# Project report

Deep learning

Yanal Serhan 327680492

Salam Qais 327876116

**התאריך להגשת הפרוייקט 21/04/2024**
ההגשה תכלול דו"ח שמתאר את המטרה, נתונים, ארכיטקטורה, שיטת הבדיקה ותוצאות (כולל פלוטים של train ו- validation loss ) צריך לצרף קוד, דטה, והסבר על איך להריץ. למעוניינים להגיש לפני תאריך הגשה נקיים פגישת זום מיחדת להצגת הפרוייקט.

**מצגות פרוייקטים:** נקיים מצגות של הפרוייקטים אחרי ההגשה. יש להכין מצגת של 10-15 דק שמסכמת את נקודות החשובות בדו"ח ולהיות מוכנים לענות על שאלות לגבי קוד.

# Introduction

The goal of the assignment is to detect the hate speech in a given data, meaning, given a csv file containing the content of the sentence and the label (if it's hate speech or not), our goal is to train a model (actually 2 models; BERT and LSTM) on that data in order to classify whether a sentence (or data in other csv file) is hate speech or not (or at least be as accurate as possible). As well as, eventually comparing between the LSTM model and the BERT model in terms of various factors such as training time, losses, accuracies, etc.

Given the data files:

1. HateSpeechDataset.csv
2. HateSpeechDatasetBalanced.csv

**Note:**

We just used this file HateSpeechDatasetBalanced, Because HateSpeechDataset ,

Is contained in HateSpeechDatasetBalanced, in addition to having a nearly equal number of samples for the two labels (0-Not-hate-speech,1-hate speach) almost the same number.

The labels are balanced, so there's no need to worry about dividing them equally.

This is small code show us the number of each label in the HateSpeechDatasetBalanced file(almost the same number.)

```
import pandas as pd
data_full = pd.read_csv('HateSpeechDatasetBalanced.csv')


data_full.groupby('Label').count()

```

| Label | Content |
|---|---|
| 0 | 361594 |
| 1 | 364525 |

## Structures:

1. **For the first network - LSTM:**
    - We started by adding embedding layer to convert the input (tokens represented as integers) to a vector of size embedding_dim=300.

    - After the embedding layer , we added a dropout layer with (dropout=0.3), this can help in preventing overfitting by regularizing the data by randomly (with 0.3 probability) resetting a part of the input to 0.

    - After the dropout layer there's bidirectional LSTM layer (to capture information from the past as well as the future) with droput rate = 0.1 (in order to regularize the data and reduce the overfitting), as well as consisting of num_layers (which we set to 1) layers. Each of these layers has hidden_dim (which we set to 128) hidden units.

    - Then, an attention layer (focuses on the relevant part of the input), followed by the fully connected layer (to perform the classification).

    In the forward method:

    - We started by embedding the input text, then applying dropout layer

    - Afterwards, we know that the seq is padded we pack it to ignore the padded values(we padded it to handle the inputs with different length)

    - Then passing it through the LSTM layer, then unpacking it(paddeding again

    - Then, we applied the attention layer to the output (a linear layer to the output of the LSTM layer we got from the previous step followed by the softmax activation to get the updated weights, then applying the weighted sum on the output).

    - Afterwards, we repeated the previous step on the output we got from it.

    - Then, passing the output to the fully connected layer then returning it.

    Note: we used Adam `optimizer , CrossEntropyLoss`

**2. For the second Network -BERT:**

- We Loaded the the pre-trained BERT model 'bert-base-uncsed'.
- After loading the pre-trained BERT model, we added a binary classification layer (the output is 2) on top to adapt the model for hate speech detection.
- Afterwards, freeze the parameters (turning the requires_grad flag to False
- After the freezing phase, we fine tune the parameters of the last 2 layers (turning the requires_grad flag to True for all parameters in the last 2 layers while the rest of the parameters still frozen, in this way, we can shift the training to the parameters of the last 2 layers).

  by default, the BERT model consists of 12 hidden layers. To fine-tune the model for our specific task, freezed the parameters of 10 hidden layers while allowing the last two layers to remain trainable.

  Freezing the parameters of these hidden layers helps in reducing the computational of the Network, while still allowing the Network to adapt to the hate speech detection task using the last two layers.

