Title: Detection of Propaganda Techniques in News Articles
Members: Zahra Bashir, Omid Ghahroodi

Abstract:

Propaganda detection is one of the most important issues in the NLP field today.In this project, we are going to step toward solving it using Deep Neural Networks.

Introduction:

The overall goal of this task is to produce models capable of detecting text fragments in which propaganda techniques are used in a news article.
We use the prepared corpus of about 550 news articles in which fragments containing one out of 18 propaganda techniques have been annotated. This prepared corpus is available on SemEval2020 website.

Challenges:

- Lack of proper data: We did not have enough data
  - As we know in order to train such a system, we need much more data.
  So, we had to make our networks more and more accurate and precise in order to cover this problem.
- Absence of test dataset: we evaluated our data on the "validation set". Whenever the SemEval 2020 releases the test data, we will apply it on our project.

Related work/Background:

The contest is currently being held and this is almost a new issue.

But there are some efforts on "*Fake News*", which solves its problem with somehow the same approach. As we know, fake news is a type of propaganda where disinformation is intentionally spread through news outlets and/or social media outlets.
It uses the BERT algorithm to predict if a news report is fake.
BERT stands for Bidirectional Encoder Representations from Transformers. The mentioned paper describes the BERT algorithm which was published by Google. BERT works by randomly masking word tokens and representing each masked word with a vector based on its context. The two applications of BERT are “pre-training” and “fine-tuning”.

Proposed method:

Preprocessing:

We only parsed the dataset and converted each word to a number like other text-related tasks.

It has to be mentioned that we had 2 main phases in this project.

Phase 1: Given a plain-text document, identifying whether a text contains propaganda or not. This is a binary sequence tagging task. We refer to it as SI (Span Identification).

Phase 2: Given a text fragment, identify the applied propaganda technique in the fragment. This is a multi-class classification task. Although the data has been annotated with 18 techniques, given the relatively low frequency of some of them, we decided to merge similar underrepresented techniques into one superclass:

- Bandwagon and Reductio ad Hitlerum into "Bandwagon,Reductio ad Hitlerum"
- Straw Men, Red Herring and Whataboutism into "Whataboutism,Straw_Men,Red_Herring"

and to eliminate "Obfuscation,Intentional Vagueness,Confusion".
Therefore this is a 15-classes classification task, which we refer to as TC (Technique Classification).
15 classes = 14 Techniques + No Propaganda class

Architecture

We used Deep Neural Architecture in order to solve this problem. As we have text data and the words in an article or sequence are relevant, therefore we must use networks which own memory. LSTM is the simplest network which has memory.
We tried to implement a baseline and then we tried to improve it:

The BaseLine Accuracy : 83%

---

1. SI TASK:

First experiment:
Using a Bidirectional LSTM and a Dense layer for the output
We chose a **Bidirectional LSTM** because it helps the network to learn the sentence from reverse.
Embedding Size: 128

```
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

We used checkpointers in order to record some history.

Compiling attributes:

model.compile('adam', 'mse', metrics=["accuracy",f1 ,recall,precision])

Why 'sigmoid'? Binary Classification

Why Adam? we use Adam because of its generalization. It is better than most of the optimizers

Second experiment:
Then we tried to expand our network using more LSTM layers, and also more Fully Connected ones.
Using 2 Bidirectional LSTM + Dropout + 3 Dense Layers
Embedding Size: 128

```
=================================================================
embedding_5 (Embedding)      (None, None, 128)         128000
_________________________________________________________________
bidirectional_8 (Bidirection (None, None, 256)         263168
_________________________________________________________________
bidirectional_9 (Bidirection (None, 256)               394240
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 256)               65792
_________________________________________________________________
dense_12 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 129
=================================================================
Total params: 884,225
Trainable params: 884,225
Non-trainable params: 0
```

