Report

Experiments and Stats

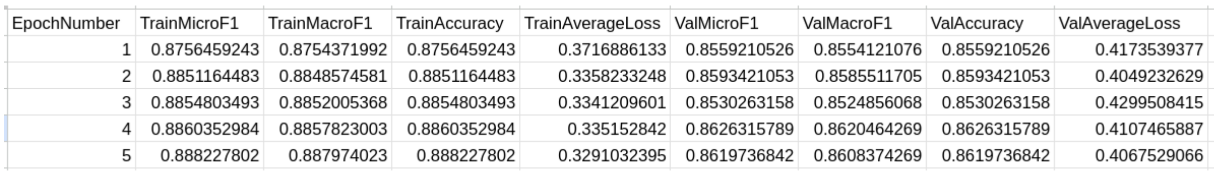
ELMo Pre-Training

- Used cross-entropy loss
- Used Adam optimizer and a learning rate of 10^-3
- Sequence Length on which the LSTMs were trained was 50 and batch size=64
- Validation Loss continuously decreases till epoch 10
- **Test Stats: Forward Perplexity: 23.40459632873535, Backward Perplexity: 24.30259132385254**

EpochNumber	ForwardTrainAveragePerplexity	BackwardTrainAveragePerplexity	ForwardValAveragePerplexity	BackwardValAveragePerple
1	104.12401580810547	106.81610870361328	61.385704040527344	64.31090545654297
2	46.669586181640625	48.642059326171875	39.40065002441406	40.62327575683594
3	33.39577102661133	34.39095687866211	31.952436447143555	32.797794342041016
4	27.273767471313477	27.902385711669922	28.26253318786621	29.05089569091797
5	23.570344924926758	24.139375686645508	26.202707290649414	27.024486541748047
6	21.075498580932617	21.606294631958008	24.958818435668945	25.838987350463867
7	19.24132537841797	19.78057098388672	24.225858688354492	25.18358612060547
8	17.87572479248047	18.39749526977539	23.810256958007812	24.763389587402344
9	16.80603790283203	17.30869483947754	23.56234359741211	24.544904708862305
10	15.937618255615234	16.415271759033203	23.453529357910156	24.414588928222656

Downstream Classifier

- Used cross-entropy loss
- Used Adam optimizer and a learning rate of 10^-3
- Used an LSTM which outputs **X** and a Linear layer which uses **X** to predict the type of news
- **When X = Hidden state**
  - Used the final hidden state of the LSTM as **X**
  - **Test Stats**
    - Average Loss 0.3374302024100007
    - Accuracy 0.8836842105263157
    - Micro_F1 0.8836842105263157
    - Macro_F1 0.8832700582486861



EpochNumber	TrainMicroF1	TrainMacroF1	TrainAccuracy	TrainAverageLoss	ValMicroF1	ValMacroF1	ValAccuracy	ValAverageLoss
1	0.8756459243	0.8754371992	0.8756459243	0.3716886133	0.8559210526	0.8554121076	0.8559210526	0.4173539377
2	0.8851164483	0.8848574581	0.8851164483	0.3358233248	0.8593421053	0.8585511705	0.8593421053	0.4049232629
3	0.8854803493	0.8852005368	0.8854803493	0.3341209601	0.8530263158	0.8524856068	0.8530263158	0.4299508415
4	0.8860352984	0.8857823003	0.8860352984	0.335152842	0.8626315789	0.8620464269	0.8626315789	0.4107465887
5	0.888227802	0.887974023	0.888227802	0.3291032395	0.8619736842	0.8608374269	0.8619736842	0.4067529066

- **When X = Cell state**
  - Used the final cell state of the LSTM as **X ,** since it captures more broad history of the sentence
  - **Using a Linear layer to weigh the concatenated vectors of each layer of Bi-LSTM.** Maybe would learn a more complex relationship?

