DL ASSIGNMENT 3 PART2

[DRIVE LINK FOR SAVED MODEL], [COLAB LINK FOR THE CODE FILE]

1. You have to implement the architecture given in the paper Sentence Classification that measures the sensitivity of CNN to sentence classification. *We have implemented the static version of the architecture, the GloVe embeddings shall not be fine-tuned by our model.*

- The Architecture

```
self.conv1 = nn.Sequential(
    nn.Conv2d(1, n_filters, kernel_size=(3,glove_dim)),
    nn.BatchNorm2d(n_filters),
    nn.ReLU(inplace=True),
    nn.Dropout2d(p=0.5),
)

self.conv2 = nn.Sequential(
    nn.Conv2d(1, n_filters, kernel_size=(4,glove_dim)),
    nn.BatchNorm2d(n_filters),
    nn.ReLU(inplace=True),
    nn.Dropout2d(p=0.5),
)

self.conv3 = nn.Sequential(
    nn.Conv2d(1, n_filters, kernel_size=(5,glove_dim)),
    nn.BatchNorm2d(n_filters),
    nn.ReLU(inplace=True),
    nn.Dropout2d(p=0.5),
)

self.classifier = nn.Sequential(
    nn.Linear(3*n_filters, 6),

    nn.Dropout2d(p=0.5),
)
```

- The forward function to better explain the architecture

DL ASSIGNMENT 3 PART2

```
def forward(self, x):

  a = self.conv1(x)
  a = a.view(a.size()[0],a.size()[1],a.size()[2])
  a = self.m1(a).view(-1, self.n_filters)

  b = self.conv2(x)
  b = b.view(b.size()[0],b.size()[1],b.size()[2])
  b = self.m2(b).view(-1, self.n_filters)

  c = self.conv3(x)
  c = c.view(c.size()[0],c.size()[1],c.size()[2])
  c = self.m3(c).view(-1, self.n_filters)

  z = torch.cat((a,b,c), dim=1)
  out = self.classifier(z)

  return out
```

2. TREC Dataset, Coarse Labels, Preprocessing

- Removing the fine labels and encoding the coarse labels to integers.

DL ASSIGNMENT 3 PART2

```
f = open("/content/drive/MyDrive/DL_A3/Trec_Train_dataset.txt","r",
X = []
y = []

for line in f.readlines():
  ind = line.find(':')
  y_ = line[:ind]
  x_ = line[ind:]
  x_ = " ".join(x_.split()[1:])
  X.append(x_)
  y.append(y_)
```

```
labels_ = np.unique(y).tolist()
y_processed = []
for label in y:
  y_processed.append(labels_.index(label))
y_processed = np.array(y_processed).astype(np.long)
y_processed
```

```
array([1, 2, 1, ..., 5, 5, 2])
```

```
print(labels_)
```

```
['ABBR', 'DESC', 'ENTY', 'HUM', 'LOC', 'NUM']
```

- Splitting the sentences into tokens and padding the shorter sentences to make them all equal in length.

```
def get_tokens(arr):
  max_len=0
  tokenized=[]

  for text in arr:
    token=word_tokenize(text)
    tokenized.append(token)
    max_len=max(max_len,len(token))

  return tokenized, max_len
```

```
tokenized, max_len = get_tokens(X)
print(max_len)
```

```
37
```

- Encoding the tokens using pre-trained glove embeddings from “GloVe-840B-300dim”.

```
def get_embedding_matrix(tokens, max_len):
  embedded = []
  for word in tokens:
    embedded.append(glove[word])

  for i in range(max_len-len(tokens)):
    embedded.append(glove["<x>"].numpy())
  return np.stack(embedded)
```

```
X_processed = []
for tokens in tokenized:
  X_processed.append(get_embedding_matrix(tokens, max_len))

X_processed = np.stack(X_processed).reshape(-1,1,max_len,glove_dim)
X_processed.shape
```

```
(5452, 1, 37, 300)
```

3. Keep 80% of the data in point 2. for the Train set and 20% of the data for the Test set.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_processed, y_processed, test_size=0.2, stratify=y, random_state=42)
```

4. Use Word2vec or GloVe pre-trained embedding models for word representations.

```
import torch
import torchtext

glove = torchtext.vocab.GloVe(name="840B",dim=glove_dim, cache="/content/drive/MyDrive/DL_A3/vectors_cache")
```

DL ASSIGNMENT 3 PART2

5. Implement and give the analysis as in the paper in the following Sections:

a) Section 4.2 - Effect of input word vectors

Various encodings were tried in the paper.

i) Using pre-trained Word2Vec trained on 100B words, gave a 300 dimensional representation for every token in the sentence.

ii) Using pre-trained GloVe trained on 840B words, gave a 300 dimensional representation for every token in the sentence.

iii) Out of GloVe and Word2Vec, the competition was purely dependent on the dataset. One performed better on a dataset while the other performed better on another dataset. Thus, they tried concatenating these two 300 dimensional vectors to give a 600 dimensional representation which gave somewhat better results.

iv) They also tried long, sparse one-hot vectors for input word representations which were found to perform poorly with CNNs for the task of sentence classification. The authors argued that the sentence might be too brief for these long vectors to provide enough information.

Even though the 600-dim vectors performed better, they could become computationally expensive, hence either Word2Vec or GloVe should be the choice when it comes to sentence classification. However their performance depends on the dataset. In our case, we have used GloVe embeddings.

b) Section 4.6 - Effect of pooling strategy

i) 1-maxpooling: The maximum value is extracted from every feature map.

ii) K-maxpooling: The k maximum values are extracted from every feature map.

iii) Local 1-maxpooling: Every feature map is divided into regions of some size and the maximum value is extracted from every such local region from every feature map.

iv) Local average-pooling: Similar as above, every feature map is divided into regions of some size but instead of the max value, the average is taken from each region of every feature map.

The paper presented an analysis of all these strategies and the result was that 1-maxpooling outperformed all the alternatives for the task of sentence classification for all of the datasets tested in the paper.

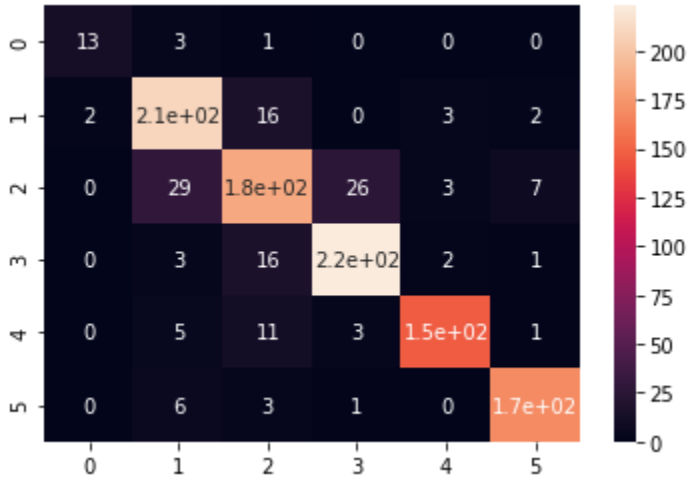
Reason:

“This may be because the location of predictive contexts does not matter, and certain n-grams in the sentence can be more predictive on their own than the entire sentence considered jointly.” - from the paper.

DL ASSIGNMENT 3 PART2

6. Report the following:

a) Confusion matrix



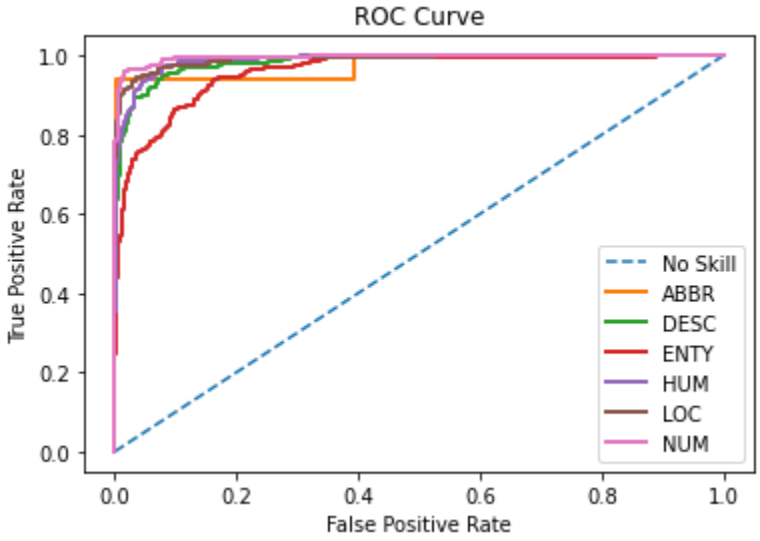
Note:

- Actual labels on the horizontal axis and predicted labels on the vertical axis.
- The labels 0 to 5 correspond to the following COARSE labels from the dataset.

```
print(labels_)
```

```
['ABBR', 'DESC', 'ENTY', 'HUM', 'LOC', 'NUM']
```

b) ROC



c) Evaluation Metrics (Macro Average Scores)

```
Precision: 0.8755153117223973
Recall: 0.856761852800673
F1: 0.8648600910240627
```

d) Accuracy

```
PATH = "/content/drive/MyDrive/DL_A3/A3_P2_Models/best_model_840B.pt"

checkpoint = torch.load(PATH)

model = CNN(checkpoint['glove_dim'], checkpoint['n_filters']).cuda()
model.load_state_dict(checkpoint['model_state_dict'])

epoch = checkpoint['epoch']
train_loss = checkpoint['train_loss']
val_loss = checkpoint['val_loss']
train_acc = checkpoint['train_acc']
val_acc = checkpoint['val_acc']
print (f'Epochs={epoch+1}, Train-Loss: {train_loss:.4f}, Val-Loss: {val_loss:.4f}, Tra
```

```
Epochs=73, Train-Loss: 0.3062, Val-Loss: 0.4712, Train-Acc: 96.97 %, Val-Acc: 86.80 %
```

Precision	Recall	F1-Score	Accuracy	Loss
0.87552	0.85676	0.86486	86.8011%	0.47116

e) All the assumptions/resource issues, if any, and the process to run your codes.

- Overview of the Code File
  - Importing the libraries
  - Importing the pre-trained GloVe embeddings from cache store in my google drive.
  - Importing the dataset and preprocessing as explained above.
  - Train-test split and creating data loaders from training-testing.
  - Define the Architecture in an nn.Module class.
  - Select the optimizer, criterion. Create a model instance.
  - Training begins, save the model as the validation loss improves.
  - Evaluate the best model.
- Assumptions/Issues
  - The static version of the model has been implemented, that is no fine-tuning of the GloVe embedding parameters.
  - BatchNorm2d layers have been added to solve the overfitting issue.
  - We do not have too many instances of class 0, which is why the Confusion Matrix might appear dark in the TP region for the class:ABBR.

DL ASSIGNMENT 3 PART2

References:

- https://torchtext.readthedocs.io/en/latest/vocab.html
- https://www.youtube.com/playlist?list=PLqnslRFeH2UrcDBWF5mfPGpqQDSta6VK4
- https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
- https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html