```
    model.add(Bidirectional(LSTM( units=128, dropout=.3, return_sequences=True,
kernel_initializer='he_normal')))
    model.add(Bidirectional(LSTM( units=128, dropout=.2,
kernel_initializer='he_normal')))
    model.add(Dropout(0.5))
    model.add(Dense(256, activation='relu', kernel_initializer='he_normal'))
    model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
    model.add(Dense(1, activation='sigmoid', kernel_initializer='he_normal'))
```

In the first LSTM, we have to set return_sequences=True due to the fact that we need a sequence to be referenced to the second LSTM

Why we used 'he_normal' as a kernel_initilizer?

In this method, the weights are initialized keeping in mind the size of the previous layer which helps in attaining a global minimum of the cost function faster and more efficiently.The weights are still random but differ in range depending on the size of the previous layer of neurons. This provides a controlled initialization hence the faster and more efficient gradient descent.

Again, We used checkpointers in order to record some history.

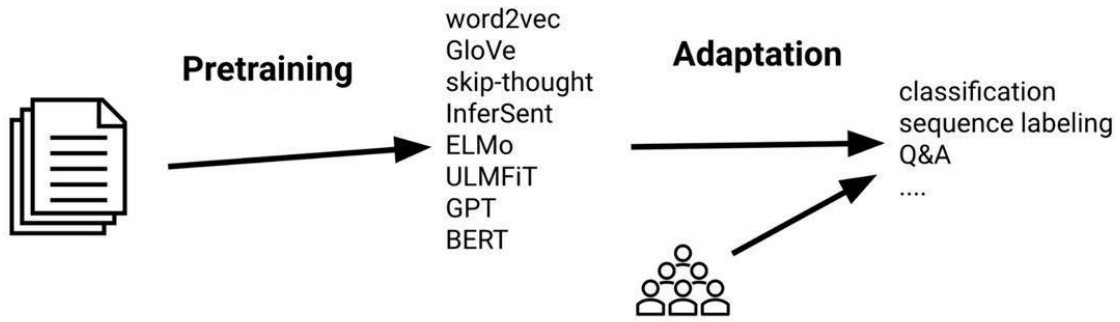
Compiling attributes:

```
    model.compile('adam', 'binary_crossentropy', metrics=["accuracy",f1 ,recall,
precision])
```

Why binary_crossentropy? *Binary Classification*

The Third experiment:

While reviewing the results of the aforementioned approaches, we understood that due to the lack of data, we have to use a pre-trained model using Transfer Learning.



So we decided to use GloVe word embeddings which stands for Global Vectors for Word Representation. . It's a somewhat popular embedding technique based on factorizing a matrix of word co-occurence statistics.
Specifically, we will use the 100-dimensional GloVe embeddings of 400k words computed on a 2014 dump of English Wikipedia.

This is the link:https://nlp.stanford.edu/projects/glove/

Using the mentioned Embedding, we embedded our dataset into vectors, then we fed it to our network. By the following code, we used the frozen network weights.

```
embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,
                              trainable=False)
```

=

2. TC TASK

First experiment:
Using a Bidirectional LSTM and a Dense layer for the output

```
EMbedding Size: 128
    model = Sequential()
    model.add(Embedding(num_word+1, 128, input_length=129))
    model.add(Bidirectional(LSTM(64, dropout=.4)))
    model.add(Dense(15, activation='softmax'))

15 = Num of Classes (Not Propaganda + 14 Types)
We used checkpointers in order to record some history.
Compiling attributes:
```

```
model.compile('adam', 'categorical_crossentropy', metrics=["accuracy",f1 ,recall, precision])
```

Why 'softmax'?

Why Adam?

Why 'categorical_crossentropy'?

Second experiment:

After the first one we tried to expand our network using more LSTM layers, and also more Fully Connected ones.

Using 2 Bidirectional LSTM + Dropout + 3 Dense Layers

Embedding Size: 128

```
  model.add(Embedding(num_word+1, 128, input_length=129))
    model.add(Bidirectional(LSTM( units=128, dropout=.4, return_sequences=True, kernel_initializer='he_normal')))
    model.add(Bidirectional(LSTM( units=128, dropout=.4, kernel_initializer='he_normal')))
    model.add(Dense(256, activation='relu', kernel_initializer='he_normal'))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
    model.add(Dense(15, activation='softmax', kernel_initializer='he_normal'))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 129, 128)          2879872
_________________________________________________________________
bidirectional_7 (Bidirection (None, 129, 256)          263168
_________________________________________________________________
bidirectional_8 (Bidirection (None, 256)               394240
_________________________________________________________________
dense_10 (Dense)             (None, 256)               65792
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_12 (Dense)             (None, 15)                1935
=================================================================
Total params: 3,637,903
Trainable params: 3,637,903
Non-trainable params: 0
_________________________________________________________________
```

Compile attributes:

```
model.compile('adam', 'categorical_crossentropy', metrics=["accuracy",f1 ,recall, precision])
```

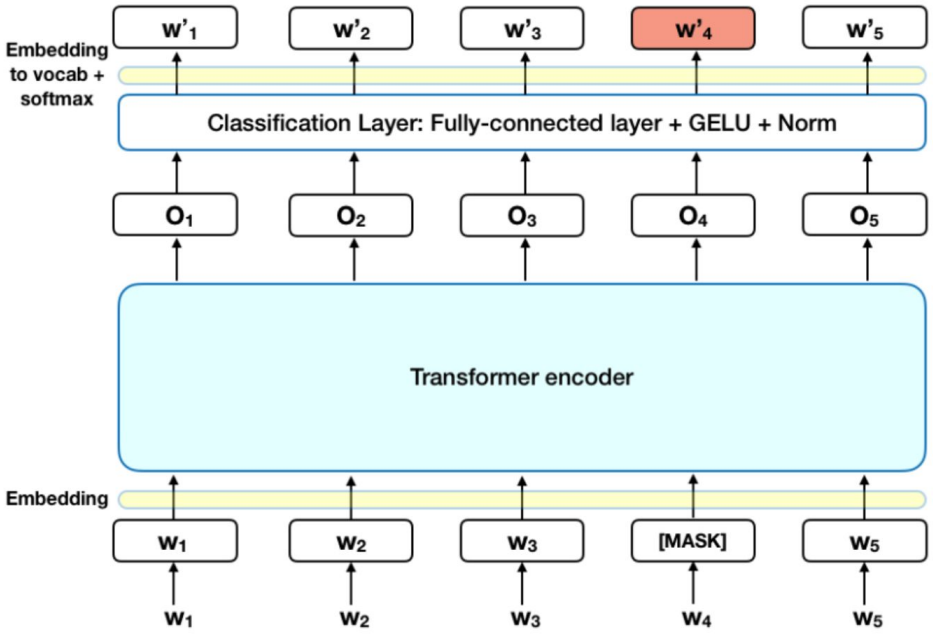
Third Experiment:

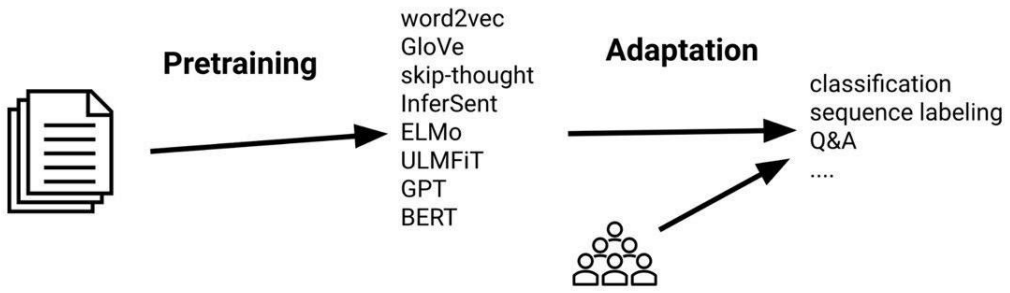
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 129, 128)          2879872
_________________________________________________________________
bidirectional_9 (Bidirection (None, 129, 256)          197376
_________________________________________________________________
bidirectional_10 (Bidirectio (None, 256)               295680
_________________________________________________________________
dense_10 (Dense)             (None, 256)               65792
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_12 (Dense)             (None, 15)                1935
=================================================================
Total params: 3,473,551
Trainable params: 3,473,551
Non-trainable params: 0
_________________________________________________________________
Train on 16653 samples, validate on 100 samples
```

2 GRU bidirectional + Dense + Dropout + Dense
We ran it on 3 epochs due to its being time-consuming.

Fourth experiment:

After all, we again understood that these layers are not enough for us and they are not performing as well as a pre-trained model. So, we used BERT.





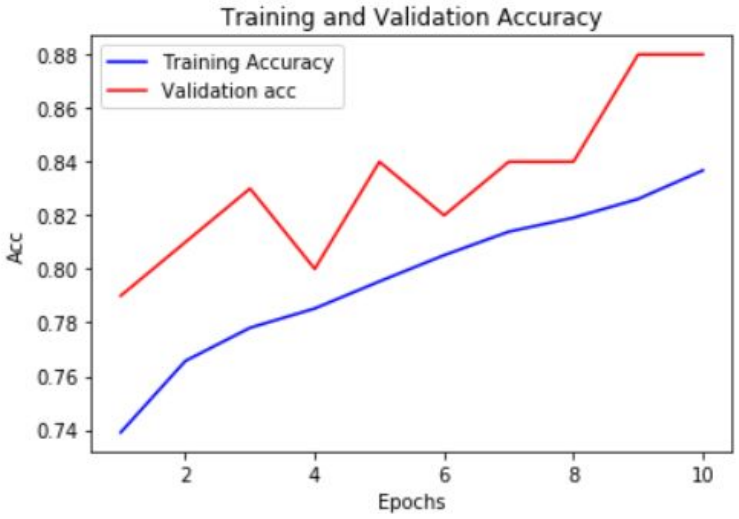
Results:

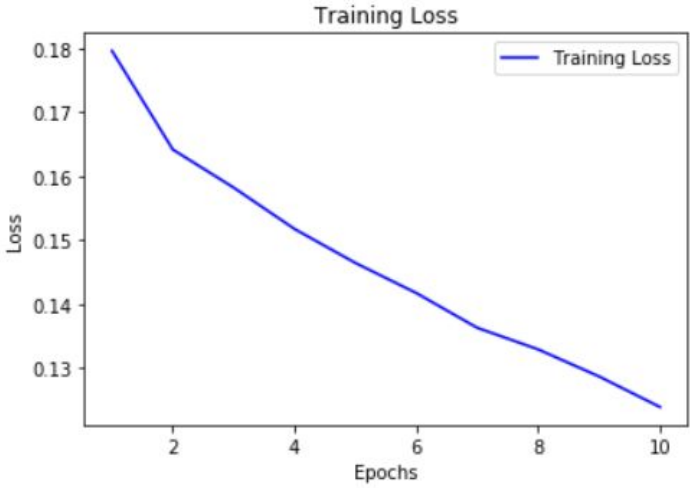
SI Task:

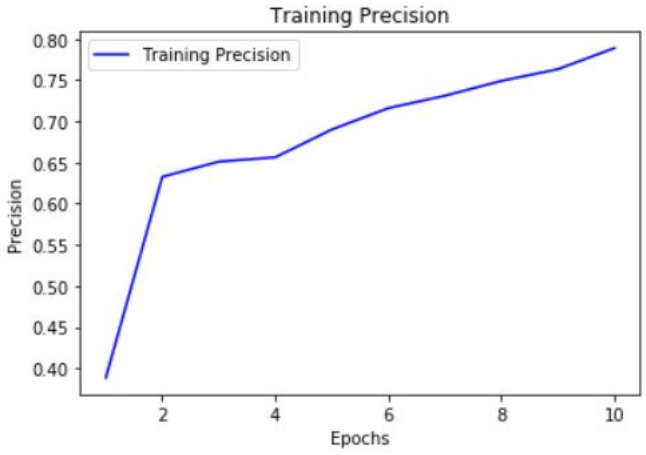
First experiment: (The BaseLine)

```
Training Acc:0.8367861646621756
Validation Acc:0.8367861646621756
Final Loss:0.12384144564272799
Final Precision:0.788950335941522
Final Recall:0.5304879386120929
```

Train Accuracy	Val Accuracy	Loss	Precision	Loss
0.835362	0.83	0.123965	0.783765	0.52787963