- **Test Stats**:
  - Average Loss 0.2867569106967509
  - Accuracy 0.9003947368421052
  - Micro_F1 0.9003947368421052
  - Macro_F1 0.9002802664369609

EpochNumber	TrainMicroF1	TrainMacroF1	TrainAccuracy	TrainAverageLoss	ValMicroF1	ValMacroF1	ValAccuracy	ValAverageLoss
1	0.8829785298	0.882725139	0.8829785298	0.3389793304	0.8601315789	0.86037738	0.8601315789	0.4130168222
2	0.8956604803	0.8954249178	0.8956604803	0.3014798389	0.8703947368	0.8700267299	0.8703947368	0.3729291702
3	0.8988628093	0.8986398781	0.8988628093	0.2943134828	0.8671052632	0.8670496682	0.8671052632	0.3790539156
4	0.9000181951	0.8997971349	0.9000181951	0.2891621889	0.8725	0.8717663635	0.8725	0.3616793189
5	0.9007459971	0.9005341398	0.9007459971	0.2868854726	0.8805263158	0.8801626646	0.8805263158	0.3521155836

- **Using two scalar trainable weights for weighted sum**

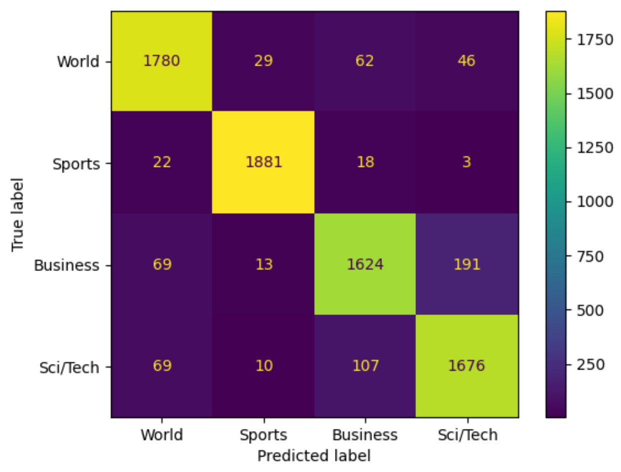
  Test Stats:
  - Micro_F1 0.9147368421052632
  - Accuracy 0.9147368421052632
  - Macro_F1 0.9142990650635662
  - Average Loss 0.2382249490929251

EpochNumber	TrainMicroF1	TrainMacroF1	TrainAccuracy	TrainAverageLoss	ValMicroF1	ValMacroF1	ValAccuracy	ValAverageLoss
1	0.8917485444	0.8914181872	0.8917485444	0.314531161	0.8792105263	0.8788738467	0.8792105263	0.3473240309
2	0.9050855167	0.9047711733	0.9050855167	0.2706379294	0.8898684211	0.8895713838	0.8898684211	0.3270527553
3	0.9122179767	0.9119444453	0.9122179767	0.2496553185	0.8980263158	0.8976150306	0.8980263158	0.3070553699
4	0.917330786	0.9170900055	0.917330786	0.2346295095	0.8994736842	0.8990461001	0.8994736842	0.2943821555
5	0.9219341339	0.9217273637	0.9219341339	0.2215684583	0.9011842105	0.9004606251	0.9011842105	0.2913947105

We can see that using the cell state performed better than hidden state. And using a weighted sum performed better than letting the Linear layer take care of analyzing the layer embeddings.

Hence the best possible configuration was using cell state and scalar trainable weights(Highlighted in Red above)

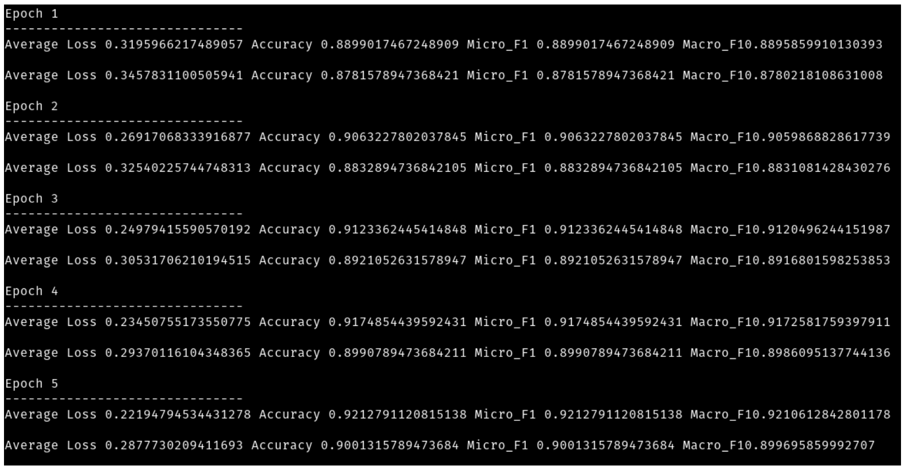
Confusion Matrix:



Bonus

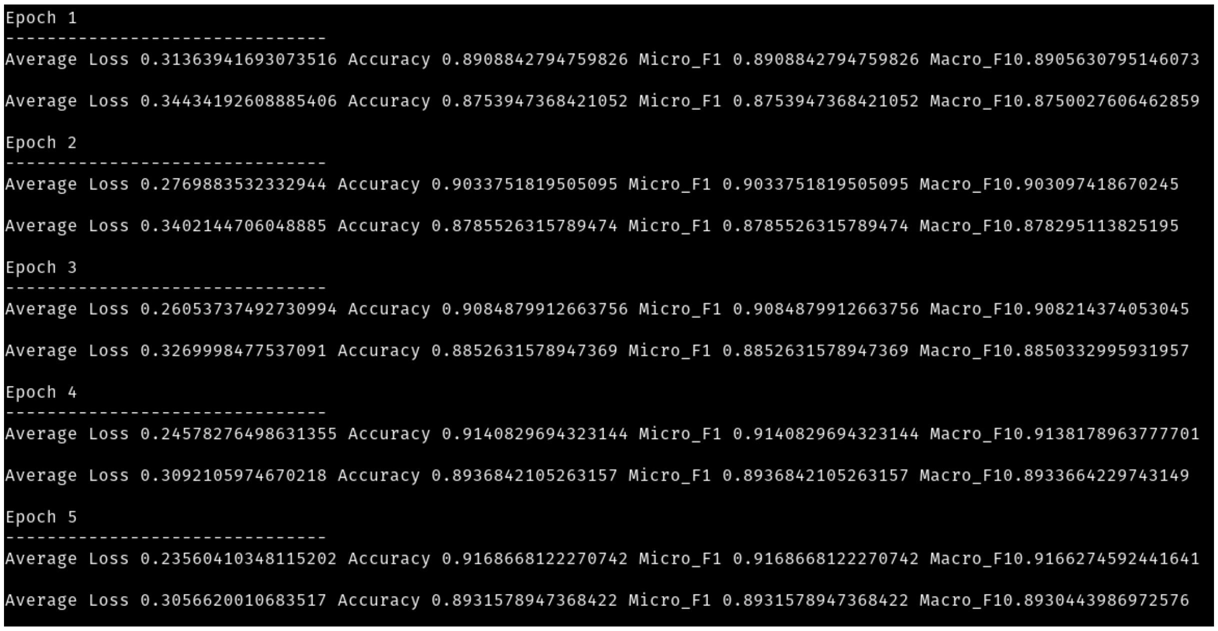
Hardcoding the Weights

- **1 - for lower layer ; 0 - for upper layer**



```
Epoch 1
-------------------------------
Average Loss 0.3195966217489057 Accuracy 0.8899017467248909 Micro_F1 0.8899017467248909 Macro_F10.8895859910130393

Average Loss 0.3457831100505941 Accuracy 0.8781578947368421 Micro_F1 0.8781578947368421 Macro_F10.8780218108631008

Epoch 2
-------------------------------
Average Loss 0.26917068333916877 Accuracy 0.9063227802037845 Micro_F1 0.9063227802037845 Macro_F10.9059868828617739

Average Loss 0.32540225744748313 Accuracy 0.8832894736842105 Micro_F1 0.8832894736842105 Macro_F10.8831081428430276

Epoch 3
-------------------------------
Average Loss 0.24979415590570192 Accuracy 0.9123362445414848 Micro_F1 0.9123362445414848 Macro_F10.9120496244151987

Average Loss 0.30531706210194515 Accuracy 0.8921052631578947 Micro_F1 0.8921052631578947 Macro_F10.8916801598253853

Epoch 4
-------------------------------
Average Loss 0.23450755173550775 Accuracy 0.9174854439592431 Micro_F1 0.9174854439592431 Macro_F10.9172581759397911

Average Loss 0.29370116104348365 Accuracy 0.8990789473684211 Micro_F1 0.8990789473684211 Macro_F10.8986095137744136

Epoch 5
-------------------------------
Average Loss 0.22194794534431278 Accuracy 0.9212791120815138 Micro_F1 0.9212791120815138 Macro_F10.9210612842801178

Average Loss 0.2877730209411693 Accuracy 0.9001315789473684 Micro_F1 0.9001315789473684 Macro_F10.899695859992707