In the forward method:

- Firstly, we apply a BERT layer on the input values (tokenized input sequence represented as integers and binary tensor indicates which tokens should be ignored and which not, if the value passed is None, then all the tokens will be considered).
- Afterwards, we apply pooler_output on the output from the previous step, which is fixed size vector representation of the entire input obtained from the non-None values (meaning, the non-padded values of the input vector) at the last layer of the BERT.
- Finally, we pass the output of the previous step to a fully connected layer which classifies whether the sequence is hate speech or not.

Note: we used Adamw `optimizer , CrossEntropyLoss a`lso

**Checking and results methods:**

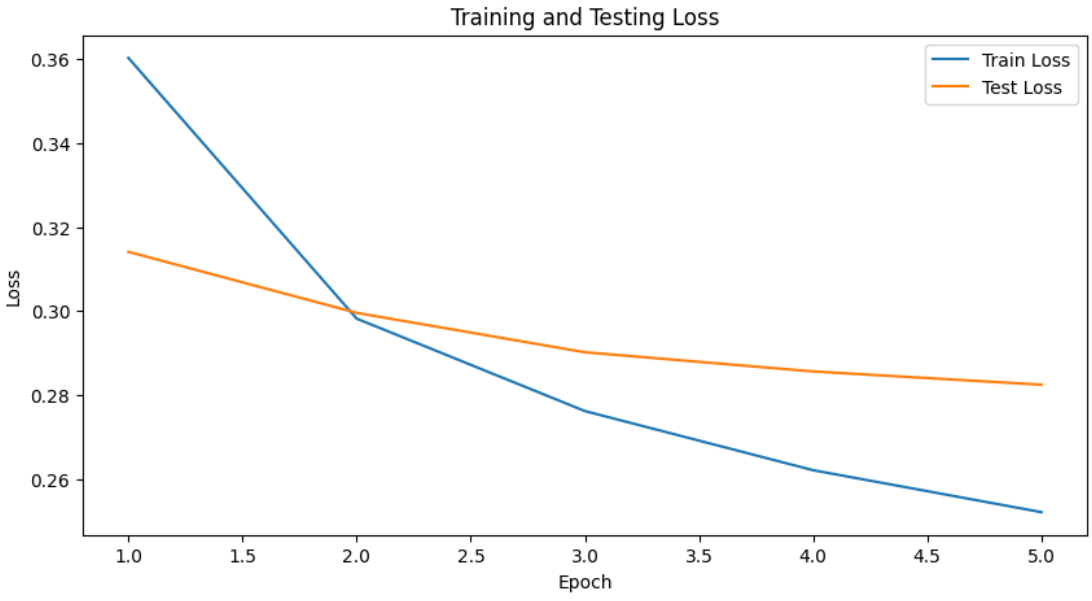
1. Checking method:

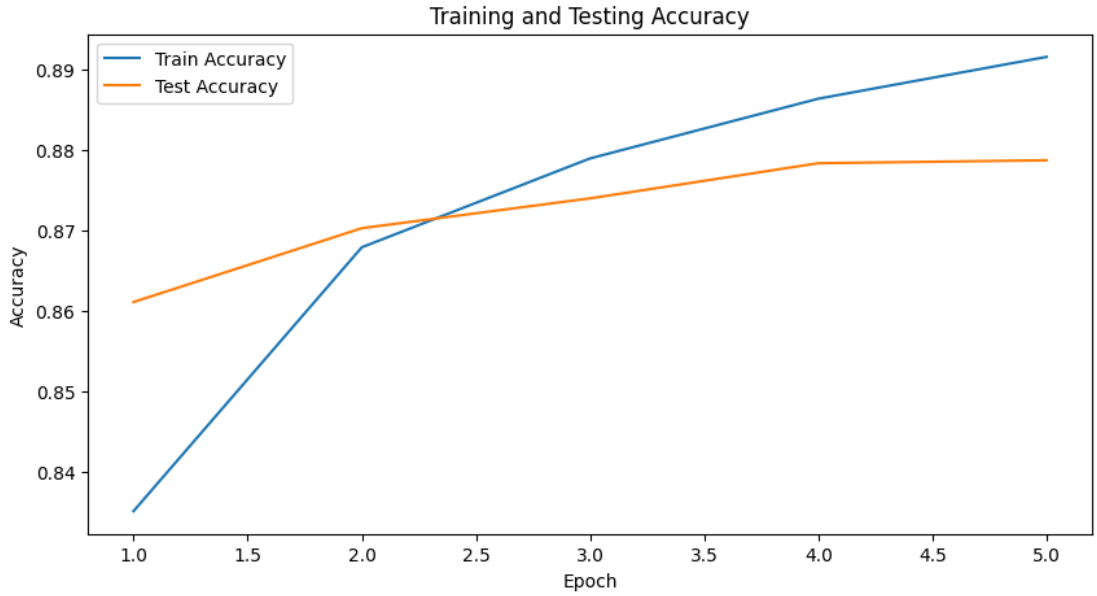
   First of all, we loaded the data HateSpeechDatasetBalanced then separated it into train and validation sets using the train-test split with a ratio 8:2. Then, preprocessed the data by tokenizing it and created the tensor of indices.

