E0 250 Deep Learning : Project 3

HIMANSHU KUMAR (16762)
Mtech(AI)
himanshuks@iisc.ac.in

Tools

- Tensorflow 2.1.0
- Python3 3.6.9
- Keras 2.3.1
- *Keras is using tensorflow as backend.*
- *Iam using library jsonlines for reading jsonl file.*
- Train files are in .ipynb format.

1. **HOW TO RUN(main.py) :**

   (i) cmd : python main.py
   As no specific format was provided for output. My output files contain a single output for each sentence pair present in test set.
   tfidf.txt : Logistic regression output
   deep_model.txt : LSTM model output

   (ii) main.py file installs *jsonlines* library, and downloads nltk stopwords.

   (iii) load_data()
   Loads the data from test file data into sentence pairs, and test label. Also, does preprocessing on text using text_clean() function.

   (iv) logistic()
   This function runs the logistic regression model on test file. It uses the saved model, and saved TF-IDF feature set on training data using library pickle.

   (v) lstm_model()
   This function runs LSTM model on test data. It uses the saved LSTM model, and saved tokens(keras Tokenizer) on training data.

2. **STEPS LSTM MODEL :**

   (i) First step in data preprocessing, is importing data from jsonl file(jsonline file). Using jsonfiles library, function jsonlines.open imports the data.

   (ii) function text_clean()
   This function does the preprocessing on the sentences.
   Firstly, the words are converted into lowercase.
   Removing stopwords using nltk library.
   Stemming is done on each word(no stopwords) in every sentence.
   Removing special characters and punctuation marks from sentence.

(iii) The maximum length of any sentence after preprocessing is 241 words, and total sentence pairs are 550152. 785 sentence pair do not belong to any class. Now, the sentences after preprocessing are converted into tokens using keras.preprocessing.text.Tokenizer() keeping number of words upto 180000.

(iv) For data split
Iam generating a random permutation of length of sentence pairs.For data split of 10%, the accuracy was very lower. Hence,I splitted this random permutation into 93% training data and 7% validation data, for both sentence 1 and 2.

(v) Iam using standford Glove dictionary for words to vectors representation. This dictionary contains pre-trained word vectors. I have used the Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab) *glove.6B.zip*, particularly *glove.6B.100d.txt* and *glove.6B.300d.txt* file. For the final model, I have used 300d vectors. After loading the words in a dictionary, an embedding matrix is created for each token present in preprocessed dataset.

(vi) Next step is to prepare data for training, and later trained my model on prepared data.

3. **LSTM**

- Train Accuracy : 77.26 %
- Test Accuracy : 74.35 %

Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	(None, 35)	0	
input_8 (InputLayer)	(None, 35)	0	
embedding_4 (Embedding)	(None, 35, 300)	6561300	input_7[0][0] input_8[0][0]
lstm_4 (LSTM)	(None, 312)	765024	embedding_4[0][0] embedding_4[1][0]
concatenate_4 (Concatenate)	(None, 624)	0	lstm_4[0][0] lstm_4[1][0]
dropout_7 (Dropout)	(None, 624)	0	concatenate_4[0][0]
batch_normalization_7 (BatchNor	(None, 624)	2496	dropout_7[0][0]
dense_7 (Dense)	(None, 312)	195000	batch_normalization_7[0][0]
dropout_8 (Dropout)	(None, 312)	0	dense_7[0][0]
batch_normalization_8 (BatchNor	(None, 312)	1248	dropout_8[0][0]
dense_8 (Dense)	(None, 4)	1252	batch_normalization_8[0][0]

Total params: 7,526,320
Trainable params: 963,148
Non-trainable params: 6,563,172

None

Figure 1: Final Model Summary

- I have considered 2 LSTM models.
- Final model
  This model has two input layers to the embedding layer. Keras EMBEDDING layer uses pre trained word embeddings(Glove embeddings).
- LSTM layer has 312 units, I kept the units size greater than the word embeddings vector size.
- In the next layer, inputs are concatenated, and passed on to Dropout layer which prevents overfitting. In next layer, Batch Normalization normalizes the activation, for each batch.
- Next, I have a dense layer with 312 units, and activation relu.
- While it is recommended, to use tanh activation with a LSTM. But, I have achieved higher accuracy with relu. The higher accuracy can be attributed to very less, or no vanishing gradient.
- A dropout and batch normalization layer again follows, after Dense layer, for faster convergence. Finally, added the output dense layer with 4 classes.
- 60 epochs, batch=1024, optimizer = adam
  As the no of epochs is high, so I have used early stopping, on validation loss. This callback halts my model on 29th epoch, as there was no significant drop in validation loss. Adam performed slightly better than RMSprop.

