Deep Learning | Assignment 2

***Working environment:***

For our environment, we’ve chosen to work on google colab so we started by mounting the drive, loaded the data as a CSV file into it, and then read it into our code using the pandas library.

# Data Preprocessing

## Mapping the labels:

After visualizing the data we realized that the labels of the data were either the word “positive” or “negative” so we changed that to 1 for “positive” and 0 for “negative” using a map that we applied to column “sentiments” which is the label column.

## Removing HTML tags:

For removing the HTML tags like (<br></br>) we used the BeautifulSoup library that helped us with this using the “html.parser” parameter.

## Removing punctuation:

For this function, we used the string library then we created a function that sets any character that belongs to the string.punctuation characters to an empty string and the hyphens are set to a space instead of an empty string.

## Lowercase:

To lowercase all the characters it was an easy job applying ***text.lower()*** to all the text that we have.

## Filtering out stop words:

We used the library nltk and made a set of stop words that are provided by the library then we made a function that split the text passed to it and reassemble it removing any word that match a stop word in the set provided by the library.

## Lemmatization of words:

For this section, we relied on ***WordNetLemmatizer()*** from the nltk library and we lemmatized each word in our sentences and then joined the words separated by spaces in between.

**BERT Model :**

# Models Architecture

Creating a ***BertClassifier*** Class to define our model specifying a logic to to make the bert layers trainable or not, the dropout percentage and the hidden layers used.

### Inside The Bert Classifier Class :

* 1. Loading the preprocessor layer that we used from TensorflowHub for encoding text into tokens.
  2. Loading the BERT encoder layer from TensorflowHub.
  3. Taking the encoders pooled Output of shape [batch\_size, 768].
  4. Adding a dropout layer with the given dropout chance.
  5. Adding our hidden layers.
  6. Add the output layer with a single sigmoid unit.

**Roberta Model :**

First we load the Roberta Tokenizer and Model from the transformers library and using the pretrained weights of ***roberta-base.***

### Defining a Tokenizer function :

1. First, It is necessary to convert the output of the tokenizer to a dictionary to use it in ***model.fit().***
2. Then, The tokenizer accepts a string or a list of string so we type caste our text into a list.
3. Apply padding and truncation so that all text has a length of ***`max\_length`***, which is ***512*** in this case.
4. Returning tensorflow tensors instead of pytorch.

Creating a ***RoBertaClassifier*** Class to define our model specifying the dropout percentage and the hidden layers used.

### Inside the Roberta Classifier Class :

1. After tokenizing our text we need to pass it to the input layers, The tokenizer outputs a dictionary with two keys, ***input\_ids*** which contain the tokens, and ***attention\_masks*** for determining if a token is a CLS or pad token. We must name the input layers with the same as the dictionary's keys to be able to use the dictionary as input.
2. Using the pretrained weights of ***roberta-base*** Model***.***
3. Taking the encoders pooled Output of shape [batch\_size, 768].
4. Adding our hidden layers.
5. Adding a dropout layer with the given dropout chance.
6. Finally, adding a binary classifier layer using sigmoid activation.

**Training & Testing**

**Training:**

The values might differ from training different models or using different data but the steps remains the same.

* **Training steps:**

1. Splitting the data using the predefined train test split function.
2. Choosing the batch size and the number of epochs.
3. Creating the optimizer adamw using the optimization library.
4. Setting the initial learning rate, train steps, and warmup steps for the optimizer.
5. Setting up the early stopping to stop training early if the validation loss doesn’t decrease after a defined number of epochs.
6. Setting up the checkpoint function to save the weights.
7. Plot the accuracy live during the training.
8. Load the previously saved weights if any.
9. Fit the model.

**Testing:**

Here we get the results from the model.

* **Testing Outputs:**

1. Evaluate on the test data to get the loss and accuracy for the model using model.evaluate() function.
2. Let the model predict on the test data and get the predicted values then flatten it and pass it to our predefined get predictions function to get the predicted values either 1 or 0.
3. Use those predicted labels and the true labels to get the confusion matrix.
4. Use the confusion matrix to compute the accuracy, precision, recall, specificity, and F-score using the predefined compute metrics function.

