CSA 250:Deep Learning
Project III

Bazil Ahmed

Dept:EE, Srno:16677
bazilahmed@iisc.ac.in

May 16, 2020

1. **Files and Directories Description:**
   (a) *models:* saved models in appropriate format along with their respective vectors
   (b) *outputDocs:* contains output file "tfidf.txt" and "deep_model.txt" having the prediction of the respective models on the same test set.
   (c) *main.py:* executing this file with "python3 main.py" will output two files in the outputDocs containing the predictions of both the models deep and logistic on tfidf. And on terminal will print respective accuracy on the test data.
   (d) *trainTfidf.py:* executing this file will produce logistic model and tfidf vectorizers in models folder after training it on the train data.
   (e) *trainDeepNLI.py:* Executing this file will automatically download data using torchtext.dataset and will train the model with the same hyper-parameters as in the final model. *Note:trainDeepNLI.py is importing nli_model.py to access deep neural model. classes*

2. **Libraries Used:**
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
import torch
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from tqdm import tqdm
import re
from scipy.sparse import hstack
from sklearn.linear_model import LogisticRegression
import joblib
from torchtext.data import Field, BucketIterator
import torch.nn as nn
import torch.nn.functional as F
from torchtext import data
from torchtext import datasets
import torch.optim as optim
```

3. **Data Pre-Processing**
   Following pre-processing have been done in the finalized model (NLTK used):

   (a) filtered out examples with no consensus
   (b) removed special characters using re library
   (c) lower cased the strings to eliminate redundant vocab
   (d) done tokenisation
   (e) done Lemmatization using WordNetLemmatizer from NLTK.stem

   Quick Note:
   Pre processing has been done for both the models (deep and logistic). And it has been observed that removing stop-words from the dataset resulted in decreased accuracy implying stop-words have a positive role to play in natural language inference.

4. **Logistic Model with Tf-Idf vector representation**

   - I have used TfIdf vectorizer from sklearn for representation of premises and hypothesis individually and then concatenated them to form a single input and the labels are also converted to integer numbers as defined in the labelMap() function. These formatted data is then feeded to the logistic model imported from the sklearn library.
   - TfIdf vectors of the data are stored in CSR format(compressed sparse row) to efficiently store the vectors with large vocab.
   - I have used 'saga' solver for the Logistic Model which is an efficient solver for multi-class classification with large data samples.
   - Predicted labels for the test data has been saved in "tfidf.txt"
   - Seeing the below confusion matrix, it's clear that this model is not biased towards any one class.

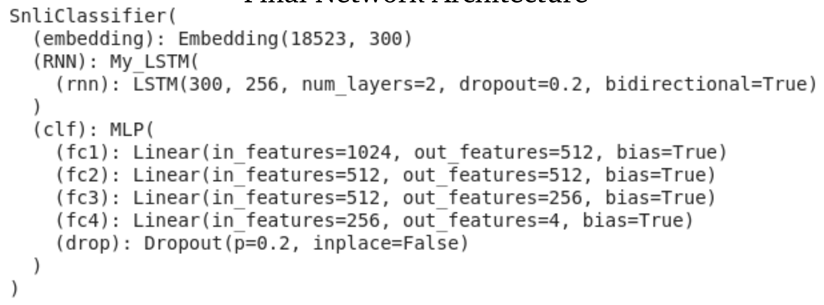
confusion matrix	pred 1	pred 2	pred 3
true 1	2389	497	482
true 2	572	2099	566
true 3	522	507	2190

   - **Mean Accuracy:** 67.9%

5. **Deep Model:**
Final Model Configuration:

Hyper-Parameters	Values
Learning Rate	0.0005
Optimizer	Adam
Scheduler	StepLR, gamma=0.1, step size=4
Loss function	Cross Entropy

Final Network Architecture



