Assignment 2 Report

Question 1:

How does ELMo differ from CoVe? Discuss and differentiate both the strategies used to obtain the contextualized representations with equations and illustrations as necessary.

Answer 1 :

ELMo (Embeddings from Language Models) and CoVe (Contextualized Word Vectors) both methods are used for getting contextualized representation of words.

ELMo (Embeddings from Language Models):

Architecture:

The core idea behind ELMo is to capture the nuanced meaning of words within a sentence by considering the context in which they appear. Unlike traditional word embeddings like Word2Vec or GloVe, which provide static representations for words, ELMo generates dynamic word embeddings that adapt to the specific context of each word. This adaptability is achieved through the use of deep bidirectional recurrent neural networks (RNNs), specifically bidirectional LSTMs (Long Short-Term Memory networks).

How ELMo works:

1. Deep Bidirectional LSTMs: ELMo employs a deep stack of bidirectional LSTMs to process input text. These bidirectional LSTMs can capture information from both the left and right contexts of a word in a sentence.
2. Layer Stacking: ELMo allows for the use of multiple layers of bidirectional LSTMs, each capturing different levels of contextual information. The representations from these various layers capture diverse aspects of syntactic and semantic context.

3. Task-Adaptive Combination: ELMo embeddings are not derived from a single layer or a fixed combination of layers. Instead, they are dynamically blended based on the specific context in which a word is situated. ELMo calculates weighted combinations of embeddings from different layers, and these weights are learned during the model's pretraining and fine-tuning stages.

4. Pretraining and Fine-Tuning: ELMo undergoes two crucial phases. First, it is pretrained on a vast corpus of text using a language modeling task. During this phase, the model learns to predict the next word in a sentence, requiring it to grasp syntactic structures, semantics, and general world knowledge. Following pretraining, ELMo embeddings can be fine-tuned on task-specific data to tailor them to particular NLP tasks.

The ELMo representation for a word is computed as

follows: ELMo(word) = **γ * [∑(i=0 to L-1) s_i * h_i]**

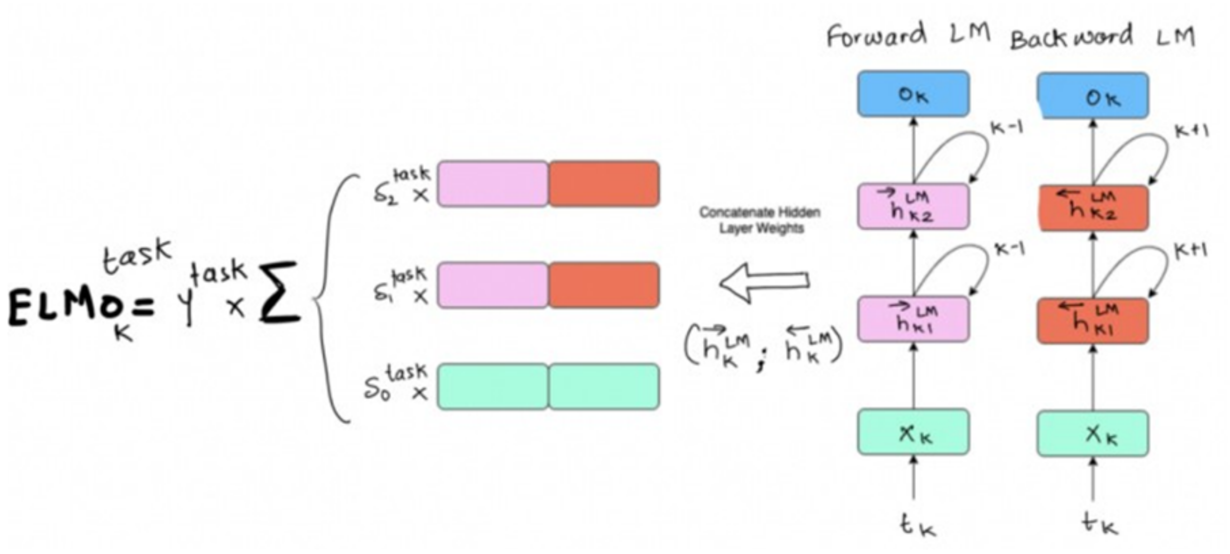
Where:

L is the number of LSTM layers.
s_i represents the scalar weights for each layer (learned during training).
h_i is the hidden state of the LSTM at layer i.

