertaTokenizer, which can be found in Hugging Face model hub. This is a multilingual tokenizer. We chose 'xlm-roberta-base' version as tokenizer, which was trained in 100+ languages. The other reason we chose this tokenizer is because it is designed specifically for XLM-RoBERTa model which we will use for embedding in the next step. With this pretrained tokenizer, we tokenize the data.

Embeddings:

To embed the tokenized data, we chose TFRobertaModel. This is XML-RoBERTa model written in TensorFlow. The embedding sizes that are produced by TFRobertaModel is 768.

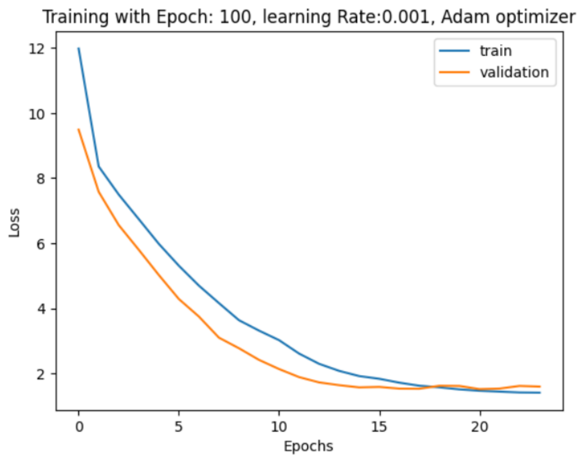
The reason we chose pre-trained models is because they give state-of-the-art performance as they were trained on large corpus. Also, they tend to capture better contextual information and meaning, while creating embedding vectors.

While this is a fantastic model, it requires a lot of computational resources. I have used 25 GB of ram, but still the Collab instance crashed.

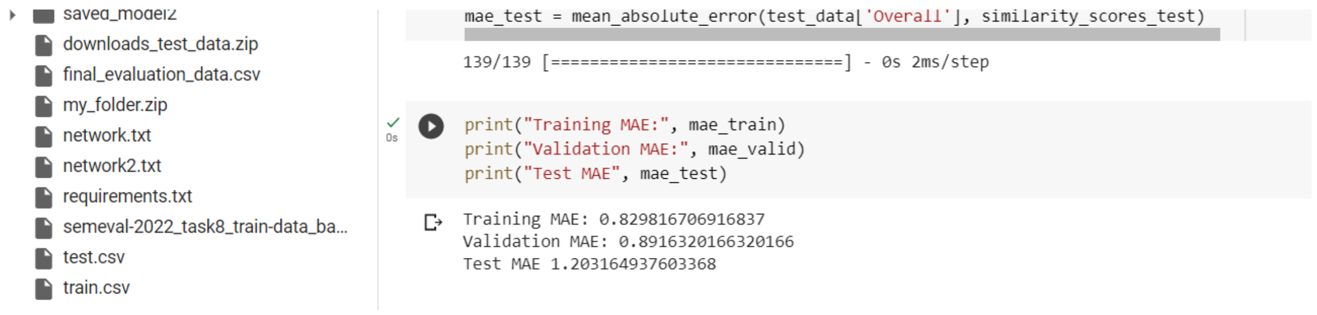
Your session crashed after using all available RAM. As a Colab Pro user, you have higher memory than non-subscribed users, but if you are interested in even higher RAM runtimes, you may want to check out Colab Pro+.

Additional Layers:

We evaluate the model's performance by adding additional layers. To prevent overfitting, we use early stopping and dropout of 0.4. We enforce the layers with batch normalization. We set its parameters momentum 0.99 and epsilon 0.001. The model is saved as saved_model2.h5. The model summary is saved in network1.txt.



```
121/121 [==============================] - 0s 2ms/step
31/31 [==============================] - 0s 2ms/step
```



saved_model2
downloads_test_data.zip
final_evaluation_data.csv
my_folder.zip
network.txt
network2.txt
requirements.txt
semeval-2022_task8_train-data_ba...
test.csv
train.csv

```
mae_test = mean_absolute_error(test_data['Overall'], similarity_scores_test)

139/139 [==============================] - 0s 2ms/step
```

```
print("Training MAE:", mae_train)
print("Validation MAE:", mae_valid)
print("Test MAE", mae_test)
```

```
Training MAE: 0.829816706916837
Validation MAE: 0.8916320166320166
Test MAE 1.203164937603368
```

