Jessica Gallo

CSC767 Neural Networks and Deep Learning

Prof. Gueorguieva

December 4, 2020

Project 2

Part a:

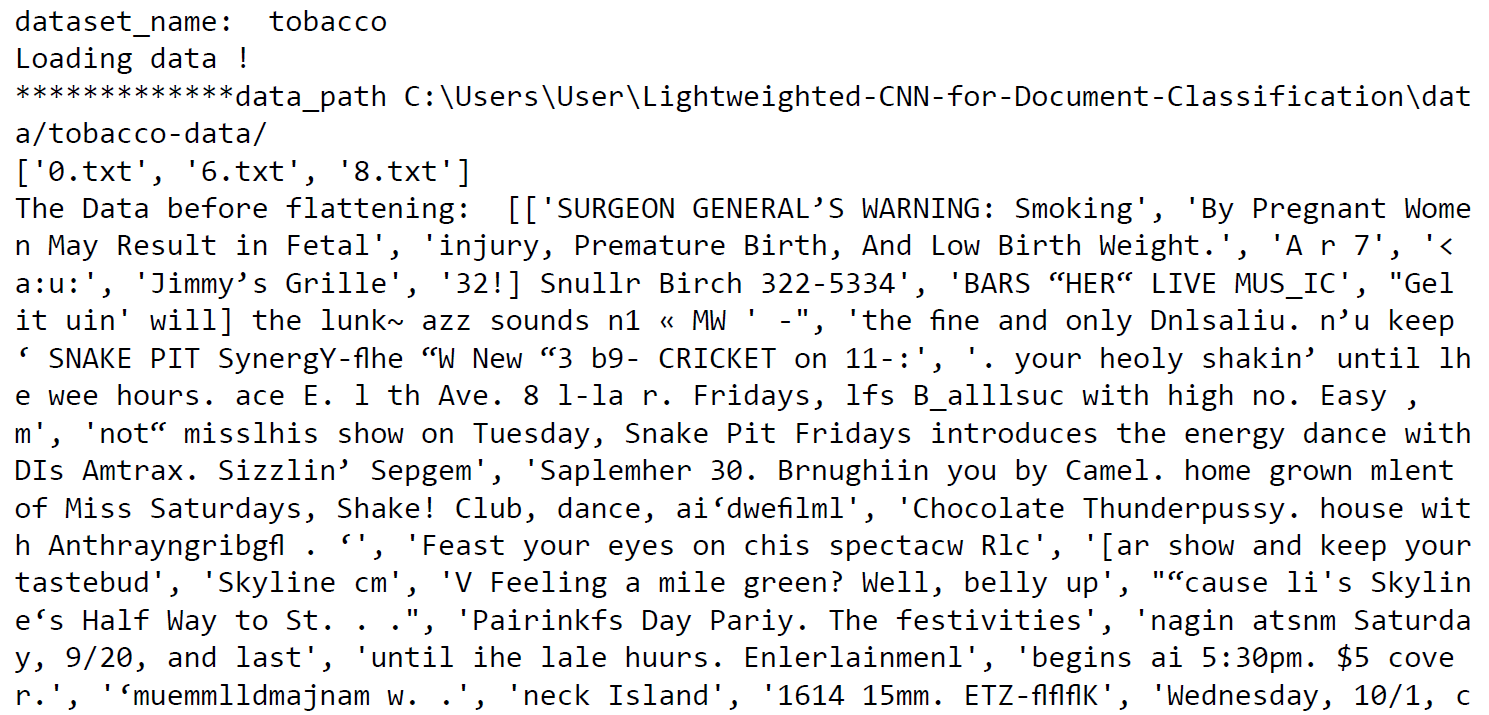
Problem Description: What is the problem that you will be investigating? Provide paper titles, authors and other details.

The problem I will be investigating is the Light-Weighted CNN for Text Classification by Ritu Yadav. The paper discuses a way to reduce trainable parameters and memory consumption. To do this, they looked into a new architecture that uses some new concept already existing in the image classification world, that can be used for text classification and cause lesser memory consumption. They came up with three solutions: an optimized way of training Text CNN network, a lightweight CNN and a CNN with a dual optimizer.