γ is a scalar parameter that scales the entire ELMo vector.

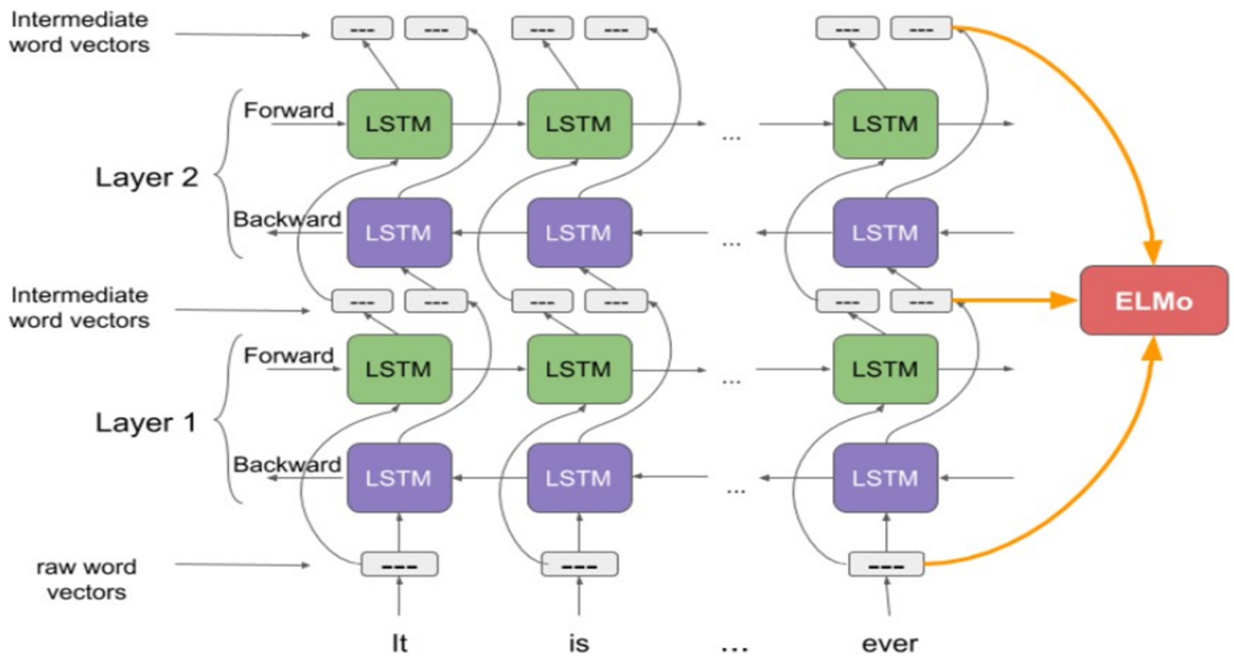
Example:

Imagine a sentence: "I love NLP Assignments."



$ELMo_k^{task} = \gamma^{task} \times \sum$
$s_2^{task} \times$
$s_1^{task} \times$
$s_0^{task} \times$
Concatenate Hidden Layer Weights
$(\overrightarrow{h}_k^{LM} ; \overleftarrow{h}_k^{LM})$
Forward LM
Backword LM
o_k
$k-1$
$k+1$
$\overrightarrow{h}_{k2}^{LM}$
$\overleftarrow{h}_{k2}^{LM}$
$\overrightarrow{h}_{k1}^{LM}$
$\overleftarrow{h}_{k1}^{LM}$
x_k
t_k

The ELMo model processes this sentence, generating contextualized representations for each word by considering both directions of the sentence.



So here we are giving glove/word2vec embeddings as input and that will be given as input to the model.

At each layer we got embeddings but the embedding will start holding more information as we go upward in the model. And the output from the forward layer will be given to the next forward layer and similarly for the backward layer to get the final embeddings we combine the output embedding from both layers at that level.

CoVe (Contextualized Word Vectors):

Architecture:

CoVe is designed as a supplementary model that utilizes the output of a machine translation model (specifically, a bidirectional LSTM-based neural machine translation system).

Here's an explanation of COVE and how it works:

Contextualization: COVE's fundamental purpose is to capture the context surrounding words within a sentence. In the realm of Natural Language Processing (NLP), the meaning of a word often hinges on the words that accompany it. COVE addresses this by providing word embeddings that are sensitive to their context.