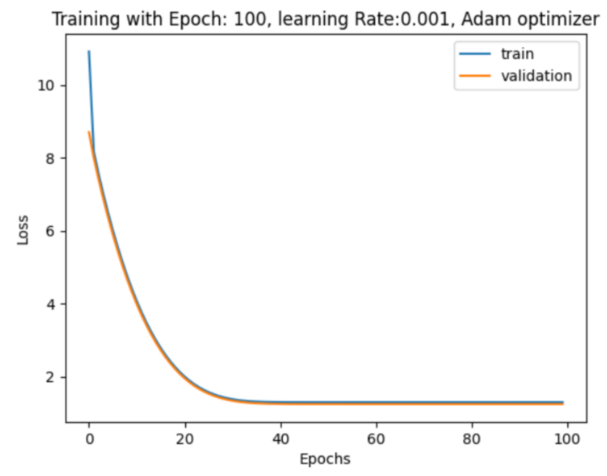
Different epoch sizes and optimizers:

The two optimizers we chose were Adam and SGD. These two optimizers have different approaches when updating parameters.

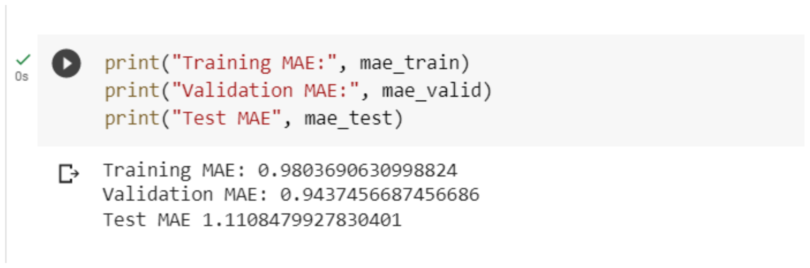
Adam is an adaptive learning optimizing algorithm. It maintains a learning rate for each parameter and adapts the learning rate based on the running average of the first and second moments of the gradients.

While SGD updates the weights of a neural network based on the gradient of the loss function with respect to the weights. SGD updates the weights after each batch of training data

Number of epochs – 100, optimizer Adam



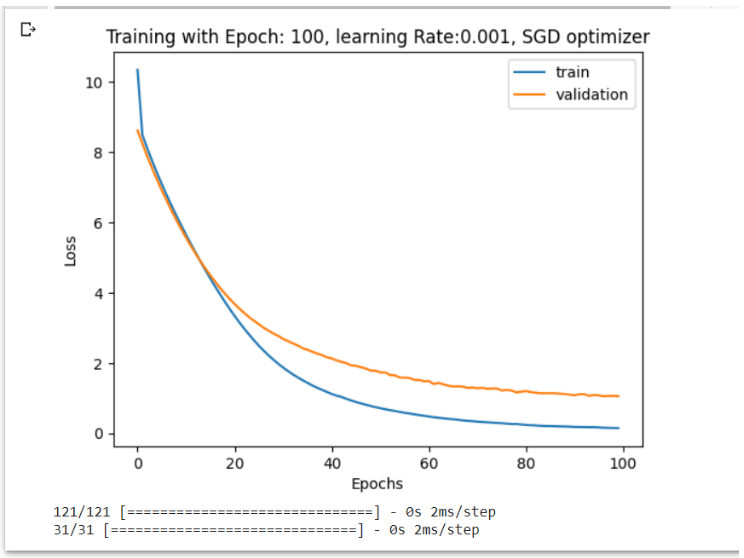
```
121/121 [==============================] - 0s 2ms/step
31/31 [==============================] - 0s 2ms/step
```



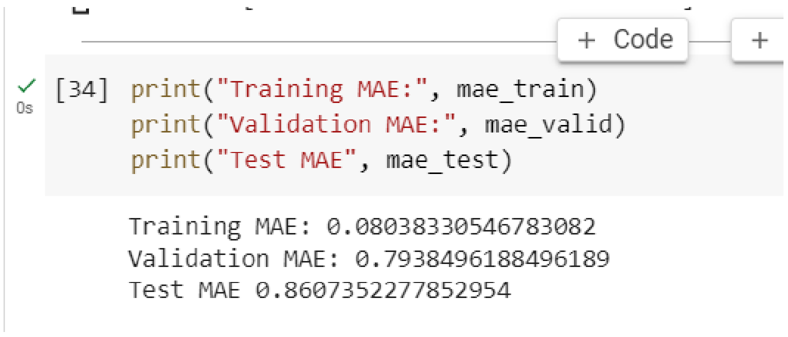
```
print("Training MAE:", mae_train)
print("Validation MAE:", mae_valid)
print("Test MAE", mae_test)
```

```
Training MAE: 0.9803690630998824
Validation MAE: 0.9437456687456686
Test MAE 1.1108479927830401
```

