NNFL Sem 2 Assignment 2 Readme

PART 1

Aim: To familiarise students with deep neural architectures and coding them, in PyTorch. Additionally, to create an exercise to simulate the typical workflow that follows in a research paper implementation in order to prepare students for the next assignment.

Objective: To build a Seq2Seq Neural Machine Translator on English-Hindi data.

Instructions: This file is a readme for the assignment. All functions and classes have been sufficiently described here. Kindly go through this file carefully and follow the step-wise instructions to successfully understand and implement this assignment. There are multiple fill in the blanks throughout this assignment that you will have to complete which will be evaluated disjointly.

Note: We DO NOT expect you to train the model. The evaluation will be based on function-specific test cases and code. However, you are free to try and run the code if you have the required computation power and time.

Also keep the bidirectional flag **False** everywhere. We are **NOT** going to train a bidirectional model, only a unidirectional one.

Important: DO NOT import any extra modules, change any lines of code, rename or edit function prototypes, change class names or default values etc. unless specifically asked to do so in the evaluative parts. Changing other code may cause the code to break.

System Requirements - Pytorch - Minimum version 1.5.0, 1.6.0 preferable.

Install all requirements from requirements.txt before starting - USE the command

pip install -r requirements.txt (We recommend you to use this inside an environment)

Directory Structure (Top-down):

1. **main.py**: Contains all code to instantiate appropriate classes and call appropriate functions in the correct order from all other files. Non evaluative.

2. **validate.py**: Contains code to test the model once it has been trained. There is a bin file with model weights which have been previously trained, you will be loading this model into the code and evaluating the test instances using this model so that we can obtain consistent results across all assignments. Partially evaluative.

3. **trainer.py**: contains code to train the model. This file is partially evaluative.

4. **seq2seq.py**: Contains the architecture of the model. The architecture is a simple encoder and decoder with uni-directional LSTM and feed-forward network. Partially evaluative.

5. **preprocess.py**: code to load and pre-process the dataset. Partially evaluative.

6. **Utils.py**: some extra utility functions, non evaluative

Implementation (Bottom-Up):

In the implementation part, we will follow a bottom-up approach. We will start with implementing data processing. This will be followed by implementation of parts of the network architecture including the seq2seq encoder and decoder layers. Finally, we will write the train function and validation function that will complete the implementation.

1. preprocess.py:

NON - Evaluative:

pad_sequences: pads or truncate sequences to a appropriate max_length

conver_to_tensor: takes a wordtoindex dictionary and sentences and converts them to input tensors.

class Data () - Dataset class

Evaluative:

I. preprocess(sentence: str, hindi=False) -> str: ***(0.5 mark)***

This function is used to carry out some basic preprocessing steps. there are 4 code blanks where you need to write the answer: "CODE_BLANK_1", … "CODE_BLANK_4".

CODE_BLANK_1 - Convert the sentence into lowercase

CODE_BLANK_2 - remove trailing or leading extra white spaces.

CODE_BLAMK_3 - remove any extra white spaces from within the sentence. Eg: <space><space>like<space>cats. Should become I<space>like<space>cats. [Note - here <space> is representative of " " - the actual space, the special token was just used for easy visual representation].

CODE_BLANK_4 - Prepend and Append respectively the start of sentence token ("SOS" - without quotes) and end of sentence token ("EOS" - without quotes) to the sentence.

II. *get_vocab(lang: pd.Series) -> List:* ***(0.5 marks)***

This functions is used to form a list of the vocabulary in the dataset.Takes sequence of sentences (in form of a panda series) and returns the list of vocabulary in sorted order. List of vocabulary refers to a list of all unique ords that are a part of any of the sentences.