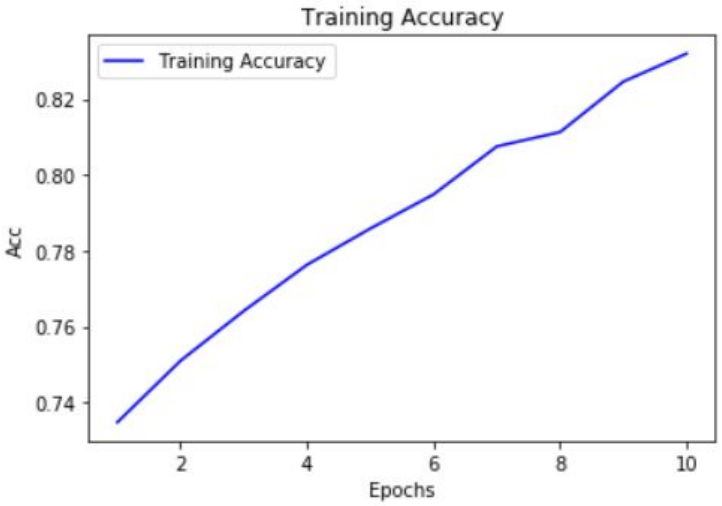




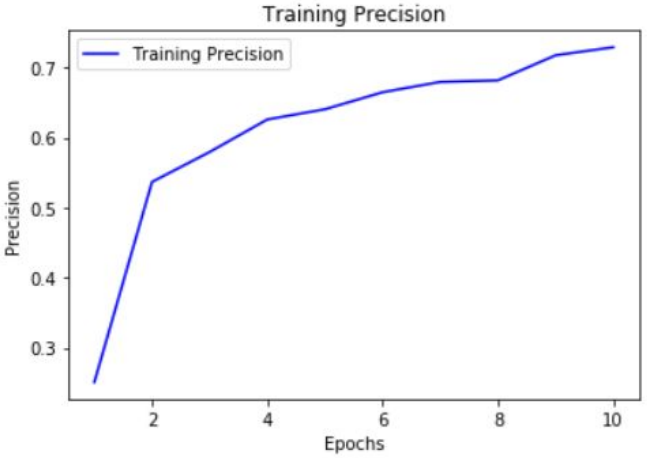


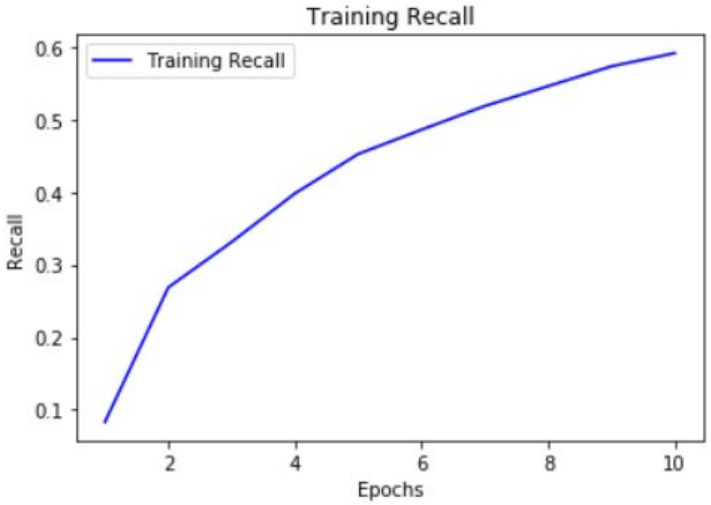
Second experiment:

Accuracy	Loss	Precision	Loss
0.831943	0.367651	0.729285	0.5913584


Training Accuracy
Training Accuracy
Acc
0.82
0.80
0.78
0.76
0.74
2
4
6
8
10
Epochs


Training Loss
Training Loss
Loss
0.550
0.525
0.500
0.475
0.450
0.425
0.400
0.375
2
4
6
8
10
Epochs


Training Precision
Training Precision
Precision
0.7
0.6
0.5
0.4
0.3
2
4
6
8
10
Epochs



Third experiment:

Global Vectors for Word Representation

Running it was Time-consuming so we ran it on two epochs:

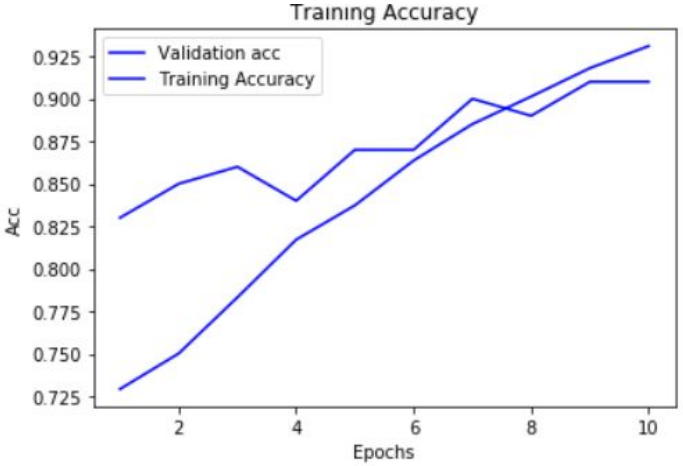
Training Accuracy	Vali Accuracy	Loss	Precision	Recall
0.74	0.65	0.9	0.8	0.63

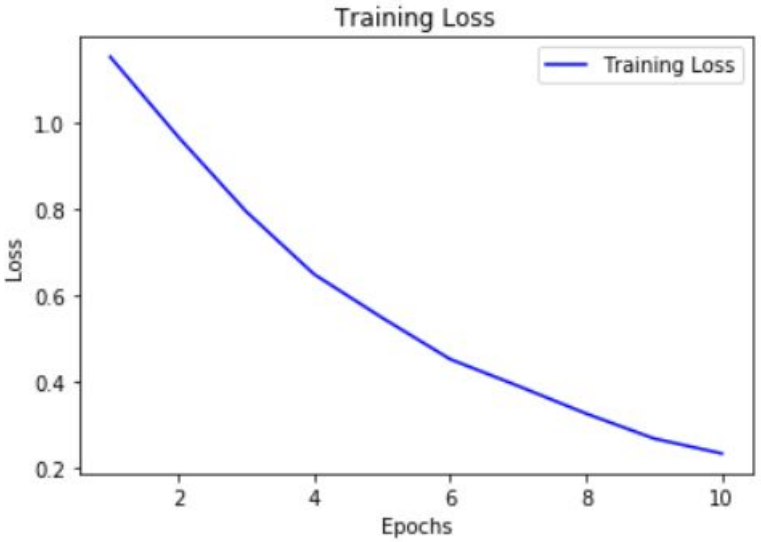
Since we limited the epochs, so we expect to have higher accuracy and precision when we apply epochs = 10.