Part b:

Data: What data will you use? Usually in the papers they present experiments using more than one dataset. Pick up one that you will be using for your experiments. It could be a particular dataset from a chosen paper or one that you propose.

I will use the same data that the paper uses, which is the Tabaco3482 dataset. They give a data folder with text files labeled 0-9. I used data files 0.txt, 6.txt and 8.txt. These files contain text from the Tobacco-3482 dataset. The categories are [‘ADVE’, ‘Email’, ‘Form’, ‘Letter’, ‘Memo’, ‘News’, ‘Note’, ‘Report’, ‘Resume’, ‘Scientific’].



Part c:

Methodology/Algorithm: In the papers very often, they propose more than one approach. Pick up the one that you like and provide full details of it.

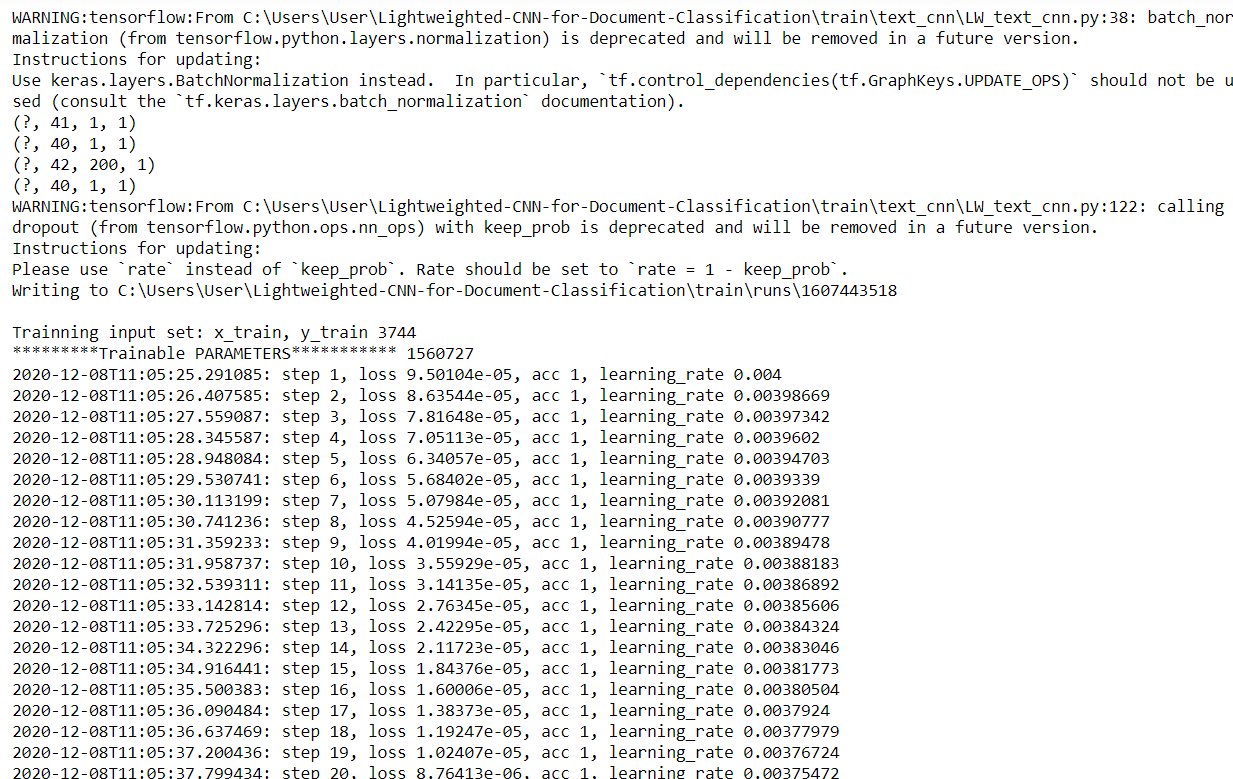
The paper offers three different approaches to achieve a reduced trainable parameters and memory consumption, which are an optimized Text CNN architecture, a lightweight CNN and a dual optimizer CNN. I have chosen the dual optimizer CNN. What this architecture does is take advantage of the two best optimizers and uses them at different stages of training. They used a hybrid simple strategy SWATS, that Switches from Adam to SGD when a triggering condition is satisfied. The Adam optimizer was picked because it helps the network learn fast, but after a few epochs, the learning process needs to slow down to stabilize performance. They used SGD with momentum as the second optimizer for the later stage of train since it performs better on later epochs. From their results, this particular solution they came up with cut the training time from the base TextCNN by and hour and a half. From their results, it only took 26 minutes to train, however the accuracy was not that high, but it was not the lowest. The accuracy was only at 43.5. The number of trainable parameters was greatly reduced from the base Text CNN. These results were from the larger Tabacco3482 dataset.

Part d:

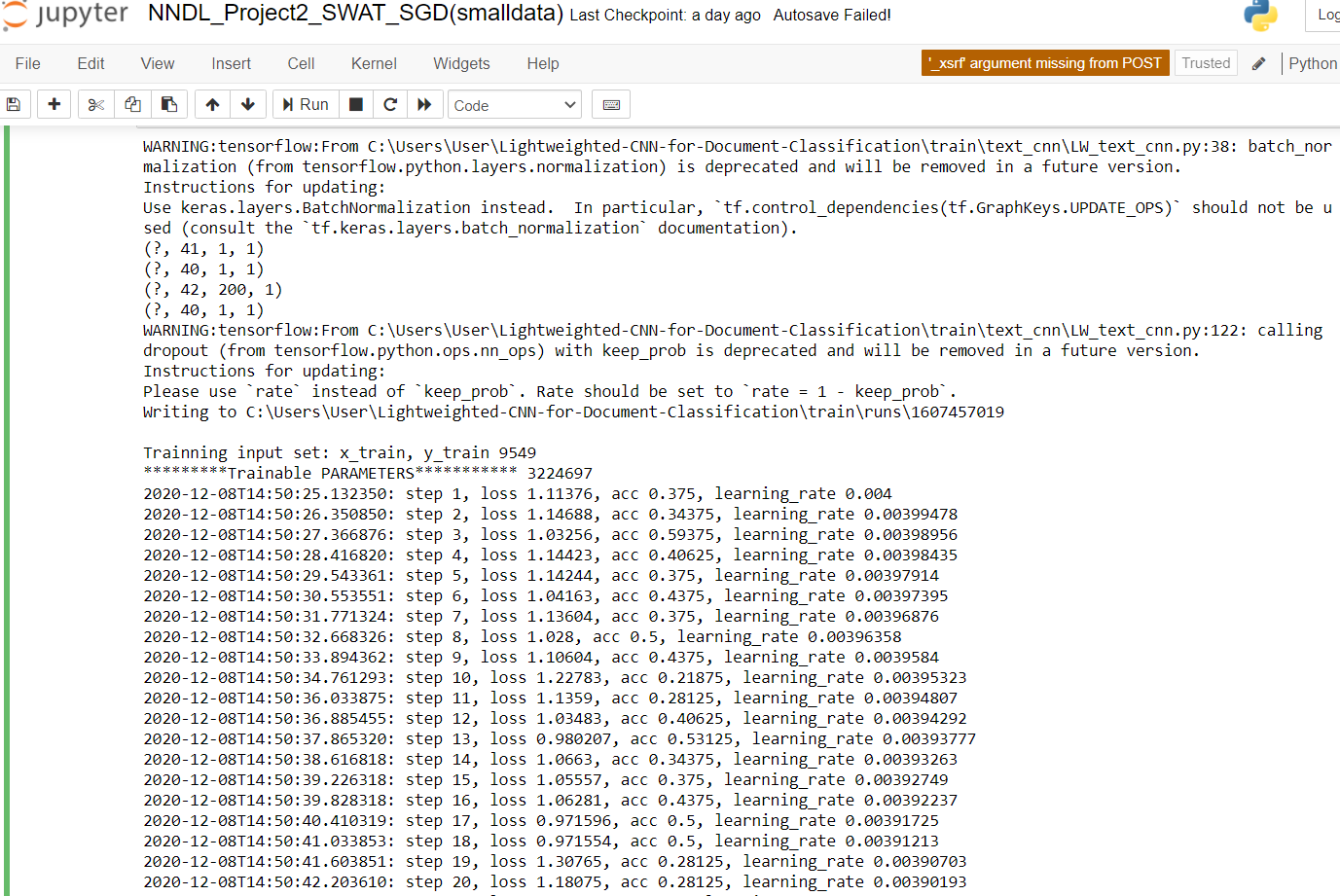
Your Approach: This section details the framework of your project. Be specific.

Our approach for this project is to change the optimizer. I decided that I would keep Adam for its swift training and the goal is to reduce training time, but I need an optimizer as good as SGD. I chose RMSProp. I chose this because it helps diminish learning rates by using a moving average of the squared gradient. It utilizes the magnitude of the recent gradient descents to normalize the gradient. In RMSProp, learning rate gets adjusted automatically and it chooses a different learning rate for each parameter. It also divides the learning rate by the average of the exponential decay of squared gradients. For the second test, I again kept Adam and changed SGD/RMSProp to Adadelta because it is basically Adam, which performs very well, but it implements momentum, like the original SGD to smooth gradients based on accumulated “first-order” information.

I wanted to run the original file (SWAT\_LW\_train.py) in Jupyter Notebook and see the results. I tried running it and experienced a lot of problems. Firstly, was that this code uses from tensorflow.contrib import learn, which is deprecated and no longer runnable on Tensorflow 2.0, so I had to change the environment to Tensorflow 1.14 in order for that to work. This was the first time using an older version of Tensorflow, and after some time working with the code, I realized I might have been able to keep our current Tensorflow 2.1 version and just add tf.compat.v1. to all the older lines with contrib, learn and VocabularyProcessor. The next problem was an error message with “unrecognizable flag error: unkown command line flag f”, so I added tf.compat.v1.flags.DEFINE\_string(‘f’, ‘’, ‘’) to resolve this error. The next error was UnicodeEncodeError:’charmap’ codec can’t encode character ‘\ufb01’ in position 282: character maps to <undefined>. To resolve this, I had to open the data\_helpers.py file. On line 65, I had to change data = list(open(read\_file, “r”).readlines()) to data = list(open(read\_file, “r”, encoding=’utf-8’).readlines()). I then added os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3' to fix tensorflow printing logging messages about memory allocation. Then I started training the model. The model was ok, but the accuracy was very low. I did not change any of the parameters, so there were 20 epochs, an evaluation after every 100 steps and a checkpoint save at every 100 steps as well. The training process was a bit different since it showed steps instead of epochs. After the first 100 steps trained, it was supposed to give an evaluation and save that checkpoint, but instead, the kernel buffered and died. I tried running it many times to no avail. The issue was with memory allocation and I received the error message: tensorflow allocation of exceeds 10% of system memory killed. To try to fix this, I tried to allocate more memory to Jupyter Notebook by opening jupyter\_notebook\_config.py and changing NotebookApp.max\_buffer\_size to 12000000000 bytes, which is 12GB. This lessened the memory allocation amount, but didn’t fix the issue. I also tried changing all float32 datatypes to float16 to use less memory. Neither of these worked. I then started cutting out the data files until I were only left with 0.txt. I also cut the epoch size to 10 and evaluating and saving checkpoints every 50 steps to help it run. The problem with keeping only 0.txt, though, was that it gave us very weird loss amounts and 100% accuracy:



Many combinations later, I finally found a combination that allowed better training while still being able to run without memory allocation problems. The data I used was 0.txt, 6.txt and 8.txt. These were the smallest files at 141KB, 51KB and 282KB, respectively. I added another folder to the data folder to put all the unused data files in, called cut-data. These are the results with 0.txt, 6.txt and 8.txt:



Our models ran a lot better than the original model described in the paper because the data was much smaller. The original file (which is the training shown above) is labeled NNDL\_Project2\_SWAT\_SGD(smalldata). The trainable parameters were much smaller than the original as well.