```

- **0 - for lower layer ; 1 - for upper layer**

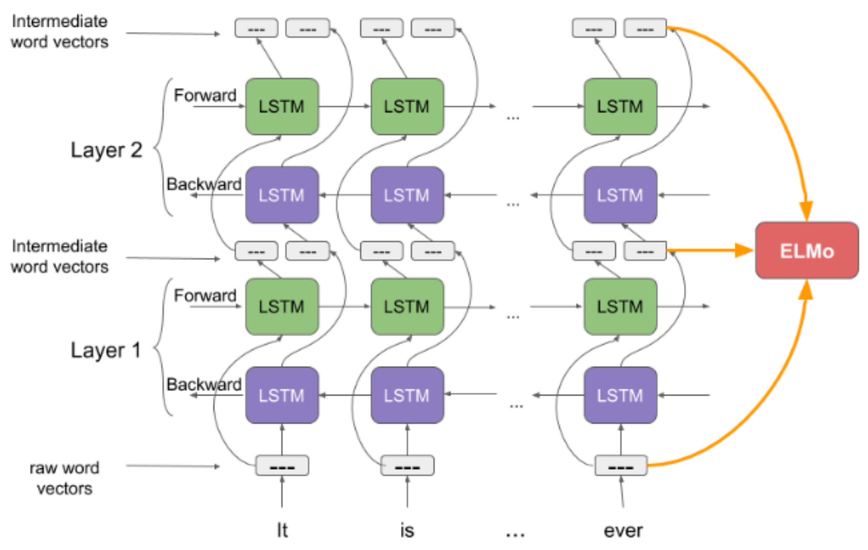


```
Epoch 1
-------------------------------
Average Loss 0.31363941693073516 Accuracy 0.8908842794759826 Micro_F1 0.8908842794759826 Macro_F10.8905630795146073

Average Loss 0.34434192608885406 Accuracy 0.8753947368421052 Micro_F1 0.8753947368421052 Macro_F10.8750027606462859

Epoch 2
-------------------------------
Average Loss 0.2769883532332944 Accuracy 0.9033751819505095 Micro_F1 0.9033751819505095 Macro_F10.903097418670245

Average Loss 0.3402144706048885 Accuracy 0.8785526315789474 Micro_F1 0.8785526315789474 Macro_F10.878295113825195

Epoch 3
-------------------------------
Average Loss 0.26053737492730994 Accuracy 0.9084879912663756 Micro_F1 0.9084879912663756 Macro_F10.908214374053045

Average Loss 0.3269998477537091 Accuracy 0.8852631578947369 Micro_F1 0.8852631578947369 Macro_F10.8850332995931957

Epoch 4
-------------------------------
Average Loss 0.24578276498631355 Accuracy 0.9140829694323144 Micro_F1 0.9140829694323144 Macro_F10.9138178963777701

Average Loss 0.3092105974670218 Accuracy 0.8936842105263157 Micro_F1 0.8936842105263157 Macro_F10.8933664229743149

Epoch 5
-------------------------------
Average Loss 0.23560410348115202 Accuracy 0.9168668122270742 Micro_F1 0.9168668122270742 Macro_F10.9166274592441641

Average Loss 0.3056620010683517 Accuracy 0.8931578947368422 Micro_F1 0.8931578947368422 Macro_F10.8930443986972576
```