2. Results:

   We go the results by training the model on our train set, then testing it on our validation set. And plotted the diagrams as well as printing the accuracies.

For the LSTM:






Training and Test loss:

| Epoch_number | Train Loss | Test Loss |
|---|---|---|
| 1 | 0.36026043525398754 | 0.3141002803876745 |
| 2 | 0.2982016594375628 | 0.29956893817296626 |
| 3 | 0.2761985036736738 | 0.29018885547054507 |
| 4 | 0.2621177564475171 | 0.28562925597036154 |
| 5 | 0.2521370776388935 | 0.2824707519045717 |

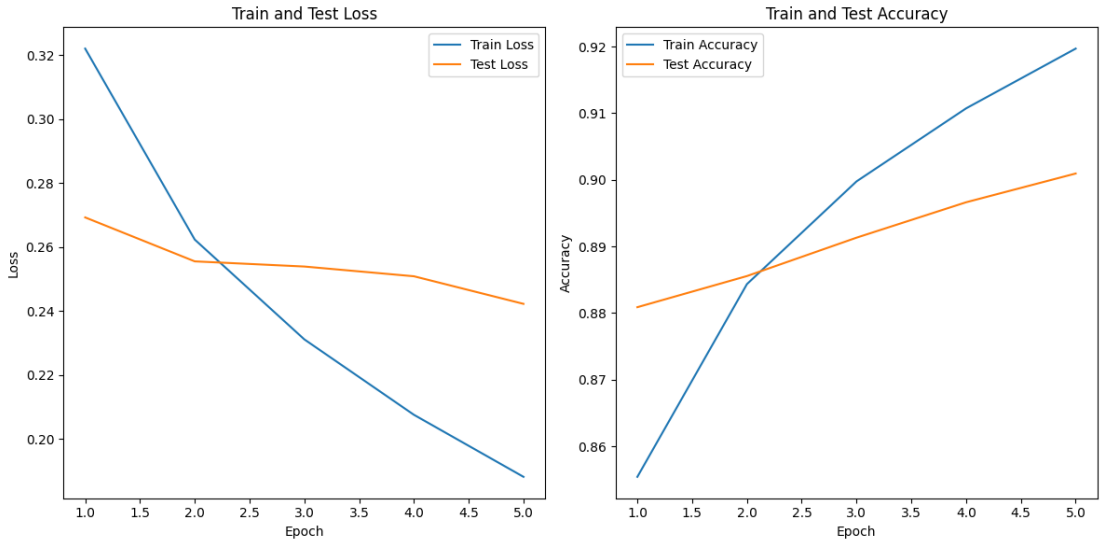
We can see that both of them decreased all the time ,and both of them have Close values which say that model improving without over fitting(not just the train improved).

Train and Test accuarcy:

| Epoch_number | Train acc | Test acc |
|---|---|---|
| 1 | 0.8351939679287995 | 0.861124882939459 |
| 2 | 0.8679158884135687 | 0.8702831487908335 |
| 3 | 0.8789350915397791 | 0.8739808847022531 |
| 4 | 0.8863426264643353 | 0.8783396683743734 |
| 5 | 0.8915277287633737 | 0.8787046218255936 |

We can see that both of them increased all the time , which say that model improving without overfitting(not just the train improved).

For the BERT:




1-Training and Test loss:

| Epoch_number | Train Loss | Test Loss |
|---|---|---|
| 1 | 0.3219865978338273 | 0.26922870540766314 |
| 2 | 0.26229755542834776 | 0.2554926824091821 |
| 3 | 0.2311121550147659 | 0.253879993314124 |
| 4 | 0.20754157328065584 | 0.25083594301159284 |
| 5 | 0.18815330549964557 | 0.2422143055576855 |

We can see that both of them decreased all the time , which say that model improving without over fitting(not just the train improved).