Number of epochs – 100, optimizer SGD



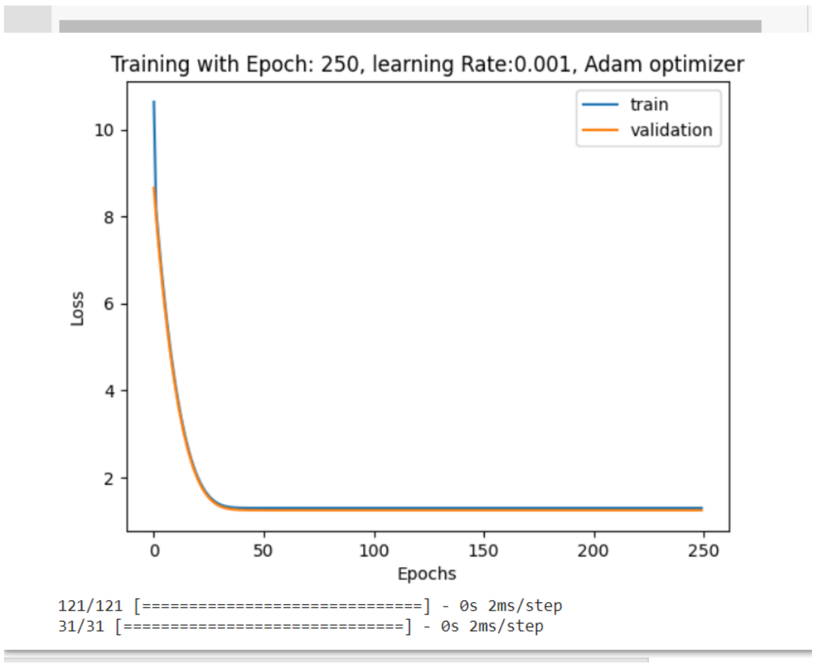
```
121/121 [==============================] - 0s 2ms/step
31/31 [==============================] - 0s 2ms/step
```



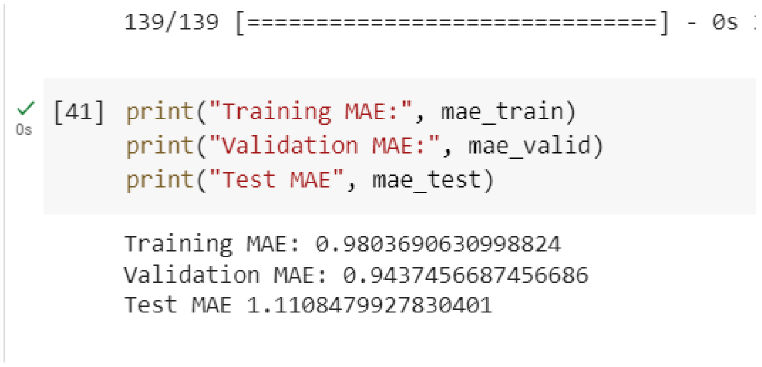
```
[34] print("Training MAE:", mae_train)
     print("Validation MAE:", mae_valid)
     print("Test MAE", mae_test)
```

```
Training MAE: 0.08038330546783082
Validation MAE: 0.7938496188496189
Test MAE 0.8607352277852954
```

Number of epochs – 250, optimizer Adam



```
121/121 [==============================] - 0s 2ms/step
31/31 [==============================] - 0s 2ms/step
```