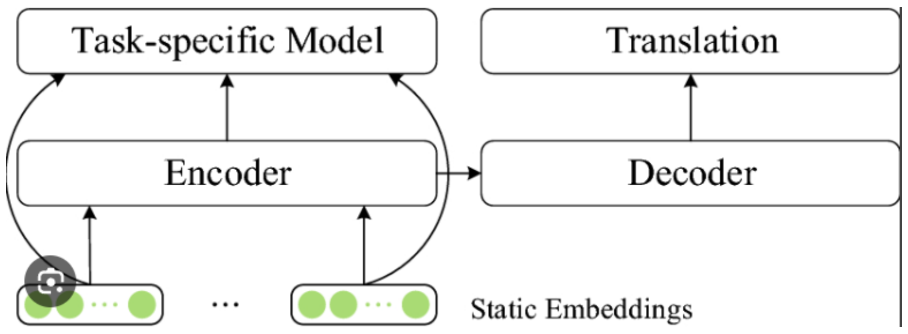
ELMo vs CoVe

ELMo and CoVe are models which produce contextualized word embeddings for words; by contextualized we mean the embedding for the word represents what the word means in that particular context/sentence.

ELMo achieves this by training a Bi-Directional LSTM with multiple layers on a corpus of token sequences. Hence the forward and backward LSTM are trained on a Language modelling task, with the forward LSTM learning how to model language from left to right and and the backward LSTM from right to left. We take the hidden states of these LSTMs , concatenate them at every layer and finally produce a single embedding for a word using a weighted sum of these layers.



CoVe on the other hand uses a Machine Translation model to get contextualized word embeddings. The Machine Translation model consists of an Encoder and Decoder. So in say a English to French translation task, the contextualized embeddings are computed from using the encoder hidden states, since the encoder essentially encodes the meaning of the English sentence, we are hoping the hidden states of the encoder give contextualized represntations of the words.



One of the key differences between CoVe and ELMo are that they are trained with different objectives in mind. ELMo is trained on a Language Modelling task, and hence the word embeddings produced by it perform better on a wide variety of downstream tasks or general tasks.

Also for pre-training for ELMo is easier than CoVe since it is self-supervised for ELMo and supervised for CoVe. They are flexible and can also be modified to be used in various specific downstream tasks. CoVe on the other hand is trained on a Machine translation task and hence the embeddings are more suited for translation related downstream tasks.

Character Convolutional Layer in ELMo

The character convolutional layer in ELMo is used to generate initial word embeddings for the model to work with. It is important since it captures subword level information and morphological features.

The character convolutional layer in ELMo processes the characters of each word in a sentence. Here's how it works:

1. **Input:** For each word in a sentence, the character-level input is a sequence of one-hot encoded characters or embeddings of characters.
2. **Convolutional Filters:** The layer applies convolutional filters to the character embeddings.
3. **Pooling:** After the convolutional operation, a max-pooling operation is often applied to reduce the dimensionality of the output. This pooling operation selects the maximum value from each feature map, resulting in a fixed-length representation for each word.
4. **Concatenation:** The fixed-length character-level representations for each word are then concatenated with the word's standard word embedding.

The advantages of using a character convolutional layer are -

Since we are dealing with the words at a character level this method of generating initial embeddings is immune to unknown words and rare words.

It can also tackle mispelled words. It will try to produce an embedding for a word based on the characters, and hence a few mispelled characters wont make a difference.

It can capture subword information that isnt available in traditional word embeddings. This helps in handling morphological variations.

Alternatives to a character level convolutional layer can be better representations for subwords

Subword Tokenization: Methods like Byte Pair Encoding - splits words into subword units based on their frequency in a corpus. It effectively captures subword information without explicit character-level modeling. This combined with subword embeddings like FastText can give good performance.

Mix: Using a combination of character level embeddings and word embeddings with subword tokenization to be more robust.

Modern tokenization strategies are being used more than character level modelling since it can be used in a variety of languages and subword phenomena.