Bidirectional LSTM: At the heart of COVE lies a bidirectional LSTM (Long Short-Term Memory) network. LSTMs are a type of recurrent neural network (RNN) renowned for their aptitude in modeling sequential data.
The bidirectional nature signifies that the LSTM processes input text both in the forward (left-to-right) and backward (right-to-left) directions. This enables it to glean information from words that precede and follow a given word in a sentence.

Word Embedding Generation: For every word in a sentence, COVE guides it through the bidirectional LSTM. In response, the LSTM generates a hidden state vector for each word.

Critically, these hidden states for each word are computed by considering the complete context of the entire sentence, thanks to the bidirectional processing. Fixed Dimensionality: In contrast to ELMo, which yields multiple embeddings per word from different layers, COVE typically furnishes a single, fixed-dimension vector representation for each word in a sentence.
This vector encapsulates the word's essence within the context of the entire sentence.

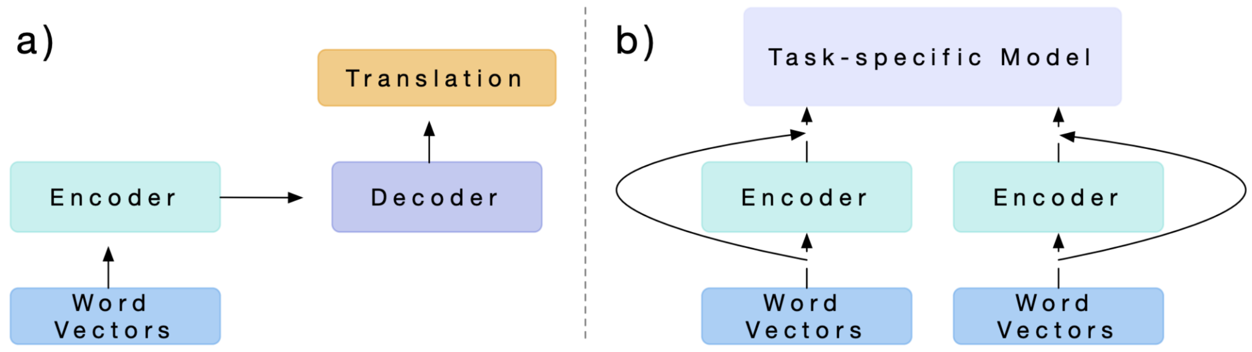
Pretraining and Fine-Tuning: COVE models typically undergo two phases. First, they are pretrained on an extensive body of text using various language modeling objectives. During this stage, the model learns to predict the next word in a sentence, a task that necessitates an understanding of syntax, semantics, and world knowledge.

Following pretraining, COVE embeddings can be fine-tuned using task-specific data, allowing them to adapt to the nuances of specific NLP tasks.
Use in Downstream Tasks: COVE embeddings serve as valuable features in a diverse array of NLP tasks, such as sentiment analysis, text classification, and named entity recognition.

Contextualized Representations:

CoVe leverages the context from a translation task to generate contextualized word vectors.

It captures contextual information by encoding a sentence both left-to-right and right-to-left in the source language and then decoding it in the target language.



Equations:

$$\text{CoVe}(w) = \text{MT-LSTM}(\text{GloVe}(w))$$

For classification and question answering, for an input sequence w, **each vector in GloVe(w) is concatenated with its corresponding vector in** CoVe(w):