**Experiments**

**BERT Model Trails:**

Experiment 1 🡪 Using Preprocessed Data:

- Batch Size: 32

- Epochs: 25

- Steps per Epoch: 1094

- Early Stopping: 5 epochs

* Model Architecture:

- Extra Layers:

- Dropout 10%

- 4 Dense Layers (512, 256, 128, 64)

- Optimizer:

- AdamW

- Initial Learning Rate: 3e-5

- Training Steps: Steps per Epoch \* Epochs

- Warm-Up Steps: 10% of Training Steps

* Training:

- Accuracy: min: 0.770, max: 0.991

- Loss: min: 0.040, max: 0.459

* Validation:

- Accuracy: min: 0.875, max: 0.906

- Loss: min: 0.249, max: 0.550

* Testing :

- Accuracy: 0.8927

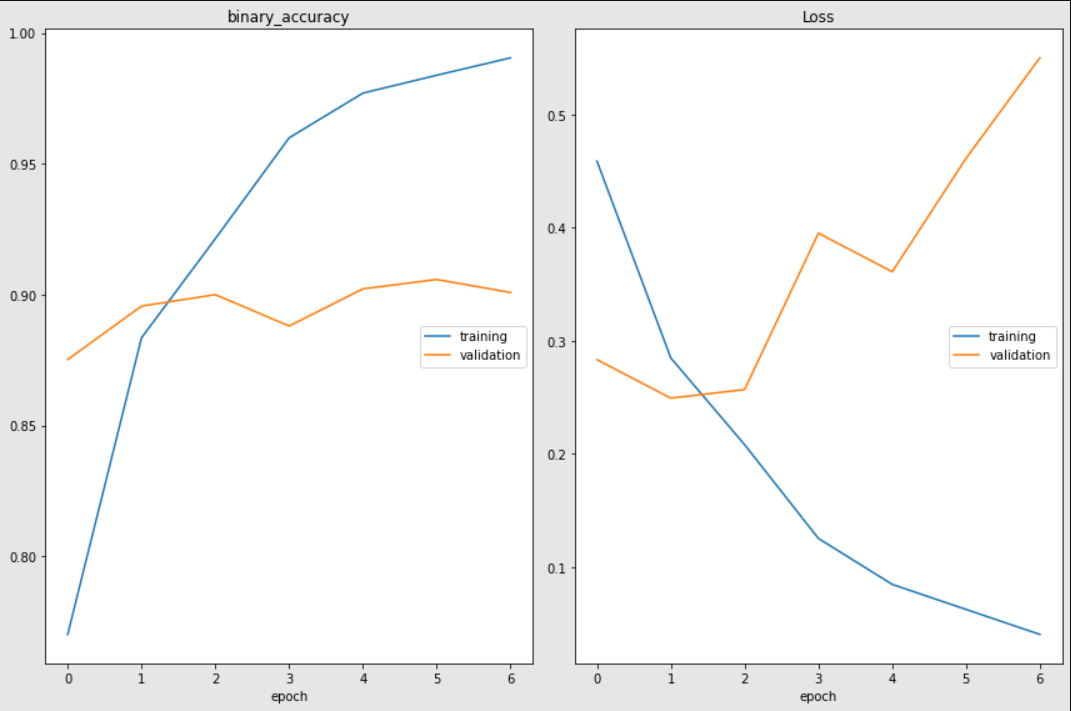
- Loss: 0.2594

- Precision: 0.9235

- Recall: 0.8683

- Specificity: 0.9199

- F-score: 0.8951



Experiment 2 🡪 Using Raw Data:

Batch Size: 32

Epochs: 25

Steps per Epoch: 1094

Early Stopping: 5 epochs

* Model

- Extra Layers:

- Dropout 10%

- 4 Dense Layers (512, 256, 128, 64)

* Optimizer

- AdamW

- Initial Learning Rate: 3e-5

- Training Steps: Steps per Epoch \* Epochs

- Warm-Up Steps: 10% of Training Steps

* Training

- Accuracy: (min: 0.781, max: 0.987)