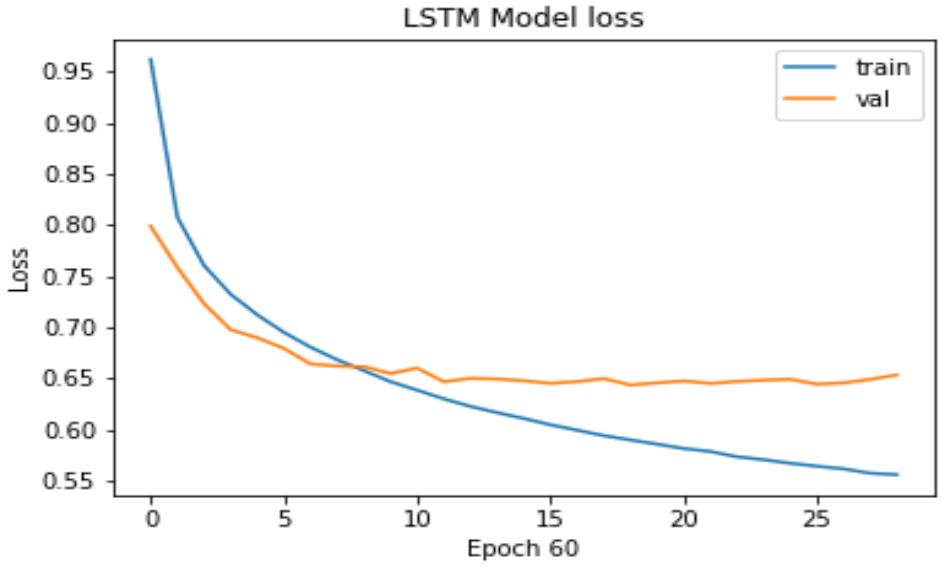

Figure 2: Training and validation loss vs Epochs

- Rejected model
  This LSTM model had 512 units, and gave me higher accuracy of 81% on train data, but gave me slightly less accuracy on test data, as compared to final model. This shows rejected model was overfitting on the training data. Even, the validation loss was increasing. This is because, I stacked extra layers, a Dropout, Batch Normalization, Dense layer(512 units) after the first dense layer(512 units in rejected model) in my final model, rest layers were same.

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_5 (InputLayer)         (None, 35)                0
_________________________________________________________________________________________________
input_6 (InputLayer)         (None, 35)                0
_________________________________________________________________________________________________
embedding_3 (Embedding)      (None, 35, 300)           6561300     input_5[0][0]
                                                                   input_6[0][0]
_________________________________________________________________________________________________
lstm_3 (LSTM)                (None, 512)               1665024     embedding_3[0][0]
                                                                   embedding_3[1][0]
_________________________________________________________________________________________________
concatenate_3 (Concatenate)  (None, 1024)              0           lstm_3[0][0]
                                                                   lstm_3[1][0]
_________________________________________________________________________________________________
dropout_6 (Dropout)          (None, 1024)              0           concatenate_3[0][0]
_________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 1024)           4096        dropout_6[0][0]
_________________________________________________________________________________________________
dense_6 (Dense)              (None, 512)               524800      batch_normalization_6[0][0]
_________________________________________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0           dense_6[0][0]
_________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 512)            2048        dropout_7[0][0]
_________________________________________________________________________________________________
dense_7 (Dense)              (None, 512)               262656      batch_normalization_7[0][0]
_________________________________________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0           dense_7[0][0]
_________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 512)            2048        dropout_8[0][0]
_________________________________________________________________________________________________
dense_8 (Dense)              (None, 4)                 2052        batch_normalization_8[0][0]
=================================================================================================
Total params: 9,024,024
Trainable params: 2,458,628
Non-trainable params: 6,565,396
```

Figure 3: Rejected Model Summary

4. **Logistic Regression**

- Train Accuracy : 65.70 %
- Test Accuracy : 63.17 %
- proj3_logistic.ipynb
- I haven't split the training set of 550152 sentences into train, and validation set. As my program computes TF-IDF score on whole training set, and these scores are later used in main.py file for test set.
- The sentences have been preprocessed with text_clean() function.
- Then the sentences have been added to a list xtext with tokens from sentence 1 preceded by 's1_' and from sentence 2 by 's2_'. This way model had greater accuracy.
- TF-IDF, 31370 features
  TfidfVectorizer() counts words, finds IDF values, then computes the TF-IDF score on the training set.
- Model
  Regularization : L2, as it the scaled sum of square of weights.
  Solver : newton-cg, as it handles multinomial loss, and gave highest accuracy.
  C : 5, regularization strength.
  max_iter : 1000, no of iteration.
  random_state : 0
  rest parameters are default.
- Classification report

Labels	precision	recall	f1-score	support
contradiction	0.66	0.63	0.65	183187
neutral	0.68	0.62	0.65	182764
entailment	0.64	0.73	0.68	183416
No class	0.00	0.00	0.00	785
weighted avg	0.66	0.66	0.66	550152

  The weighted average of F1-score is 0.66, which takes precision and recall in computation. As the no of samples from each class, are almost equally balanced, F1-score are almost equal for the three classes.

- Confusion matrix

Actual/Predicted	contradiction	neutral	entailment	No class
contradiction	115406	29651	38130	0
neutral	31994	112692	38078	0
entailment	26402	23628	133386	0
No class	197	261	327	0

  The classifier performs well for entailment class, sentence pairs.