COMP 5600/6600

Assignment 3 – Deep Learning Applications

Due: 11/15/2024 11:59 pm

Submission: On Canvas

Part 1 **[25 points]**

In this assignment you will implement two deep learning algorithms to predict the genres of a movie from a short description given in text. This will be a multi-class classification problem since a movie can belong to multiple genres at the same time.

Models:
1. Implement a RNN model to predict the genre classes of a movie given the short textual description. There are 20 classes of genres in the dataset and each sample may have multiple genres to predict. You can use a sigmoid activation function to compute the probability of each classes and find multiple genres given a text description.

2. Implement a LSTM model to do the same and find accuracy, precision and recall and compare the numbers in your report.

3. The dataset contains the titles of the movies as well. Use the titles along with your descriptions and feed it to your one of your models and see if you can get any boost in your performance.

Dataset:
IMDB data from 2006 to 2016(https://www.kaggle.com/datasets/PromptCloudHQ/imdb-data) has 1000 samples with information Title, Genre, Description, Director, Actors, Year, Runtime, Rating, Votes, Revenue, Metascrore. For this assignment you will use only the Title, Genre and Description columns.To load the data you can refer to the following code snippet:

```
import kagglehub
path = kagglehub.dataset_download("PromptCloudHQ/imdb-data")
print("Path to dataset files:", path)
files = os.listdir(path)
print("Files in dataset directory:", files)
csv_file_path = os.path.join(path, 'IMDB-Movie-Data.csv')  #
Replace with the actual filename
data = pd.read_csv(csv_file_path)
print(data.columns)
print(data['Title'][1])
print(data['Genre'][1])
print(data['Description'][1])
```

Preprocessing:

Since the text description of the movies contain some noise and stopwords you need to preprocess and clean the data such as remove the stopwords, remove numbers, lemmatization etc. You can refer to the following code to do some preprocessing on your data or you can write your own.

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
stop_words = set(stopwords.words('english'))
print(stop_words)
lemmatizer = WordNetLemmatizer()
tokens = nltk.word_tokenize(data['Description'][1].lower())
print(tokens)
filtered_tokens = [lemmatizer.lemmatize(token) for token in
tokens if token not in stop_words and token.isalnum()]
print(filtered_tokens)
```

Since you cannot feed your text data directly into a neural network, you need to tokenize each word and then use some sort of vector embedding to represent your words. In this assignment, you will use the glove embedding vectors to represent each of your words. Also, the neural network will only take a fixed size of input to it's model, so you need to pad your text sequences to give it a fixed size. You can use the pad_sequences function from Tensorflow. Here is how you can download the Golve embedding and load the embedding vectors for all your words. This might take a while to download.

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
embeddings_index = {}
with open('glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
print('Found %s word vectors.' % len(embeddings_index))
```

Multi-Label labelling:

Since your dataset has multiple labels for each sample, you need to preprocess the labels as well. There are 20 classes of genres and you need to create a vector of size 20 for each sample such as only the corresponding indices of genres are 1 and all other values are 0. You can follow the following code to preprocess your labels:

```
from sklearn.preprocessing import MultiLabelBinarizer
genres = data['Genre'].apply(lambda x: x.split(','))
print(genres)
mlb = MultiLabelBinarizer()
genres_encoded = mlb.fit_transform(genres)
print(genres_encoded[0])
print(mlb.classes_)
```

Once you have done all the preprocessing for your data you can load the data using a custom dataloader from keras. Use first 700 samples for training, 100 samples for validation and the last 200 to test your model. Train your networks for 20 epochs. Plot the graphs for training and validation loss, accuracy etc.

Deliverables:
One .ipynb file with codes with data preprocessing and implementation of two models (feed forward and LSTM). Plot the graphs for loss and accuracy for training and validation. Report the numbers for test set for both models. Add titles to your text descriptions. Report the numbers for one of the models and compare with the previous model. Comment on how it helps or does not help.

Part 2 **[25 points]**

In this part you will implement a convolutional neural network to classify natural images into 6 classes. This dataset is called Intel Image Classification dataset which has 6 classes. The dataset has 14,000 training images; 3,000 validation and 7000 pred_seg images. You need to train your model using the 14000 training images and test it on 3000 validation images. You do not need to do anything on the 7000 pred_seg images. The images of the dataset belong to classes such as streets, buildings, woods, mountains, seas, and glaciers. The data is organized into subfolders with class names for both training and validation (test) sets. When you load the images to a dotaloader, you need to label the images according to your folder names. You can load the images using opencv library.

This is how you can download the data:

```
import kagglehub
import os
path = kagglehub.dataset_download("puneet6060/intel-image-
classification")
print("Path to dataset files:", path)
```

You can also download the images from the following link, unzip it and then upload to your google drive:

https://www.kaggle.com/datasets/puneet6060/intel-image-classification?resource=download

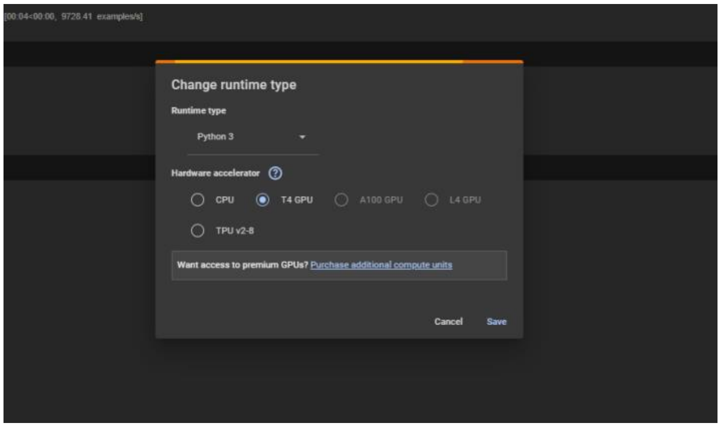
Models:

1. Implement one CNN model with 3 Conv layer and three and 3 max_pooling layers. Add a dropout layer and a dense layer before the output layer
2. Implement a model with 6 Conv layers and 3 pooling layers. Add a dropout layer and a dense layer before the output layer
3. Play around with different learning rates, batch sizes and optimizers.

Deliverables:
Include your code and report. Document your findings in a table and write a summary. Report your numbers in terms of total accuracy and per-class accuracy on the test set (3000 images). Visualize your prediction from test set with the label for 2 samples.

****Submit your .ipynb file with output cells. Please do not submit without running the code or with empty output cells.**
**** Use the free GPUs in google colab, otherwise it will take a long time to train your models**

Click on Runtime, change runtime type then select the available GPUs.



[00:04<00:00, 9728.41 examples/s]

Change runtime type

Runtime type

Python 3

Hardware accelerator

○ CPU ● T4 GPU ○ A100 GPU ○ L4 GPU

○ TPU v2-8

Want access to premium GPUs? Purchase additional compute units