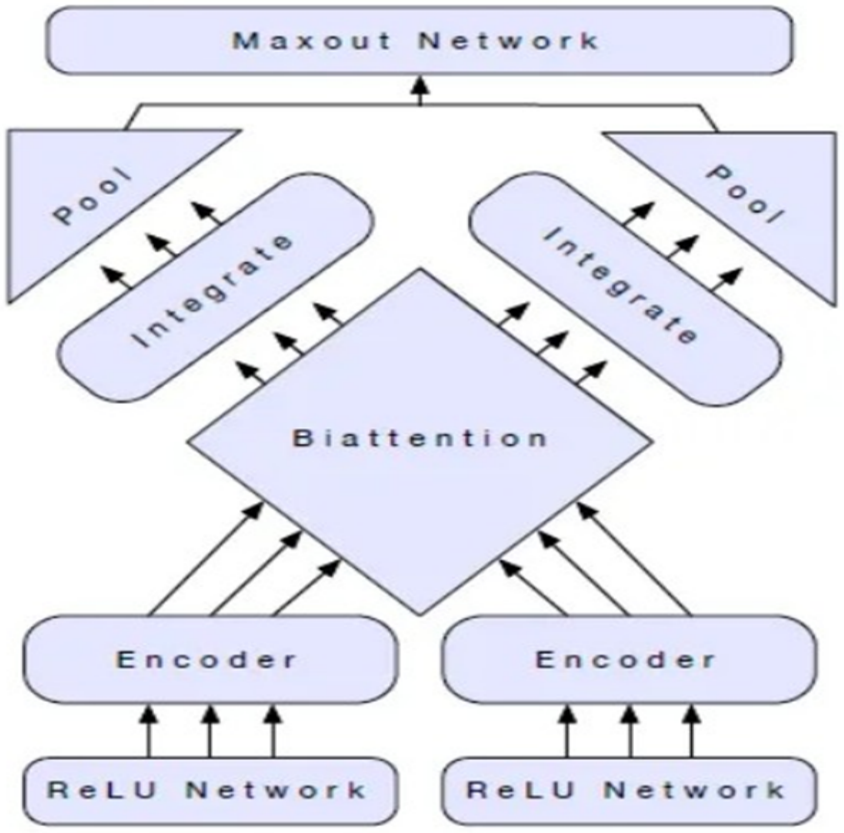
$$\tilde{w} = [\text{GloVe}(w); \text{CoVe}(w)]$$

CoVe relies on the hidden states of the encoder (bidirectional LSTM) in the machine translation model. The representation for a word can be obtained as follows:

CoVe(word) = [h_L_enc, ..., h_1_enc, h_1_dec, ..., h_L_dec]

Where:

h_L_enc to h_1_enc are the hidden states from the encoder
LSTM
h_1_dec to h_L_dec are the hidden states from the decoder
LSTM.



Biattentive classification network (BCN) (Inputs are already CoVe here)

Example:

Take, for instance, an English sentence: "I love NLP assignments." In the CoVe approach, this sentence is initially transformed into a fixed-size vector via the encoder LSTM, which captures context from both the left-to-right and right-to-left perspectives. Subsequently, this vector is employed to generate a counterpart sentence in a different language (e.g., French), all while retaining the contextual information.

Similarities:
When it comes to computing contextual word representations, CoVe and ELMo share several similarities. For instance:

- Both methods involve the creation of contextual word vectors.
- They rely on large corpora for pretraining.
- The pretrained models are leveraged to transform the acquired knowledge into word representations.
- The computed word vectors are input into task-specific networks for later use.

- During task training, both approaches make use of preset weights derived from previous training.

Key Differences:
Here are some of the primary distinctions between ELMo and CoVe:

- ELMo employs a language model, whereas CoVe utilizes a machine translation encoder to construct word representations during the pretraining phase.
- ELMo's training is based on unsupervised language model data, while CoVe's training is grounded in supervised machine translation datasets.
- In ELMo, the final representation is a weighted aggregation of all layers in the language model LSTM, whereas CoVe treats the output of the last layer of the machine translation model as the context vectors.
- ELMo adjusts the weights from the bidirectional language model using task-specific data before fixing them for task training. In contrast, CoVe vectors remain fixed during task training.
- ELMo adapts its architecture to suit specific task requirements, employing distinct topologies for each task. On the other hand, researchers employing CoVe often use a consistent architecture across various downstream classification tasks.

In summary, ELMo and CoVe share the common goal of acquiring contextualized word representations. However, they employ distinct architectural approaches and techniques for capturing contextual information. ELMo directly generates contextual embeddings, whereas CoVe relies on a machine translation model to extract contextualized word vectors.

QUESTION 2:

The architecture described in the ELMo paper includes a character convolutional layer at its base. Find out more on this, and describe this layer. Why is it used? Is there any alternative to this? [Hint: Developments in word tokenization

Answer 2 :

In the original ELMo paper, the architectural design encompasses a foundational element known as the character-level convolutional layer. This integral layer serves the purpose of extracting character-level representations for words prior to their progression through the subsequent layers of the model.

