CSE 4392 - Assignments - Assignment 8

List of assignment due dates.

The assignment should be submitted via Canvas. Submit a file called assignment8.zip, containing all source code files needed to run your solutions for the programming tasks. Your Python code should run on Google Colab, unless permission is obtained via e-mail from the instructor or the teaching assistant.

All specified naming conventions for files and function names are mandatory, non-adherence to these specifications can incur a penalty of up to 20 points.

Your name and UTA ID number should appear on the top line of answers.pdf and all source code files.

Task 1 (100 points, programming)

Files you need to download for this task:

- dataset.zip: The dataset we will use.
- reverse_main.py: An incomplete program, that you will complete by writing the appropriate code in a file that should be called reverse_solution.py.
- reverse_common.py: A file containing auxiliary functions used in reverse_main.py.

In this task, you will implement a system that solves the following problem: the input is a piece of text (a sentence), that is either in the correct form, or it contains the words in reverse order. The probability of either case is 50%. The output should be the sentence with its words in the right order.

The training, validation and test data for this program are stored at file dataset.zip, which you should download and unzip. File reverse_test.txt contains 1205 examples of inputs and target outputs for your system. Every line in that file contains an example input, the <TAB> ("\t") character, and then the target output. Here are two examples:

- Example 1:
  - Input: happy people make to like i
  - Target output: I like to make people happy.
  - Comment: here, the input contains the words in reverse order, and the output contains the words in the correct order.
- Example 2:
  - Input: she came to see us yesterday
  - Target output: She came to see us yesterday.
  - Comment: here, the input contains the words in the correct order, and the output contains the words in the the same order as the input.

Note that in reverse_test.txt the input sentence is in standardized form (lower case, no punctuation), whereas the target sentence is in non-standardized form. Your system can produce its output in standardized form, it is NOT expected to produce upper case characters and punctuation. Your system's output will be compared with

the standardized form of the target output.

File reverse_train.txt contains 5736 sentences, and reverse_validation.txt contains 1214 example sentences. You should use these sentences to obtain the training and validation sets for training your model. It is entirely up to you how you will produce training and validation sets based on the sentences in these files. Every line in these files is a single sentence, with words in the correct order.

All sentences in this dataset come from the Spanish-English translation dataset from http://www.manythings.org/anki/. The dataset that you are given contains sentences that were selected from the original dataset based on these criteria:

- Only sentences in English were used.
- Only sentences with length between 5 words and 8 words were used.
- The vocabulary of all words appearing in the selected sentences consists of 250 words. This will allow you to create neural networks with fewer parameters for the word embedding layers, compared to datasets with larger vocabularies.

File reverse_main.py contains an incomplete program that trains and evaluates neural network models for this task. These models can be used to convert an input sentence to the corresponding output sentence where words appear in the correct order.

To complete that code, you must create a file called reverse_solution.py, where you implement the following Python functions:

- `(model, source_vec_layer, target_vec_layer) = train_enc_dec(train_sentences, validation_sentences, epochs)`

  The `train_enc_dec` function takes as arguments:

  - `train_sentences`, which is a list of the sentences in file reverse_train.txt.
  - `validation_sentences`, which is a list of the sentences in file reverse_validation.txt.
  - `epochs`, which is the number of epochs you should use for training your model.

  Your function should somehow (how exactly is up to you) use the training and validation sentences to create appropriate training and validation sets, and then it should use those sets to train an encoder-decoder RNN model (or encoder-decoder Transformer model, if you prefer that option). The function returns the trained model, as well as the source vectorization layer and the target vectorization layer that will be needed for model evaluation.

- `results = get_enc_dec_results(model, test_sentences, source_vec_layer, target_vec_layer)` The `get_enc_dec_results` function is used to map test input sentences (where the word order can be correct or reverse) to output sentences (where the word order is hopefully correct). It takes the following arguments:
  - `model`, which is the Encoder-Decoder model produced by your `train_enc_dec` function.
  - `test_sentences`, which is the complete set or a smaller subset of the input examples from reverse_test.txt. You may want to use a smaller subset of 100 sentences or so if you want to do some quicker tests.
  - `source_vec_layer` and `target_vec_layer`, which are the source vectorization layer and target

vectorization layer produced by your `train_enc_dec` function.