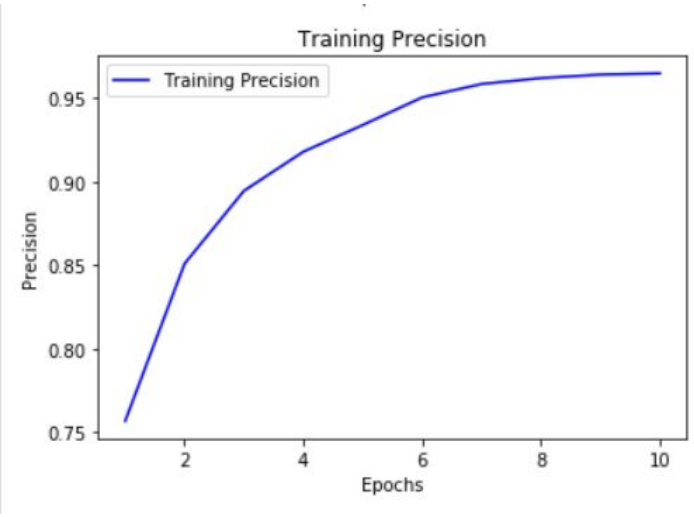
TC Task:

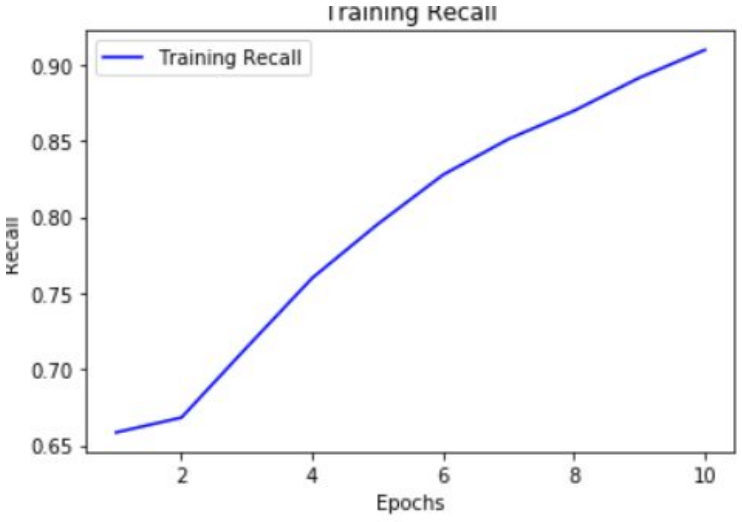
First experiment:

Training Accuracy	Vali Accuracy	Loss	Precision	Recall
0.925	0.90	0.2	0.96	0.90



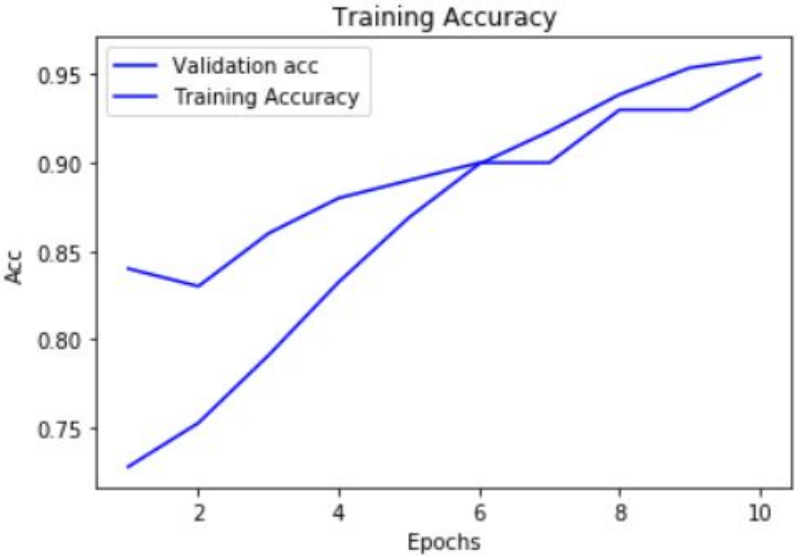


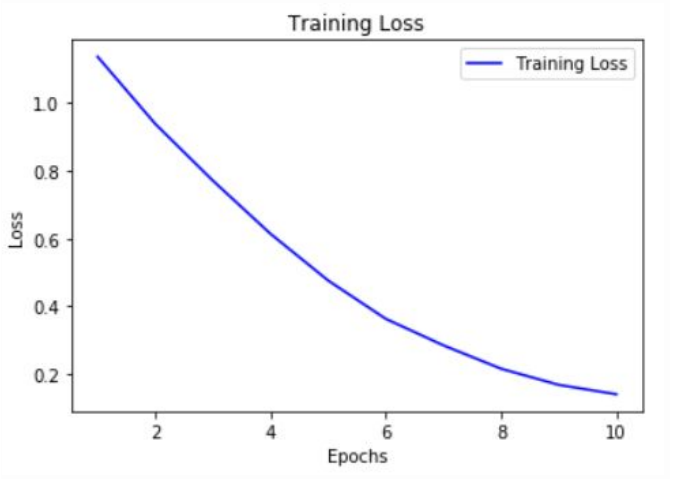


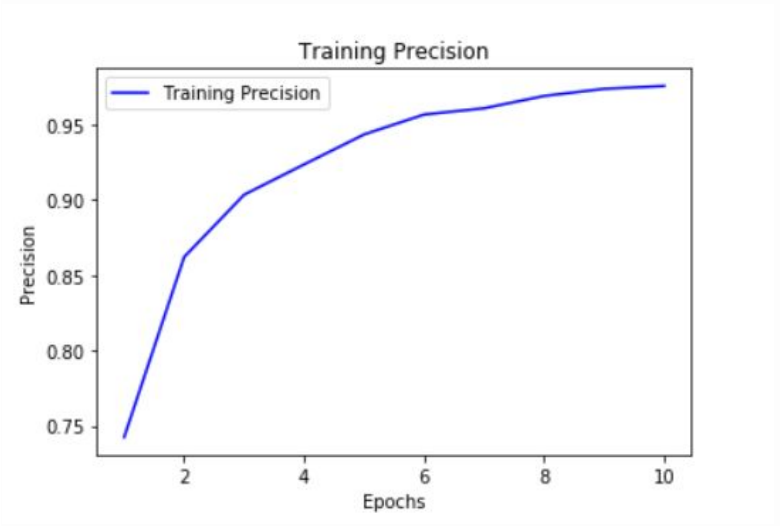


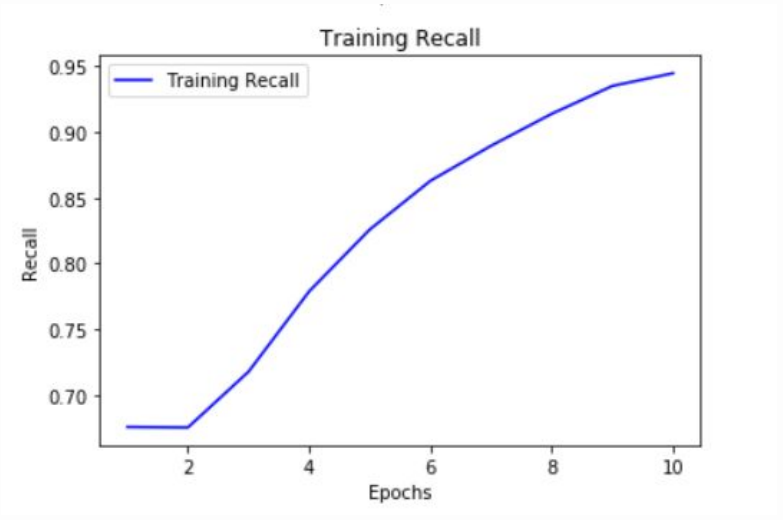
Second experiment:

Training Accuracy	Vali Accuracy	Loss	Precision	Recall
0.95	0.95	0.14	0.93	0.90


Training Accuracy
Validation acc
Training Accuracy
Acc
Epochs
0.95
0.90
0.85
0.80
0.75
2
4
6
8
10


Training Loss
Training Loss
Loss
Epochs
1.0
0.8
0.6
0.4
0.2
2
4
6
8
10


Training Precision
Training Precision
Precision
Epochs
0.95
0.90
0.85
0.80
0.75
2
4
6
8
10



Third experiment:(GRU)