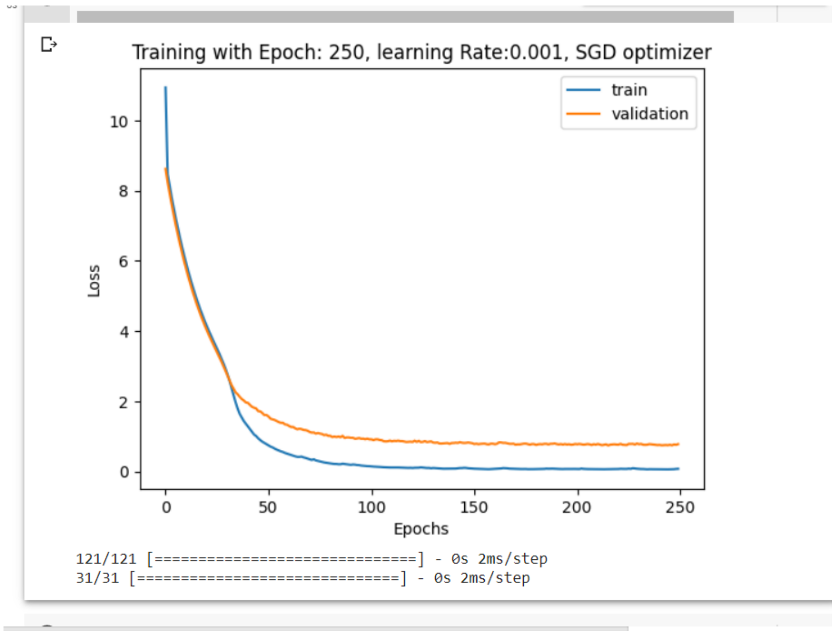


```
139/139 [==============================] - 0s
```

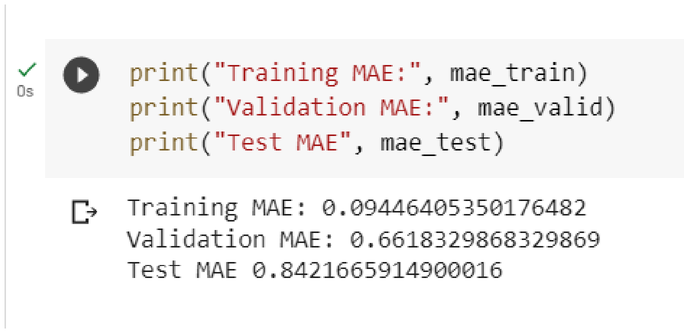
```
[41] print("Training MAE:", mae_train)
     print("Validation MAE:", mae_valid)
     print("Test MAE", mae_test)
```

```
Training MAE: 0.9803690630998824
Validation MAE: 0.9437456687456686
Test MAE 1.1108479927830401
```

Number of epochs – 250, optimizer SGD



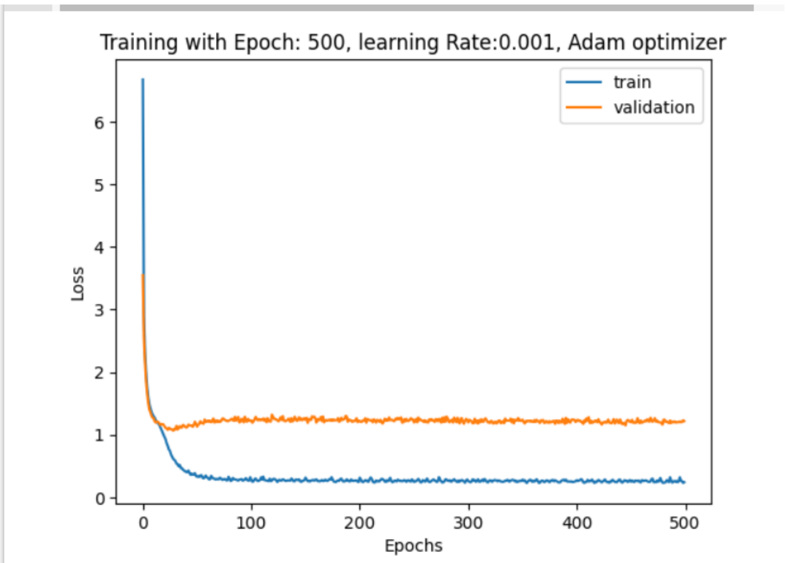
```
121/121 [==============================] - 0s 2ms/step
31/31 [==============================] - 0s 2ms/step
```



```
print("Training MAE:", mae_train)
print("Validation MAE:", mae_valid)
print("Test MAE", mae_test)
```

```
Training MAE: 0.09446405350176482
Validation MAE: 0.6618329868329869
Test MAE 0.8421665914900016
```