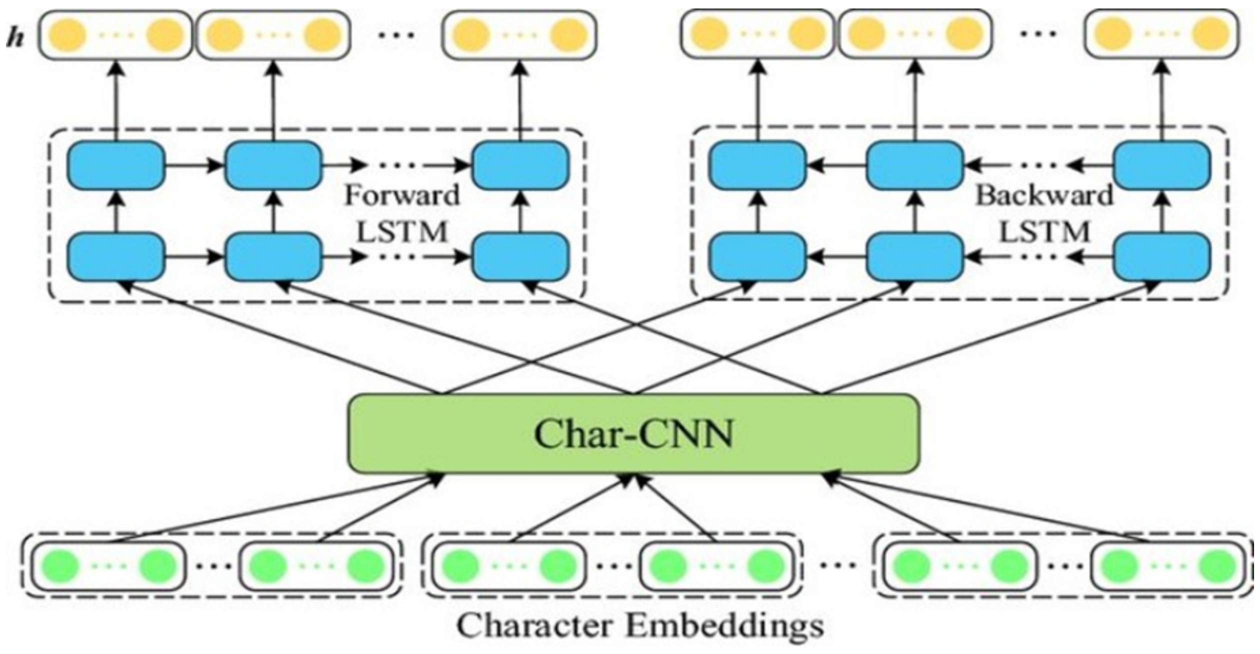
Why Character Convolutional Layer:

Purpose:

The character convolutional layer in ELMo is designed to capture subword-level information from the characters in each word. Which Gives more in depth intuition as we can also manage out of vocabulary words. Thats how ELMo helps in handling out-of-vocabulary words and capturing morphological and orthographic features of words.

Architecture:

The character convolutional layer typically comprises one or multiple convolutional neural networks (CNNs). Each CNN functions by analyzing a window of character embeddings, producing more advanced representations at the character level. These character-level representations are subsequently combined with word-level embeddings to compose the ultimate contextualized word representations.



Advantages:

- Handling Out-of-Vocabulary Words with Confidence: ELMo's character-level processing equips it to adeptly manage words absent from its vocabulary by acquiring meaningful character-level representations.
- Embracing Morphological Insights: This approach empowers ELMo to grasp information pertaining to word morphology, encompassing prefixes, suffixes, and other subword structures.
- Enhancing Semantic Representations: The inclusion of character-level data enhances the semantic depiction of words, rendering it more imbued with contextual depth.

Alternatives to Character Convolutional Layer:

While character convolutional layers excel in capturing subword information, advancements in word tokenization have introduced alternative approaches:

1) Subword Tokenization:
    - Subword tokenization methods such as Byte-Pair Encoding (BPE), SentencePiece, and WordPiece have emerged. These techniques deconstruct words into subword units.
    - These methods generate subword tokens, which are fragments of words, enabling models to represent and learn from subword details without necessitating a dedicated character convolutional layer.
    - Models like BERT and its variants, including RoBERTa and GPT-2, frequently adopt subword tokenization as a viable substitute for character-level processing.

2) WordPiece and SentencePiece Models:
    a) Google introduced these tokenization models, which possess the capability to generate subword tokens in a data-centric fashion.
    b) They construct a vocabulary consisting of subword units and perform text tokenization based on this vocabulary, efficiently capturing subword nuances without relying on character-level CNNs.

3) Hybrid Approaches:
    - Certain models, such as XLNet, adopt a hybrid strategy that incorporates both word-level and subword-level representations.

