ANLP- Assignment -2

Advaith Malladi - 2021114005

Theory part

Question-1 :-

Sol

we know ELMO,

it is used to get contextual word embeddings

How ELMO works:

we ~~Run~~ train a stacked Bi-LSTM

→ a forward stacked LSTM

→ a backward stacked LSTM

→ trained on forward-language modelling

→ trained on backward language modelling

→ for a word w

let h_{fiw} be the forward hidden state at the ith layer for the w word

let h_{biw} be the backward hidden state at the ith layer for the w word

let h_{iw} be the hidden state at ith layer layer

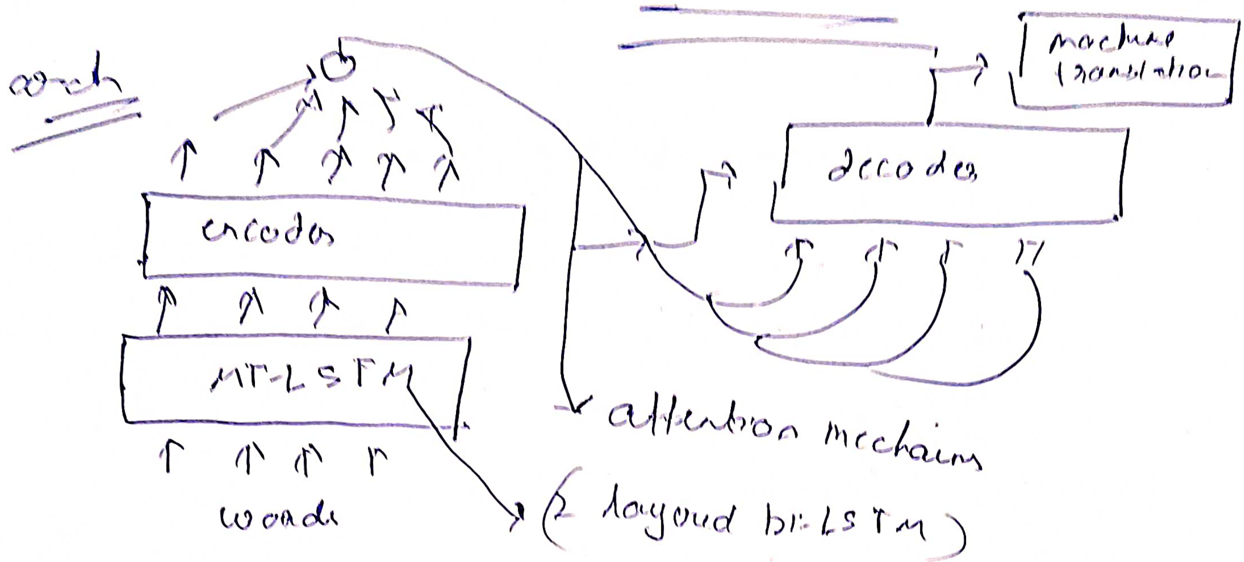
$$h_{iw} = \underbrace{[h_{fiw}, h_{biw}]}_{\text{concatenation}}$$

$$\text{Embedding } E[w] = \sum_{\forall i} \lambda_i h_{iw}$$

these λ_i's are learned for downstream tasks

Thus Elmo gives contextual word embedding for. by obtaining hidden state at multiple (all) hidden layers of the stacked-biLSTM and scaling and adding them using weights learned for a downstream task

COVE :- Contextualized word vectors.



COVE are also pretrained contextualized vectors learned from machine translation. done using encoder. attention based decoder + MT-LSTM

2 layered bi-LSTM

→ words are fed to MT-LSTM

then fed to encoder → the decoder takes 2 inputs

previous output/hidden state

and attention scaled sum of all encoding representation of input

→ The task is Machine translation

-> The M·T-LSTM is frozen and used after pre training

-> say we have a word w, MT-LSTM(w) = E

-> thus the COVE force

-> These COVE embeddings can be used by passing word to MT-LSTM

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

and Glove

$$E_G = \text{Glove}(w)$$

$$E = [E_C, E_G]$$ -> concatenation

-> this gives best Result

Basically training task for ELMo is forward and Backward language modelling

-> COVE is trained on Machine Translation

Question 2

Answer:

The character convolutional layer is another important piece of the ELMO architecture. It was implemented to generate character level embeddings. These character level embeddings can be used to enhance certain linguistic properties such as POS tag and morphological intricacies which are largely character dependent. This will also learn how to handle unknown words as we will have character level embeddings.

- The model gets character level input and has an embedding layer for the characters.
- Convolutional Filters: The layer applies convolutional filters to these character embeddings. Convolutional filters are small matrices that slide over the character embeddings, capturing local patterns within the characters
- After applying the filters, a max-pooling operation is often used. Max-pooling selects the maximum value from each filtered region. This operation helps in identifying the most salient character-level features

Alternatives for character convolutional layer:

As for alternatives in word tokenization is the use of subword tokenization methods, such as Byte-Pair Encoding (BPE) and WordPiece

Analysis and Report
Advaith Malladi
2021114005

EMLO Pretaining:

- I trained two LSTM's in the forward direction
- I trained two LSTM's in the backword direction

Architecture of a stacked lstm in a single direction:

- number of stacks: 2
- output dimension of embedding layer: 300
- input dimension for lstm1: 300
- output/hidden dimension for lstm1: 300
- input dimension for lstm1: 300
- output/hidden dimension: 300
- final linear layer input dimension: 300
- final linear layer output dimension: vocab_size

The same architecture is followed for the backward lstm also.

