Advanced Audio Processing: Exercise 05

Language modeling with recurrent neural networks

1 Introduction

Text is one of the major modalities of stimuli, closely associated with audio. There are many research directions where the input to a method is audio and the output is text, for example, speech recognition (speech-to-text) and audio captioning. Also, there are tasks where the input to a method is text and the output is audio, e.g. text-to-speech. As presented in the lectures, words in a text exhibit temporal associations, where some words are really likely to be with (i.e. precede or follow) other ones. For example, “for example” or “I am”. Learning these associations is termed as language modeling. In this exercise you will implement a language model using recurrent neural networks (RNNs), according to what you have been taught in the lectures, and using data from an existing dataset of the audio captioning task.

The rest of the document is organized as follows. Section 2 you will develop the processes for handling the dataset for this task, and in Section 3 you will develop your language model.

2 Getting and setting up the dataset (1.5 points)

For this exercise you will be using a subset of the freely available Clotho dataset for audio captioning, which can be found online [1]. Clotho consists of audio files and CSV files. The latter contain the text, a subset of which you will be using. In this section, you will download the dataset and set-up the routines for reading the data from CSV files, and set-up the data-loader for feeding the data to your RNN-based language model.

2.1 Downloading and reading the dataset (1 point)

From the Moodle page of the course, download the file **clotho_captions_subset.csv** file. This file is a CSV file, having six columns: i) file_name, ii) caption_1, iii) caption_2, iv) caption_3, v) caption_4, and vi) caption_5. You will have to make a function in order to read the contents of all columns, except the file name. Also, you will have to process the text, make all captions to have the same amount of words (i.e. the same length), and append and prepped specific tokens that indicate the start and the end of a sequence.

Specifically, you have to:

1. Download the **clotho_captions_subset.csv** from the Moodle page of the exercise, and add the file to the root directory of your code for this exercise.
2. Use the built-in csv.DictReader class in order to read the contents of all columns and put them in a list. Each element in the list should have the contents of each cell (i.e. each element in the list should be one caption). Implemented in read_captions function.
3. Remove all punctuation from all captions. Implemented in preprocess_captions function.
4. Add to the beginning of each caption the string <sos>.

5. Make all captions have the same length in number of words, by truncating the long ones and appending the string <eos> to the shorter ones. Use as threshold the length of 14 words.
6. Append (once more) the string <eos> to all captions, so that all captions will end with the <eos> string and all will have the same length.
7. Create a set of unique words in all captions. That is, make a list (ordered set) that it will have the unique words that exist in all captions.
8. Create a one-hot encoded version of all captions (i.e. making each word one-hot encoded in each caption).

Your code should be placed in a file called **data_handling.py**, which will be submitted for this exercise. For the one-hot encoding, you can use the functions in the provided **aux_functions.py** file. Note bold that for creating the one-hot encodings, you will need the list of the unique words.

2.2 Creating the dataset and data loader (0.5 points)

In order to use the data in the CSV file with PyTorch and create your language model, you have to create your dataset and data loader. You will use the function created in the previous task, and create the data set class. Then, you will use your dataset class in a data loader. Have in mind that at your RNN-based language model, the output of the linear layer will be a vector, containing an one-hot encoding of the predicted word. This means that you need to have a list of the unique words in your text, so you can assign a number (e.g. index in a list) to each word. Also, this number should be in a one-hot encoding format.

Additionally, your dataset object should provide as an input a sentence from your text (i.e. the content of a column from the CSV) and as a targeted output the same sentence but shifted by one word. For example, if the content of a column is “A person is turning a map over and over.”, then using the <sos> and <eos> tokens, the sentence will be “<sos> A person is turning a map over and over <eos>”. The input that the dataset will provide would be:

<sos> A person is turning a map over and over

and the targeted output would be

A person is turning a map over and over <eos>

Please note, that all these words should be one-hot encoded. This means that the input at time-step 0 will be the one-hot encoding of “<sos>” and the targeted output would be the one-hot encoding of “A”.
Thus, you have to:

1. Create a dataset class, which will read the data from the CSV file, using the functionality developed in the previous task. You should also use the list of indices for the words and the one-hot encoding form of the captions.
2. When asked for the next item (i.e. in the __getitem__ function), the dataset should return the input and output arrays, according to the above scheme.

Your code should be placed in a file called **dataset_class.py** given as a template. For the one-hot encoding, you can use the provided function in the provided **aux_functions.py** file.

3 Creating and sampling a language model (1.5 points)

For this task, first you will create an RNN-based language model, and then you will generate some text using your developed language model (i.e. you will sample your language model). Only the creation of the language model is graded, the sampling is to show you how one can use a language model to automatically generate some text and is an optional and non-graded task. To do the sampling of the language model, you have to serialize your learned language model by using the `torch.save` function, serializing the state/weights of the language model (obtained by `.state_dict` method). More information about saving and loading a model in PyTorch can be found online [2].

3.1 Creating a language model (1.5 points)

To create the language model, you will use an RNN-based deep neural network (DNN), with a linear layer followed by a softmax nonlinearity at the end. The inputs and outputs to the DNN will be provided by the dataset and data loader that you created at the previous task of this exercise. For your RNN-based DNN, you can use any of the GRU classes in PyTorch (i.e. either `toch.nn.GRU` or `toch.nn.GRUCell`). As a loss function, you will use the `CrossEntropy` loss.

The steps are:

1. Firstly, use a linear layer with input dimensionality equal to the amount of your unique words (i.e. equal to the length of the one-hot encoding) and output features equal to 64, followed by a tanh nonlinearity. This linear layer will get as an input the one-hot encoding of each word and export an embedding (the 64 long vector) for each word. This embedding will be the input to the RNN for each time-step.
2. Create a three-layered GRU-based DNN, using input features 64 and output 64. The GRU DNN will get as an input the output of the initial linear layer.
3. Then, create the classifier of your system using as input features 64 and output equal to the length of your one-hot encoding.
4. Use the Adam optimizer to optimize your layers for 200 epochs.
5. Train your layers for at least 200 epochs. At each 20 epochs, observe their predicted output for a segment of your data. That is, use a subset of your training data (e.g. by using a second loop and putting a break statement where needed) as an input to your layers, and decode the output.

Your code should be placed in a file called **language_model_system.py** and **language_model_training.py**, which will be submitted for this exercise together with **dataset_class.py** and **data_handling.py**.

3.2 Sampling your language model (0 points)

To sample your language model, you have to iteratively give inputs to it and record its outputs. First, you start by giving as an input the <sos> token, and record the output of your language model, e.g. W_1. Then, you give to the language model as inputs the <sos> and W_1,

and record the next output, e.g. W_2. Again, you give as an input the <sos>, W_1, and W_2, and record the next output, e.g. W_3. This process continues until the time step t, where W_t =<eos>. The resulting sequence of words (i.e. W_1, W_2, . . . , W_t) is the generated text of your language model.

The steps to follow are:

1. Employ your RNN-based language model from the previous task.
2. Load the state dict of your language model, using the function torch.load and the method load state dict.
3. Given as an input to your model the one-hot encoding of the <sos> token and record the output to use with the <sos> as next inputs.
4. Give to your model as input the <sos> and the output from the previous step, and record the output so you can use it with the current inputs, as future inputs.
5. Repeat the process until your language model outputs <eos>

If you implement the above, then submit with your code a .txt file with the text that your language model generated. Happy text generation!

References

[1] https://zenodo.org/record/3490684
[2] https://pytorch.org/tutorials/beginner/saving_loading_models.html