As you can see in reverse_main.py, the results produced by the `get_enc_dec_results` function are evaluated using function `word_accuracy`. This function is implemented in file reverse_common.py, and counts the percentage of words in each result sentence that are equal to the word in the same position in the corresponding target sentence.

- `(model, source_vec_layer, target_vec_layer) = train_best_model(train_sentences, validation_sentences, epochs)`

  The `train_best_model` function takes the same arguments (except that it does not take an `epochs` argument) and returns the same type of results as `train_enc_dec`. The only difference is that in `train_best_model` you can train any type of model you like, it does not have to be an Encoder-Decoder RNN (or Transformer). Since you have a lot more freedom in designing the model, you can hardcode in your implementation the number of epochs that you think is best for your model.

- `results = get_best_model_results(model, test_sentences, source_vec_layer, target_vec_layer)`

  The `get_best_model_results` function takes the same arguments and returns the same type of results as `get_rnn_model_results`. The difference is that `get_best_model_results` should be implemented so that it works appropriately with the model that is produced by your `train_best_model` function.

Some Additional Information

Here is some information about my solution, that you may choose to use or not in your solution:

- For the Encoder-Decoder model:
  - I used the same basic structure that was used in the slides for the Encoder-Decoder RNN model that did English-to-Spanish translation.
  - I used 64 dimensions for the word embeddings.
  - I used 64 "latent dimensions", i.e., 64 units for each recurrent layer.
  - I trained the model for 150 epochs. On my computer, training took about 2-3 seconds per epoch.
  - I trained 10 models. The worst word accuracy was 69.84%, and the best word accuracy was 74.07%.
- For the "best" model:
  - Overall, the model did not have more parameters or take longer to train than my Encoder-Decoder RNN model.
  - I trained my "best" model 10 times. The worst word accuracy was 94.30%, and the best word accuracy was 97.71%.

Expected Results and Grading

80 points will be given for correct implementation of the Encoder-Decoder model. Your model should reach or exceed 70% word accuracy on the test set, to match the word accuracy (70%-74%) that I got for my Encoder-Decoder model. We will also look at your code, to ensure that you implemented an Encoder-Decoder model and not some other type of model. If your implementation does not achieve 70% word

accuracy, as long as it does implement an Encoder-Decoder model, it will get at least 4 points for every percentage point by which its word accuracy exceeds 50%. For example, if the word accuracy is 58%, you will get at least 32 points. If the word accuracy is under 50%, partial credit will be given on a case-by-case basis, based on how close or far your implementation was from being correct.

20 points (and possibly six extra credit points) will be given for your implementation of the "best" model. Obviously, here you have a lot more freedom in how to design it. For every extra percentage point over 74% in word accuracy you receive one point. If your word accuracy is worse than 74%, zero points will be given. If your word accuracy is over 94%, you get extra credit points.

Task 1b (Extra Credit, maximum 10 points)

10 points will be given to each of the three solutions that give the best word accuracy on the test set for the Encoder-Decoder model.

To qualify for consideration, you need to run your solution 10 times (or more, if you want), and report a summary of results in answers.pdf. Your result summary should report smallest, largest, mean, and median of the word accuracies on the test set that your solution achieved. You should also document in answers.pdf the design choices that you made.

Task 1c (Extra Credit, maximum 10 points)

Here, you are allowed to use more training data, and/or do transfer learning from other models that you may find or build yourself, and/or do anything else you like to achieve the best accuracy. 10 points will be given to each of the three solutions that give the best word accuracy on the test set. Obviously, you are not allowed to use your test data as training or validation data.

To qualify for consideration, you need to run your solution 10 times (or more, if you want), and report a summary of results in answers.pdf. Your result summary should report smallest, largest, mean, and median of the word accuracies on the test set that your solution achieved. You should also document in answers.pdf if you added additional training data (and what that data was), what base models you used for transfer learning (if you used transfer learning), and any other design choices you made.

CSE 4392 - Assignments - Assignment 8