Performance:

- Forward Loss: 4.805687427520752,
- Backward Loss: 4.807983875274658,
- Forward Perplexity: 122.20347595214844,
- Backward Perplexity: 122.48442840576172

Downstream Task:

- My concatenated word hidden states from pre-trained stacked bi-LSTM had a dimension of 1800
- I passed this through a linear a layer which gave a final embedding dimension of 600.
- I used a linear layer because this can learn any linear function necessary
- Then, I got word embedding for all the words in my sentence into an LSTM.
- Then, I took the last hidden state and passed into a linear layer for the final classification

Model Architecture:

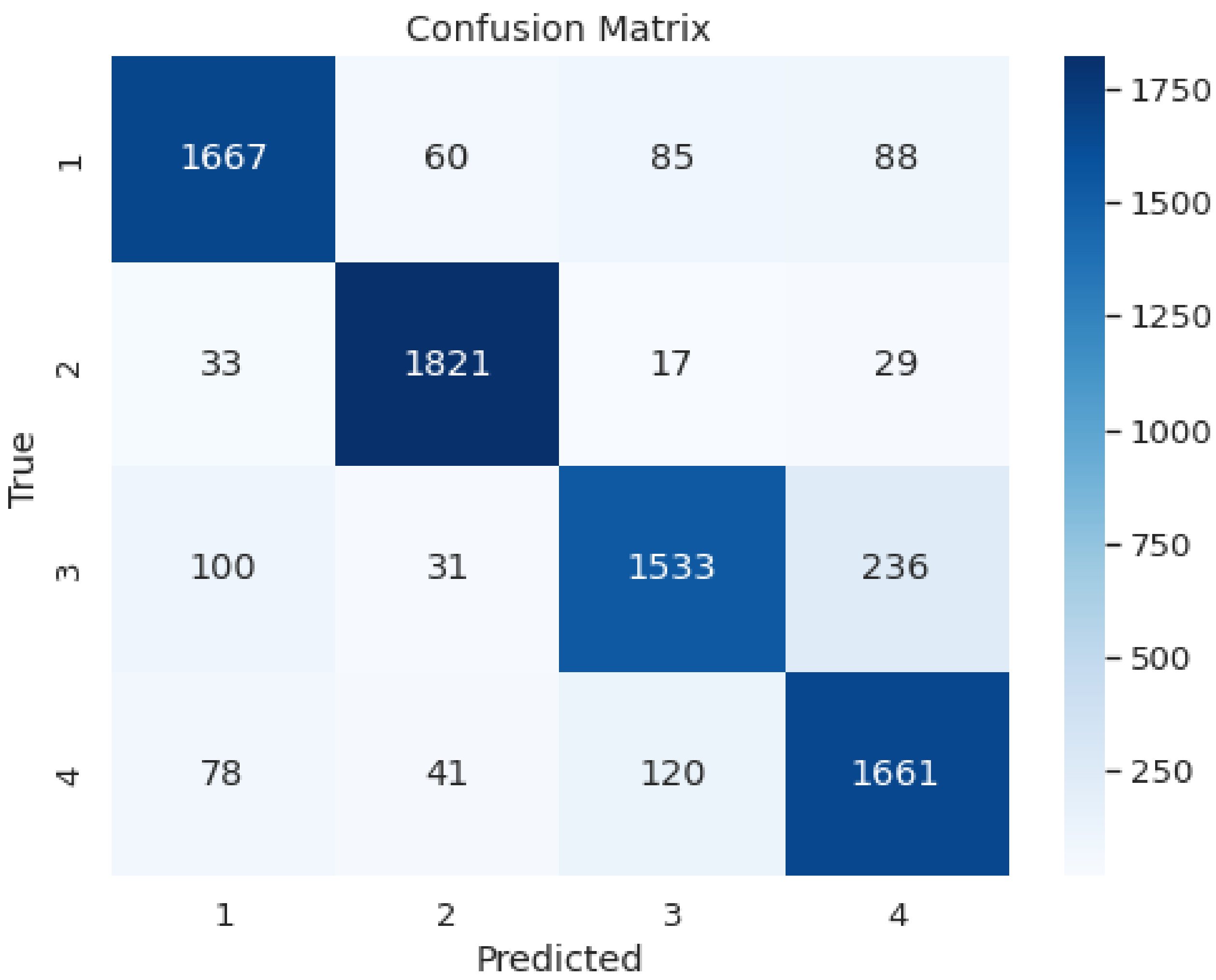
- linear layer with input dim = 1800 and output dim = 600
- lstm with input dim = 600 and output dim = 300
- linear layer with input dim = 300, output dim = number of classes

Performance on next page

Performance on downstream classification task:

- Train accuracy: 0.98
- Train micro-f1: 0.98
- Val accuracy: 0.89
- Val micro-f1: 0.89
- Test accuracy: 0.87
- Test micro-f1: 0.87

Confusion matrix for test set:



continued on next page:

Confusion matrix on the entire training process: