Exercise IV

AMTH/CPSC 663 - Spring semester 2019

Due: Friday, April 12, 2019 - 11:59 PM

Compress your solutions into a single zip file titled `<lastname and initials>_ assignment4.zip`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment4.zip`. Include a single PDF titled `<lastname and initials>_assignment4.pdf` and any Python scripts specified. Any requested plots should be sufficiently labeled for full points. Your homework should be submitted to Canvas before Friday, April 12, 2019 at 11:59 PM.

Programming assignments should use built-in functions in Python and TensorFlow; In general, you may use the `scipy` stack [1]; however, exercises are designed to emphasize the nuances of machine learning and deep learning algorithms - if a function exists that trivially solves an entire problem, please consult with the TA before using it.

Problem 1 (5 pts)

Give a brief explanation of the following question.

- Check file `tfconv1.py`.

1. Explain image = china[150:220, 130:250].
2. Explain what is going on in lines 71-72.

- Check file `tfconv1_2.py`

3. What does the line $X_reshaped = tf.reshape(X, shape = [-1, height, width, channels])$ do?
4. Explain what is set up in lines 47-53.
5. Explain lines 55-56. What do the parameters of the pool command ksize and strides do? Why is pool3 reshaped? How does the pooling work?
6. What is fc1?

7. What is Y_proba in line 63?
8. Explain what is going on in lines 71-73.

- Check file `tfconv1_3.py`

9. Explain how the dropout (lines 66-69) works.
10. Explain what early stopping is, and how it is implemented in the program.

Problem 2 (5 pts)

1. Principal Component Analysis (PCA) is a very common machine learning method for dimensionality reduction. Conceptually, describe how the principal components in PCA are chosen. If PCA is to implemented in TensorFlow, the function tf.svd() will play an important role in the algorithm. Please briefly describe what this function does, and why the results will be useful in implementing PCA.
2. Using Tensorflow, implement linear autoencoders on the MNIST dataset with a single hidden layer with 4, 8, and 16 nodes. Select two test examples of your results. Compare the results of your autoencoder with the original images. Include in your report both the original images and the reconstructed images (there should be 8 images in total). Save your code as `prob3_2.py`.
3. Describe the similarity between PCA and an autoencoder.
4. What is the difference between a convolutional autoencoder and linear autoencoder?
5. What similarities and differences are there between a denoising autoencoder and a variational autoencoder?

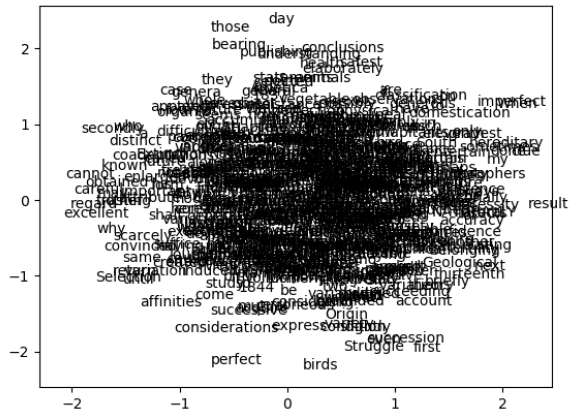
Problem 3 (10 pts)

Implement an LSTM autoencoder to learn your own word embeddings:
Fill in the TODOs in `rnn.py` to train your own LSTM and learn word embeddings for a custom text file of your choosing.

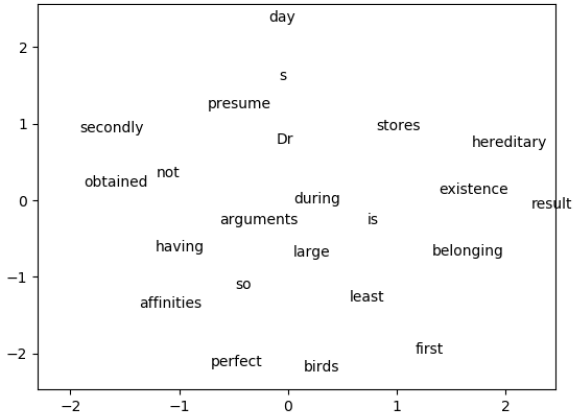
1. TODO 1: Choose any text you would like and put it into a file in your working directory. Put the name of the file into the variable in this line.
2. TODO 2: Create an embedding matrix with random initial values, using any initializing function you would like. This matrix has a row for each word in the vocabulary, with each row containing a learnable embedding of size *EMBEDDING_SIZE*. Create a TensorFlow object to look up the embedding value for each token in x. Hint: read about tf.nn.embedding_lookup.
3. TODO 3: Define an LSTM model for an encoder that takes the input embeddings you created in the previous step and produces a final state. There are many possible ways to do this, but some useful functions might be: `dynamic_rnn`, `LSTMStateTuple`, `LSTMCell`, and `MultiRNNCell`.

4. TODO 4: Define an LSTM model for a decoder that takes the final state of your encoder and produces a sequence of outputs that will be trained to reproduce the original sequence. You should be able to reuse much or all of the code from the previous part.

5. TODO 5: Plot the word embeddings with the given code. You are able to adjust the plotting parameters to suit your needs for making a compelling visualization. Discuss what you notice in your embeddings. For example, using the introduction to Charles Darwin's *On the Origin of Species* as a text file, I obtain this embedding:



Since this is too crowded to interpret, we've provided code to randomly select words to plot as long as there is space:



Either use this code multiple times or create your own code to obtain a visualization(s) that facilitates allows you to learn something about your data.

References

[1] "The scipy stack specification." [Online]. Available: https://www.scipy.org/stackspec.html