- Loss: (min: 0.052, max: 0.443)

* Validation

- Accuracy: (min: 0.875, max: 0.893)

- Loss: (min: 0.309, max: 0.532)

* Test

- Accuracy: 0.8787

- Loss: 0.3018

- Precision: 0.8154

- Recall: 0.9313

- Specificity: 0.8384

- F-score: 0.8695

Chart, line chart

Description automatically generated

**Experiment 3 🡪 Using Preprocessed Data :**

Batch Size: 32

Epochs: 10

Early Stopping: 3 Epochs

Steps per Epoch: 1094

* Model**:**

- Extra Layers:

- Dropout 50% After The Dense Layers

- 5 Dense Layers (512, 256, 128, 64, 32)

- Optimizer:

- AdamW

- Initial Learning Rate: 3e-5

- Training Steps: Steps per Epoch \* Epochs

- Warm-Up Steps: 10% of Training Steps

* Training:

- Accuracy: (min: 0.776, max: 0.984)

- Loss: (min: 0.081, max: 0.462)

* Validation:

- Accuracy: (min: 0.883, max: 0.905)

- Loss: (min: 0.253, max: 0.491)

* Testing :

- Accuracy: 0.8994

- Loss : 0.2571

- Precision: 0.9096

- Recall: 0.8898

- Specificity: 0.9092

- F-score: 0.8996

Chart, line chart

Description automatically generated

**Experiment 4 🡪 Using Raw Data :**

Batch Size: 32

Epochs: 10

Early Stopping: 3 Epochs

Steps per Epoch: 1094

* Model**:**

- Extra Layers:

- Dropout 50% After The Dense Layers

- 5 Dense Layers (512, 256, 128, 64, 32)

- Optimizer:

- AdamW

- Initial Learning Rate: 3e-5

- Training Steps: Steps per Epoch \* Epochs

- Warm-Up Steps: 10% of Training Steps

* Training:

- Accuracy: (min: 0.797, max: 0.974)

- Loss: (min: 0.117, max: 0.443)

* Validation:

- Accuracy: (min: 0.873, max: 0.898)

- Loss: (min: 0.289, max: 0.475)

* Testing :