```
SnliClassifier(
  (embedding): Embedding(18523, 300)
  (RNN): My_LSTM(
    (rnn): LSTM(300, 256, num_layers=2, dropout=0.2, bidirectional=True)
  )
  (clf): MLP(
    (fc1): Linear(in_features=1024, out_features=512, bias=True)
    (fc2): Linear(in_features=512, out_features=512, bias=True)
    (fc3): Linear(in_features=512, out_features=256, bias=True)
    (fc4): Linear(in_features=256, out_features=4, bias=True)
    (drop): Dropout(p=0.2, inplace=False)
  )
)
```

Mean Accuracy:77.1%

(a) **WorkFlow:**

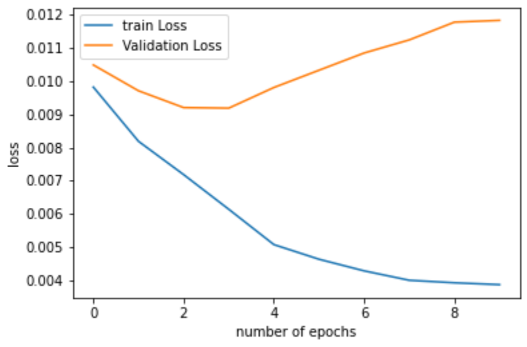
- Data pre-processing and split in train, dev(validation set), test data using torchtext.dataset.snli.splits library.
- Vocab creation with all data (train,dev,test) using torchtext.data.Field.buildVocab()
- Now unlike like tfIdf representation in case of Logistic, we will learn the embedding for the sentences by treating them as a learning parameter of the model. For this we use the Embedding class of nn module which generates a matrix of dimensions (vocab length, number of sentences) containing random real valued vector at the start then learns appropriate representation.
- These embedding is passed through LSTM word by word for the premise and the final state output is collected and similarly it's collected for hypothesis. Both these output are simply concatenated together to form the input for the input layer. Our LSTM is bidirectional which means it does same reading for the sentences in reverse order. Hence input layer size for feed forward network is 4 times the size of embedding $(2 * (embed(premise) + embed(hypothesis)))$.
- Finally the loss is computed at the output end of the feed forward network and back-propagated to learn the parameters.

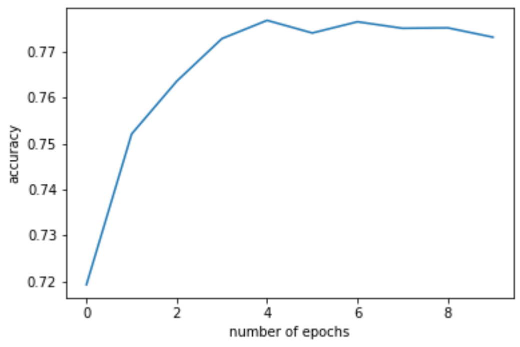
(b) **Explorations that worked:**

Sr.No.	Configurations		Accuracy	Remarks
1.	Embedding_dimensions	300	61%	*Training loss was fluctuating at high value, learning rate might be high*
	RNN_hidden_layers	1		
	hidden_features	256		
	drop_p	0		
	learning_rate	0.01		
	FF_hidden_layers	2		
	hidden_features	(512,512)		
	drop_p	0.2		
2.	Embedding_dimensions	300	68%	*Training loss reduced to some extent, possibly RNN needs to be deeper to learn the embedding better.*
	RNN_hidden_layers	1		
	hidden_features	256		
	drop_p	0		
	learning_rate	0.005*		
	FF_hidden_layers	2		
	hidden_features	(512,512)		
	drop_p	0.2		
3.	Embedding_dimensions	300	75%	*Training loss reduced (training loss = 0.011 before validation loss diverged) but not much yet, System needs more capacity. Increasing the FF_hidden layers might help.*
	RNN_hidden_layers	2*		
	hidden_features	256		
	drop_p	0		
	learning_rate	0.005		
	FF_hidden_layers	2		
	hidden_features	(512,512)		
	drop_p	0.2		
4.	Embedding_dimensions	300	78.04%* (best accuracy achieved)	*Final Model*
	RNN_hidden_layers	2		
	hidden_features	256		
	drop_p	0.2*		
	learning_rate	0.0005*		
	FF_hidden_layers	3*		
	hidden_features	(512,512,256)		
	drop_p	0.2		

Note: *many other explorations were done than what have been mentioned, but they didn't increase the accuracy to a noticeable point.*

(c) **plots:**





6. **Conclusion:**
   Deep Models have great capacity to learn representations, unlike tfidf representations, deep models can learn the representation in a better way which will be more consistent with the kind of problems being solved using suitable loss function. Hence deep model gives better accuracy than simple logistic regression on TfIdf representations.