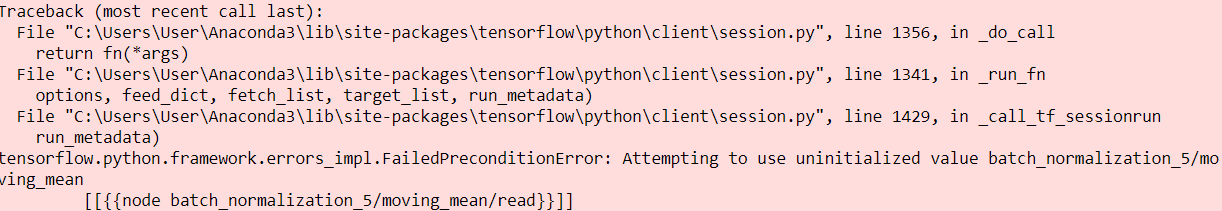
Used for all files:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epochs | Steps per Evaluation | Steps per Checkpoint | Data Used | Total Steps | Input Set | Trainable Parameters |
| 10 | 50 | 50 | 0.txt 6.txt 8.txt | 2950 | 9549 | 3224697 |

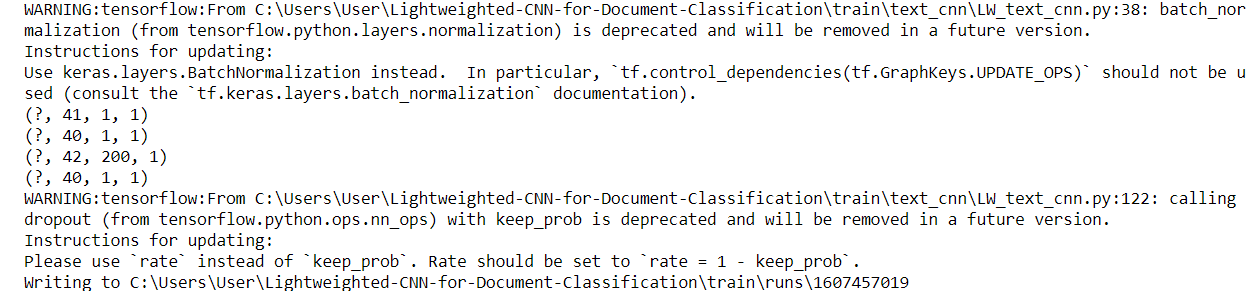
Training Information

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Elapsed Time | Loss First 1st Eval (Step 50) | Loss Last Eval  (Step 2950) | Accuracy  1st Eval  (Step 50) | Acc Last Eval  (Step 2950) |
| SWAT\_SGB (smalldata) | 34 min | 0.864921 | 0.749915 | 0.595727 | 0.724759 |
| SWAT\_RMSProp | 32 min | 0.876958 | 1.02848 | 0.561793 | 0.708421 |
| SWAT\_AdaDelta | 34 min | 0.87477 | 0.649049 | 0.562212 | 0.732719 |

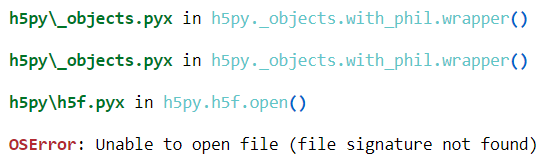
There were problems using the eval.py given. After fixing all the errors (such as removing compat.v1 where it was not needed and wouldn’t run with it there), it still gave us error messages . I tried running it directly after training with !python eval\_SGD\_small.py --eval\_train --checkpoint\_dir="./runs/1607457019/checkpoints/", which is the command it showed us to use in the readme.txt. The error is:



There seems to be a problem with batch normalization. The problem might be related to the WARNING message from training which says:



One thing to note is that for this training, I did not get the same warning message about batch normalization. I didn’t fix it when running the first time since it was just a warning about the older syntax I were using was deprecated. I tried running a different evaluation with model = load\_model('./runs/1607457019/checkpoints/model-2950.meta', compile=False) that I would normally use, but it can’t be used since the model would have to be in .h5 format, which it is not.



I then tried using the update showed in the Warning. The update was using tk.keras.layers. That only gave us errors since I would have had to change everything to tk.keras and change the format a bit. I tried many other different approaches to fix the issue, but nothing worked.

Evaluation Information with Eval.py for all files:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Total # test examples | Vocabulary Size | Train/Dev Split | Size of x\_dev | Size of y\_dev |
| 936 | 16099 | 9549/2387 | 2387 | 2387 |