- Accuracy: 0.8787

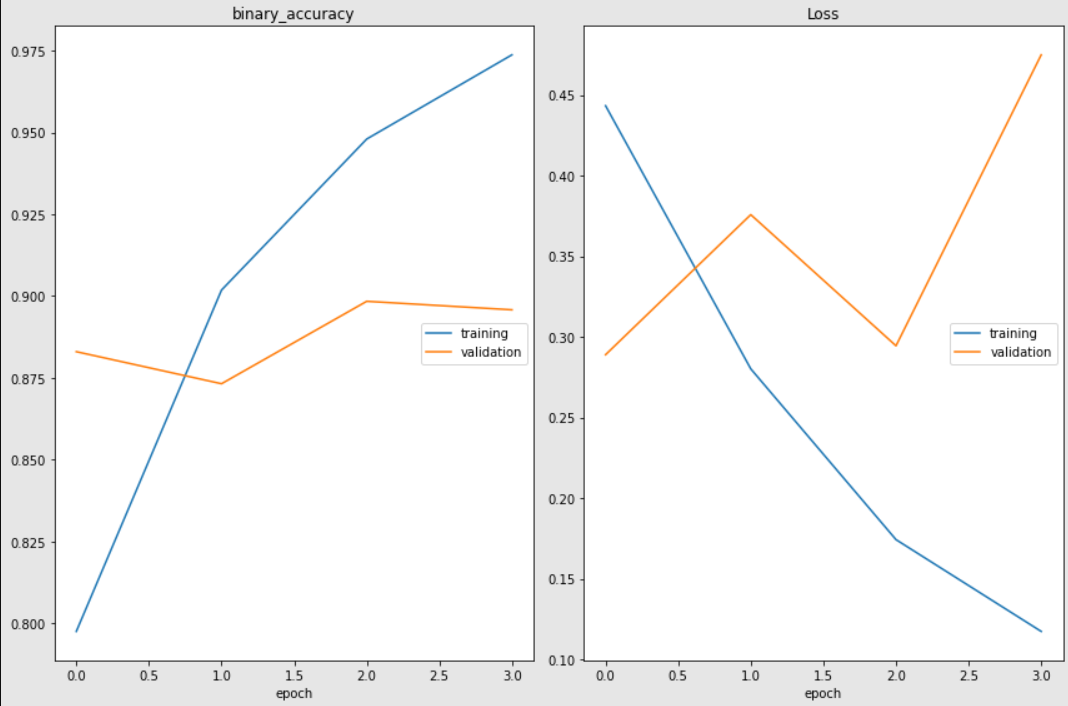
- Loss : 0.3018

- Precision: 0.8153

- Recall: 0.9313

- Specificity: 0.8383

- F-score: 0.8694

**  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Comparisons:**

Best Model: Model of experiment 1

Worst Model: Model of experiment 2

**Roberta Model Trails:**

**Experiment 1 🡪 Using Preprocessed Data :**

Batch Size: 8

Epochs: 10

Early Stopping: 3 Epochs

* Model**:**

- Extra Layers:

- Dropout 50% chance

- 4 Dense Layers (512, 256, 128, 64)

- Optimizer:

- Adam

- Initial Learning Rate: 1e-5

- Clip Norm : 1

* Training:

- Accuracy: (min: 0.872, max: 0.920)

- Loss: (min: 0.250, max: 0.339)

* Validation:

- Accuracy: (min: 0.902, max: 0.908)

- Loss: (min: 0.249, max: 0.276)

* Testing :

- Accuracy: 0.9044

- Loss : 0.2616

- Precision: 0.9025

- Recall: 0.9044

- Specificity: 0.9044

- F-score: 0.9035

**Chart, line chart

Description automatically generated**

**Experiment 2 🡪 Using Raw Data :**

Batch Size: 8

Epochs: 10

Early Stopping: 3 Epochs

* Model**:**

- Extra Layers:

- Dropout 50% chance

- 4 Dense Layers (512, 256, 128, 64)

- Optimizer:

- Adam

- Initial Learning Rate: 1e-5

- Clip Norm : 1

* Training:

- Accuracy: (min: 0.915, max: 0.971)

- Loss: (min: 0.111, max: 0.245)

* Validation:

- Accuracy: (min: 0.937, max: 0.947)

- Loss: (min: 0.176, max: 0.213)

* Testing :

- Accuracy: 0.9377

- Loss : 0.2066

- Precision: 0.9399

- Recall: 0.9348

- Specificity: 0.9406

**Chart, line chart

Description automatically generated** - F-score: 0.9373

**Comparisons:**

Best Model: Model of experiment 2

Worst Model: Model of experiment 1

**Bonus RNN Model** **Architecture**

First we generate the vocabulary of the whole dataset.

* Inside The Generate Vocabulary Class :

1. Split each text by whitespaces and add them to a list
2. Find the words in each review without duplicates
3. Get the total vocabulary of the dataset.

Creating a simple ***RNNClassifier*** Class to define our model specifying the vocabulary and sequence\_length for the ***TextVectorization*** layer, embedding\_output\_dim for the ***embedding*** layer and lstm\_units, dropout and recurrent\_dropout for the **LSTM** layer.

### Inside The Bert Classifier Class :

* 1. Adding the first layer which has to be of size 1 and string data type.
  2. Adding the TextVectorization layer.
  3. Adding the embedding layer.
  4. Adding LSTM layer.
  5. Flattening the output of the previous layer.
  6. Add the output layer with a single sigmoid unit.

**Training:**

* **Training steps:**

1. Splitting the data using the predefined train test split function.
2. Getting the vocabulary of the training data
3. Choosing the batch size and the number of epochs.
4. Creating the optimizer adamw using the optimization library.
5. Setting the initial learning rate, train steps, and warmup steps for the optimizer.
6. Setting up the early stopping to stop training early if the validation loss doesn’t decrease after a defined number of epochs.
7. Setting up the checkpoint function to save the weights.
8. Plot the accuracy live during the training.
9. Load the previously saved weights if any.
10. Fit the model.