1-Train and Test accuarcy:

| Epoch_number | Train acc | Test acc |
|---|---|---|
| 1 | 0.8554076037838163 | 0.8808805707045667 |
| 2 | 0.8843474293977397 | 0.8855561064286894 |
| 3 | 0.8997701822188175 | 0.8913196165922987 |
| 4 | 0.9107360194183114 | 0.8966355423346003 |
| 5 | 0.9197083810327168 | 0.9009530105216769 |

We can see that both of them increased all the time , which say that model improving without overfitting(not just the train improved).

Comparing:

1-Module speed: It took approximately 3 hours to train the LSTM model. In contrast, training the BERT model took around 10 hours. Clearly, BERT required more time. However, it's important to note that we didn't write accurate numbers because. Sometimes, the GPU allocation on Google Colab ended, leading us to continue running the models on the following day, so this is approximations..

2-Number of parmetrs

The Lstm Network have 21200191(21 million) parmetet but the Bert have 14177282 (14 million ) both of them have huge numbers of parmeters but the Lstm have more tha the bert with 7 million.

3- Test and train loss

| LSTM | | |
|---|---|---|
| Epoch | Train Loss | Test Loss |
| 1 | 0.36026043525398754 | 0.3141002803876745 |
| 2 | 0.2982016594375628 | 0.29956893817296626 |
| 3 | 0.2761985036736738 | 0.29018885547054507 |
| 4 | 0.2621177564475171 | 0.28562925597036154 |
| 5 | 0.2521370776388935 | 0.2824707519045717 |

| BERT | | |
|---|---|---|
| Epoch | Train Loss | Test Loss |
| 1 | 0.3219865978338273 | 0.26922870540766314 |
| 2 | 0.26229755542834776 | 0.2554926824091821 |
| 3 | 0.2311121550147659 | 0.253879993314124 |
| 4 | 0.20754157328065584 | 0.25083594301159284 |
| 5 | 0.18815330549964557 | 0.2422143055576855 |

**We can say obviously that Bert give better results from the first epoch till the last epoch in both of train and test loss.**

**4-** Test and train acc

| LSTM | | |
|---|---|---|
| Epoch_number | Train acc | Test acc |
| 1 | 0.8351939679287995 | 0.861124882939459 |
| 2 | 0.8679158884135687 | 0.8702831487908335 |
| 3 | 0.8789350915397791 | 0.8739808847022531 |
| 4 | 0.8863426264643353 | 0.8783396683743734 |
| 5 | 0.8915277287633737 | 0.8787046218255936 |

| BERT | | |
|---|---|---|
| Epoch_number | Train acc | Test acc |
| 1 | 0.8554076037838163 | 0.8808805707045667 |
| 2 | 0.8843474293977397 | 0.8855561064286894 |
| 3 | 0.8997701822188175 | 0.8913196165922987 |
| 4 | 0.9107360194183114 | 0.8966355423346003 |
| 5 | 0.9197083810327168 | 0.9009530105216769 |

Also, here we obviously see that Bert give acc better than the LSTM, and in the last epoch in the tests accuracy, the LSTM arrived to 87 %, but the LSTM arrived to 90 % knowing that the two models being testing in the same number of epochs so the comparing now is more reasonable and accurate.

## Conclusion:

In conclusion, While LSTM offers a simpler architecture and faster training times, BERT provides better accuracy .in addition, using Bert is more easy to user; as almost don't need to build anything.

So there's a trade off between the simplicity of the model (and the training time) to the accuracy it gives.

**How to run:**

Open Test_HateSpeach.ipynp

We put a Drive Link in Test_HateSpeach, open this Link then download the

Files FTBertTrained.pth, LSTMTrained.pth, HateSpeechDatasetBalanced.csv

Then load the files that u downloaded

LSTM:

To run the LSTM -run the First cell in Test_HateSpeach.ipynp

BERT:

To run the BERT -run the Seconed cell in Test_HateSpeach.ipynp

Notes:

- In the provided link, in the test file, we used only about 10000 samples of the data (which is small portion of the actual data) which is around 0.05% of the data due to the long run time takes to complete. (we originally divided the samples into 8:2 portion, as said before), thus, the accuracy may be reduced a little bit than the results we provided.

## What we submitted:

**We have subimitted 3 files ipynp**

**1)Lstm_HateSpeachDetection.ipynp: have the train of the Lstm**

**1)Bert_HateSpeachDetection.ipynp: have the train of the BERT**

**3)Test_HateSpeach: Testing both of them**

**In one of the files there's a link to google drive which contains the trained data (for the 2 models) and the actual data.**