2 GRU bidirectional + Dense + Dropout + Dense

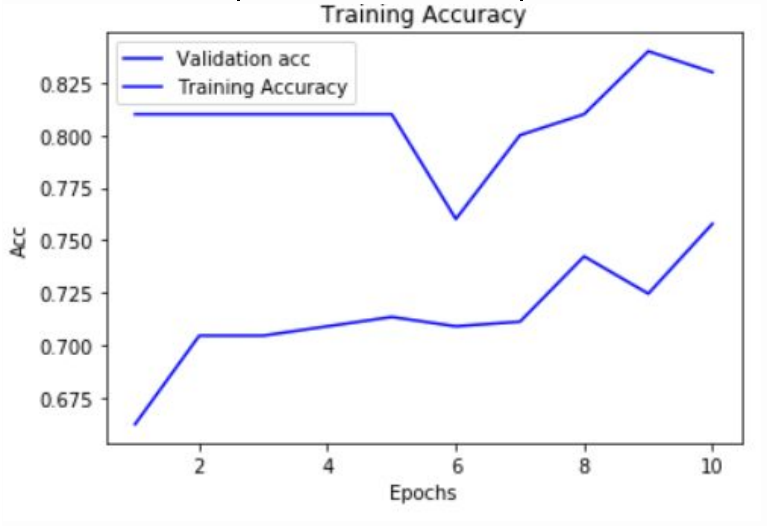
Training Accuracy	Vali Accuracy	Loss	Precision	Recall
0.84	0.74	0.14	0.93	0.90

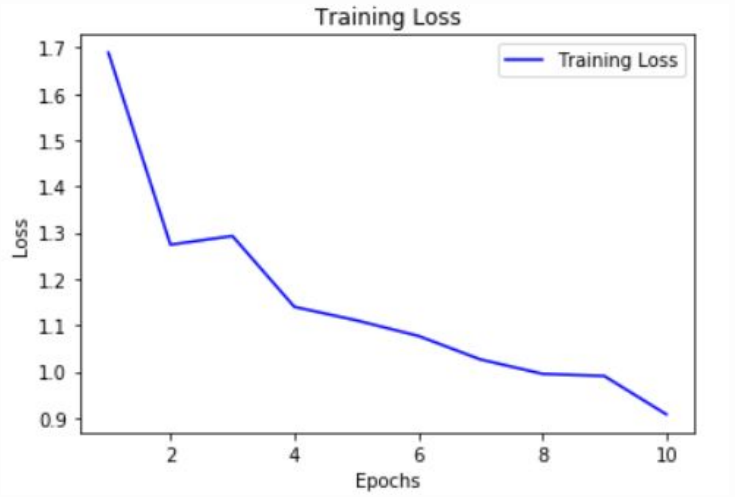
Fourth experiment:(BERT)

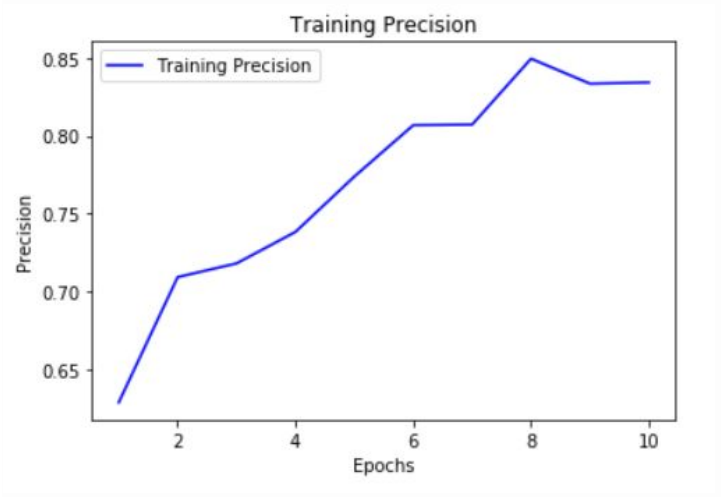
Epochs: 10

Data: 500

Training Accuracy	Vali Accuracy	Loss	Precision	Recall
0.75	0.80	0.8	0.83	0.67







We limited the data to 500, so we expect the model to behave much better when we run it on the whole dataset which contains 16000 data.

Discussion:

- **Why is this specific type of Deep Neural Networks suitable for this task?**
    1. LSTMs have memory
    2. Bidirectional LSTMs have better performance because they consider the reverse sequence.
    3. Pre-trained Models have the following benefits:
        - They achieve solid (same or even better) model performance quickly.

- We faced a lack of data and they are trained on Enough data, so they are **useful for us.**

- **What are the benefits of your proposed model against other approaches?**
  Using Transform Learning which has the below-mentioned benefits:
  1. avoiding training time
  2. better performance of neural networks (in most cases)
  3. not needing a lot of data

References:

- https://arxiv.org/abs/1810.04805
- https://towardsdatascience.com/bert-in-keras-with-tensorflow-hub-76bcbc9417b
- https://tfhub.dev/s?q=bert
- https://propaganda.qcri.org/semeval2020-task11/
- https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
- https://nlp.stanford.edu/projects/glove/
- https://www.aclweb.org/anthology/D19-5021.pdf
- https://www.cs.columbia.edu/~tariq/papers/2019NLP4IF_Propaganda_Shared_Task.pdf