**Testing:**

* **Testing Outputs:**

1. Evaluate on the test data to get the loss and accuracy for the model using model.evaluate() function.
2. Let the model predict on the test data and get the predicted values then flatten it and pass it to our predefined get predictions function to get the predicted values either 1 or 0.
3. Use those predicted labels and the true labels to get the confusion matrix.
4. Use the confusion matrix to compute the accuracy, precision, recall, specificity, and F-score using the predefined compute metrics function.

**Experiments**

**RNN Using Preprocessed Data 🡪 Experiment 1:**

Batch Size: 32

Epochs: 25

Early Stopping: 3 Epochs

Steps per Epoch: 547

**Model:**

**- Embedding Layer:**

- input size = 37500

- output size = 128

**- LSTM Layer:**

- units = 128

- activation = tanh

**- Extra Layers:**

- No extra layers

- no dropout

- no recurrent dropout

**- Optimizer:**

- AdamW

- Initial Learning Rate: 3e-5

- Training Steps: Steps per Epoch \* Epochs

- Warm-Up Steps: 10% of Training Steps

**Training:**

Accuracy: (min: 0.504, max: 0.628)

Loss: (min: 0.660, max: 0.693)

**Validation:**

Accuracy: (min: 0.505, max: 0.629)

Loss: (min: 0.659, max: 0.694)

**Test:**

Accuracy: 0.6335

Loss: 0.6567

Precision: 0.6998

Recall: 0.6208

Specificity: 0.6503

F-score: 0.6579

**A picture containing diagram

Description automatically generated**

**RNN Using Preprocessed 🡪 Experiment 2:**

Batch Size: 32

Epochs: 15

Early Stopping: 3 Epochs

Steps per Epoch: 547

**Model:\***

**- Embedding Layer:**

- input size = Length of vocabulary

- output size = 128

**- LSTM Layer:**

- units = 128

- activation = tanh

- Dropout: 0.25

- Recurrent Dropout: 0.5

**- Extra Layers:**

- No extra layers

- no dropout

- no recurrent dropout

**- Optimizer:**

- AdamW

- Initial Learning Rate: 5e-5

- Training Steps: Steps per Epoch \* Epochs

- Warm-Up Steps: 10% of Training Steps

**Training:**

Accuracy: (min: 0.509, max: 0.944)

Loss: (min: 0.154, max: 0.693)

**Validation:**

Accuracy: (min: 0.529, max: 0.893)

Loss: (min: 0.265, max: 0.691)

**Chart

Description automatically generated  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
RNN Using RAW Data 🡪 Experiment 3:**

Batch Size: 64

Epochs: 10

Early Stopping: 3 Epochs

Steps per Epoch: 547

**Model:**

**- Embedding Layer:**

- input size = Length of vocabulary

- output size = 128

**- LSTM Layer:**

- units = 128

- activation = tanh

- Dropout: 0.25

- Recurrent Dropout: 0.5

**- Extra Layers:**

- No extra layers

- no dropout

- no recurrent dropout

**- Optimizer:**

- AdamW

- Initial Learning Rate: 5e-5

- Training Steps: Steps per Epoch \* Epochs

- Warm-Up Steps: 10% of Training Steps

**Training:**

Accuracy: (min: 0.517, max: 0.933)

Loss: (min: 0.175, max: 0.692)

**Validation:**

Accuracy: (min: 0.500, max: 0.880)

Loss: (min: 0.304, max: 0.688)

**Test:**

Accuracy: 0.8836

Loss: 0.3015

**Chart

Description automatically generated** Precision: 0.8898

Recall: 0.8771

Specificity: 0.8902

F-score: 0.8834

**Comparisons:**

Best Model: Model of experiment 2

Worst Model: Model of experiment 1

**Colab Link:** **https://colab.research.google.com/drive/1TGkrYNL5UQI0iY154UEThWKj7swXPL3D?usp=sharing**