Number of epochs – 500, optimizer Adam

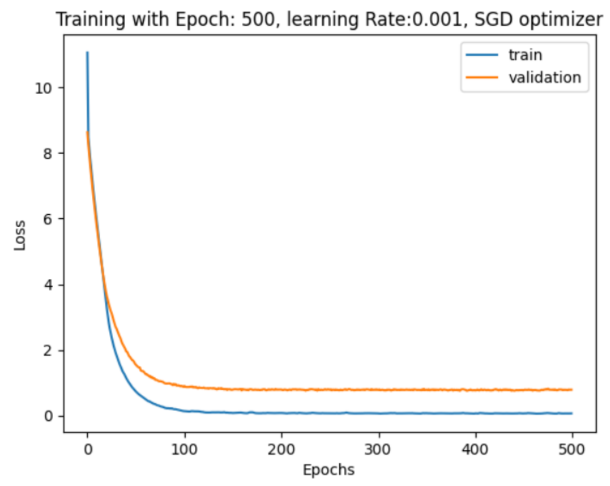


```
121/121 [==============================] - 0s 2ms/step
31/31 [==============================] - 0s 2ms/step
```

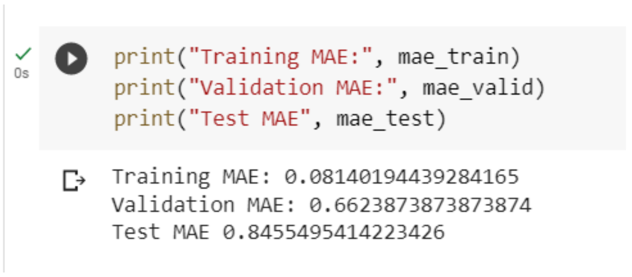
```
[63] print("Training MAE:", mae_train)
     print("Validation MAE:", mae_valid)
     print("Test MAE", mae_test)
```

```
Training MAE: 0.16049105207752803
Validation MAE: 0.813981288981289
Test MAE 0.8964441437377837
```

Number of epochs – 500, optimizer SGD



```
121/121 [==============================] - 0s 2ms/step
31/31 [==============================] - 0s 2ms/step
```



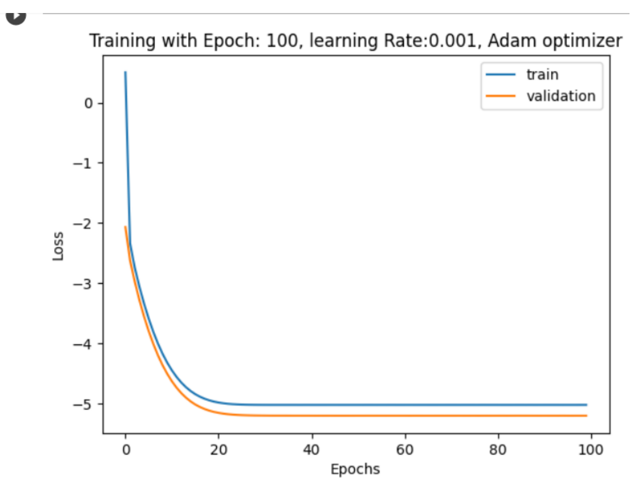
```
print("Training MAE:", mae_train)
print("Validation MAE:", mae_valid)
print("Test MAE", mae_test)
```

```
Training MAE: 0.08140194439284165
Validation MAE: 0.6623873873873874
Test MAE 0.8455495414223426
```

Different loss functions:

Contrastive loss:

Constrastive loss is used for learning similarity between pairs.



```
121/121 [==============================] - 0s 1ms/step
31/31 [==============================] - 0s 2ms/step
```

```
[90] print("Training MAE:", mae_train)
     print("Validation MAE:", mae_valid)
     print("Test MAE", mae_test)
```

```
Training MAE: 1.7962604495634404
Validation MAE: 1.8352564102564102
Test MAE 1.4198240866035183
```

Conclusion:

Assessing multilingual news similarity can be modelled with simple Siamese neural architecture. This is because sentence transformers are trained on huge amount of corpus. They can encode context and meaning of text. Observing the results of experimentation, the model performs better with 100 epochs, Adam optimizer and Mean Squared Error (MSE) loss. There is room to experiment with different architectures and improve performance. The performance can also be improved with hyper parameter tuning methods, different loss functions and optimizers. NLP is an experimental field, there is no end for options. The model we developed can help the users to cut down the duplication of news in their daily life.

References:

1. Language Contact and Similarity Across Languages
2. RoBERTa: A Robustly Optimized BERT Pretraining Approach
3. Intro to Sentence Embeddings with Transformers
4. Medium: Assesing MultiLingual News Article Similarity