- In these approaches, character-level processing may be applied to handle out-of-vocabulary words, while subword tokenization is employed for the remainder of the words.

In summary, although character convolutional layers excel in capturing subword information and elevating word representations, recent word tokenization advancements, such as subword tokenization techniques like BPE and WordPiece, have introduced viable alternatives that offer similar benefits. The choice between character-level processing and subword tokenization can be made based on the specific task and the availability of training data.

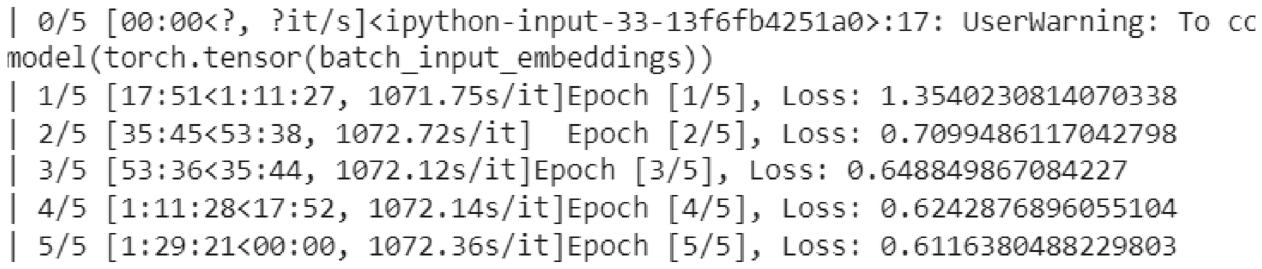
Model Analysis :

Due to computational crunch i was not able to train on much larger data thats why the i have little bit low accuracy.

Mainly i have trained three models with different permutations and combinations of learning rate, dropout, trainable parameters, optimizers, non trainable fixed parameters.

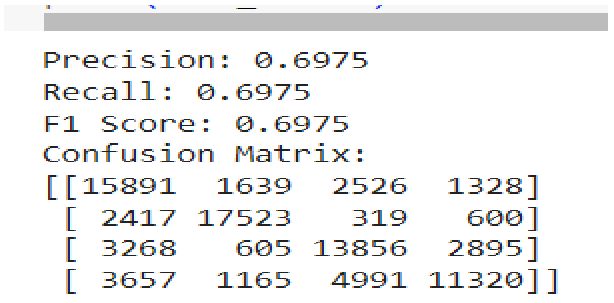
But here is the gist of all the models

1) Model with trainable parameters:
2) Here is the loss for elmo model:



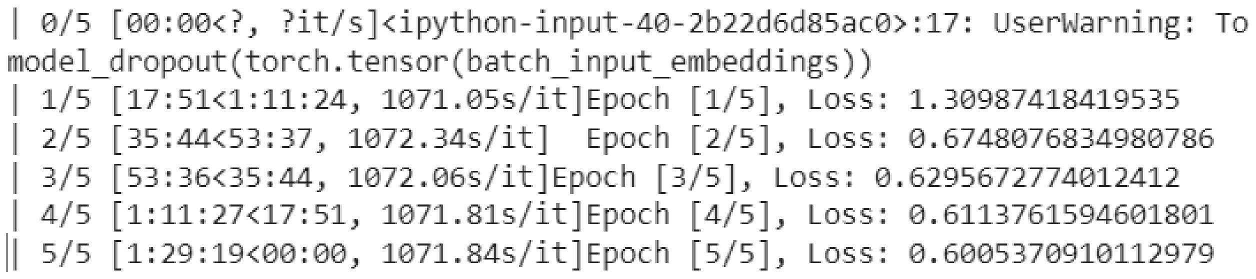
```
| 0/5 [00:00<?, ?it/s]<ipython-input-33-13f6fb4251a0>:17: UserWarning: To cc
model(torch.tensor(batch_input_embeddings))
| 1/5 [17:51<1:11:27, 1071.75s/it]Epoch [1/5], Loss: 1.3540230814070338
| 2/5 [35:45<53:38, 1072.72s/it]  Epoch [2/5], Loss: 0.7099486117042798
| 3/5 [53:36<35:44, 1072.12s/it]Epoch [3/5], Loss: 0.648849867084227
| 4/5 [1:11:28<17:52, 1072.14s/it]Epoch [4/5], Loss: 0.6242876896055104
| 5/5 [1:29:21<00:00, 1072.36s/it]Epoch [5/5], Loss: 0.6116380488229803
```

And here is the final prediction results

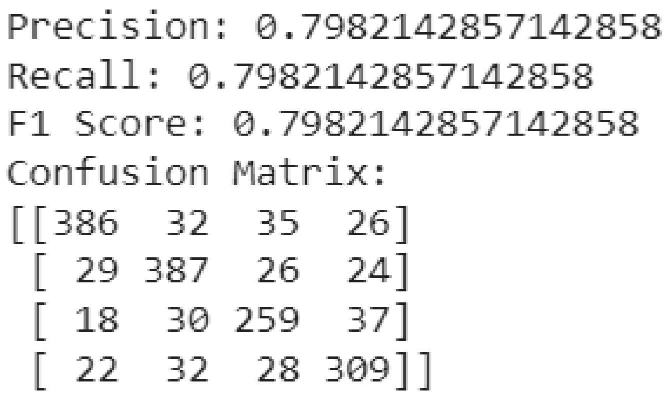


```
Precision: 0.6975
Recall: 0.6975
F1 Score: 0.6975
Confusion Matrix:
[[15891  1639  2526  1328]
 [ 2417 17523   319   600]
 [ 3268   605 13856  2895]
 [ 3657  1165  4991 11320]]
```

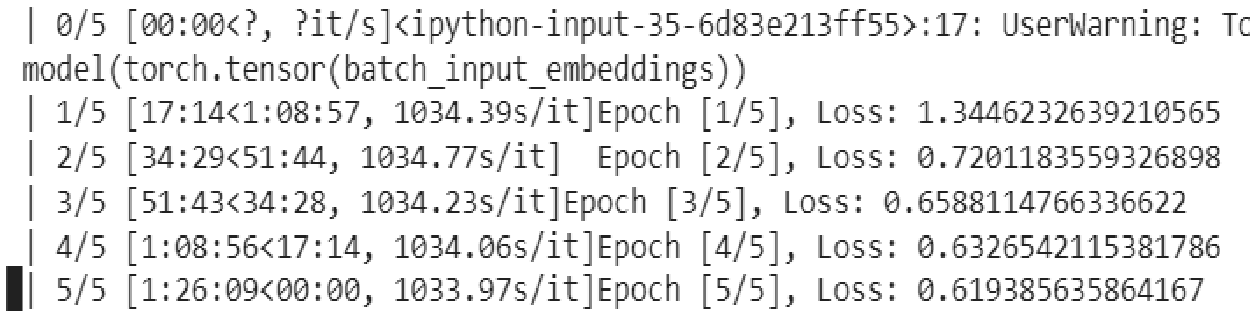
2) For model with dropout and trainable parameters:


```
| 0/5 [00:00<?, ?it/s]<ipython-input-40-2b22d6d85ac0>:17: UserWarning: To
model_dropout(torch.tensor(batch_input_embeddings))
| 1/5 [17:51<1:11:24, 1071.05s/it]Epoch [1/5], Loss: 1.30987418419535
| 2/5 [35:44<53:37, 1072.34s/it]  Epoch [2/5], Loss: 0.6748076834980786
| 3/5 [53:36<35:44, 1072.06s/it]Epoch [3/5], Loss: 0.6295672774012412
| 4/5 [1:11:27<17:51, 1071.81s/it]Epoch [4/5], Loss: 0.6113761594601801
|| 5/5 [1:29:19<00:00, 1071.84s/it]Epoch [5/5], Loss: 0.6005370910112979
```

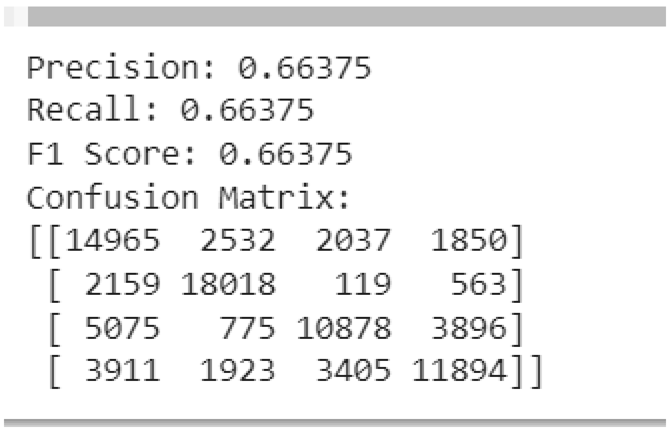
And here is the final prediction results:


```
Precision: 0.7982142857142858
Recall: 0.7982142857142858
F1 Score: 0.7982142857142858
Confusion Matrix:
[[386  32  35  26]
 [ 29 387  26  24]
 [ 18  30 259  37]
 [ 22  32  28 309]]
```

3) Model with fixed parameters:

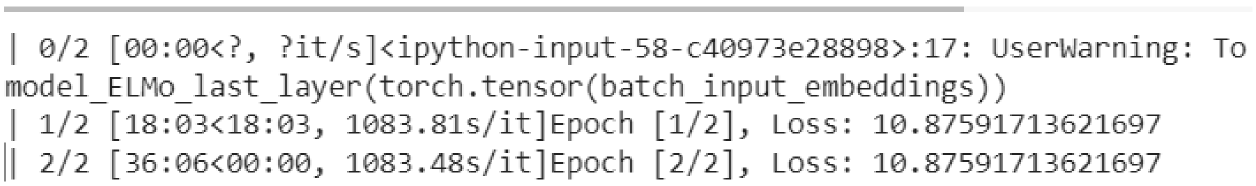

```
| 0/5 [00:00<?, ?it/s]<ipython-input-35-6d83e213ff55>:17: UserWarning: Tc
model(torch.tensor(batch_input_embeddings))
| 1/5 [17:14<1:08:57, 1034.39s/it]Epoch [1/5], Loss: 1.3446232639210565
| 2/5 [34:29<51:44, 1034.77s/it]  Epoch [2/5], Loss: 0.7201183559326898
| 3/5 [51:43<34:28, 1034.23s/it]Epoch [3/5], Loss: 0.6588114766336622
| 4/5 [1:08:56<17:14, 1034.06s/it]Epoch [4/5], Loss: 0.6326542115381786
█| 5/5 [1:26:09<00:00, 1033.97s/it]Epoch [5/5], Loss: 0.619385635864167
```

And here is the final prediction results:



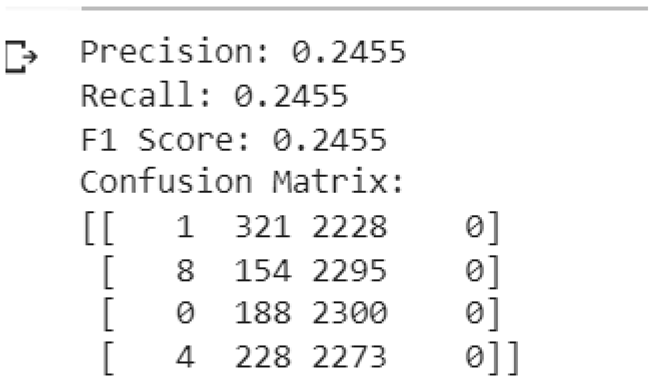
```
Precision: 0.66375
Recall: 0.66375
F1 Score: 0.66375
Confusion Matrix:
[[14965  2532  2037  1850]
 [ 2159 18018   119   563]
 [ 5075   775 10878  3896]
 [ 3911  1923  3405 11894]]
```

4) Also i trained only on the last layer input:



```
| 0/2 [00:00<?, ?it/s]<ipython-input-58-c40973e28898>:17: UserWarning: To
model_ELMo_last_layer(torch.tensor(batch_input_embeddings))
| 1/2 [18:03<18:03, 1083.81s/it]Epoch [1/2], Loss: 10.87591713621697
|| 2/2 [36:06<00:00, 1083.48s/it]Epoch [2/2], Loss: 10.87591713621697
```

And here is the final prediction results:



```
Precision: 0.2455
Recall: 0.2455
F1 Score: 0.2455
Confusion Matrix:
[[   1  321 2228    0]
 [   8  154 2295    0]
 [   0  188 2300    0]
 [   4  228 2273    0]]
```

So this model gives the worst performance

1) Due to less no of epochs
2) Due to none to less context and weightage to previous layers
3) I think the model is overfitting

But i think model should perform well enough as the last layer have all the context.

So the model with trainable parameters performs the best as it can adapt to the changing data.

And hence can give more weightage to layers with more important information.

Also the accuracy increased a lot when I added the dropout layer.

One thing I noticed is the model is giving more weightage to the upper layer.

BONUS PART:

So what i noticed is that when i am giving more weightage to upper layer embedding the output is better.