Eg - ["I love dogs", "I hate cats"]. output should be: ["cats", "dogs", "hate", "I", "love"]

III. token_idx(words: List) -> Dict ***(0.5 marks)***

This function assigns a unique index to each word in the vocabulary. Takes a list of words and returns a dictionary with each unique word having a unique index. Note: in this dictionary - the index of PAD is 0, SOS is 1, and, EOS is 2, the index of the rest of the vocab starts from 3. So essentially the key value pair for this dictionary is that the word would be key and the unique index would be its value

Eg - if the input was the list in the example above, the output would be: {'PAD':0, 'SOS': 1, 'EOS': 2, 'cats': 3, 'dogs': 4, 'hate': 5. 'I': 6, 'love': 7}

IV. idx_token(wor2idx: Dict) -> Dict: **(0.5 marks)**

This is the function that will be later used to obtain the word token corresponding to a particular index. This function takes the dictionary formed in the previous function as input and reverses the dictionary. That is the (key, value) pair reverses.

Eg - {'0': 'PAD', '1': 'SOS', '2': 'EOS', ….}

V. get_dataset(batch_size=2, types="train", shuffle=True, num_workers=1, pin_memory=False, drop_last=True) ***[1.25 mark]***

This function is only partially evaluative - please check that you are writing code only where it is supposed to be written and not altering anything else.There are multiple codeblanks in this function, fill in your answer t each code blank appropriately

CODE_BLANK_1 - Remove duplicate examples from dataframe 'lines'

CODE_BLANK_2 - Call the preprocess function on english sentences in the dataframe.

CODE_BLANK_3 - Call the preprocess function on the hindi sentences in the dataframe

CODE_BLANK_4 - Retrieve length of each english sentence and store it in the lines dataframe under a new column "length_english_sentence"

CODE_BLANK_5 - Retrieve length of each hindi sentences and store it in the lines dataframe under a new column "length_hindi_sentence"

CODE_BLANK_6 - remove all data points where the length of english sentence is less than max_length

CODE_BLANK_7 - remove all data points where the length of hindi sentence is less than max_length

CODE_BLANK_8 - Get the english vocabulary (list of english words) by calling appropriate function with correct arguments.

CODE_BLANK_9 -Get the hindi vocabulary (list of hindi words) by calling appropriate function with correct arguments.

CODE_BLANK_10 -Get the word to index dictionary for english by calling correct function

CODE_BLANK_11 -Get the word to index dictionary for hindi by calling correct function

CODE_BLANK_12 -Get the index to word dictionary for english by calling correct function

CODE_BLANK_13 -Get the index to word dictionary for hindi by calling correct function

CODE_BLANK_14 - Convert the english sentence to tensors using appropriate functions and dictionaries created above.

CODE_BLANK_15 - Convert the hindi sentence to tensors using appropriate functions and dictionaries created above.

2. se2seq.py:

This file contains the architecture of our model. It has three classes - an encoder class for the encoder RNN, a decoder class for the decoder RNN and a thid class for the linear layers.

Only the forward functions of all three of these architectural components are evaluative. You are however encouraged to go through the non-evaluative parts of the code that is already give, especially the _init() functions to understand the different initializations and attributes of each of the architectural components.

Evaluative

I. EncoderRNN.forward(self, input, hidden,bidirectional=False): ***(0.75 marks)***

Apply the self.embedding function on the input and then change the shape of this calculated tensor to 1,1,-1 using view function. (CODE_BLANK_1). This is now the rnn_input. on the rnn_input and hidden variable from the function definition apply self.lstm function and save the output, hidden state and cell state in three different variables (CODE_BLANK_2).. Return two objects finally - the output state and the **tuple** created using hidden and cell state.(CODE_BLANK_3)

II. DecoderRNN.forward(self, input, hidden) ***(0.75 marks)***

Apply the self.embedding function on the input and then change the shape of this calculated tensor to 1,1,-1 using view function. Let this variable be called output.(CODE_BLANK_1). On output and hidden, apply the self.lstm function and again store this in output and (h_n ,c_n) (CODE_BLANK_2). Apply self.out on output[0] we calculated in CODE_BLANK_2 just in the previous step(CODE_BLANK_3) .

III. *Linear.forward(self, input)****(0.75marks)***

CODE_BLANK_1 : Initialize self.bidirectional with the value of variable "bidirectional" passed in init.

CODE_BLANK_2 : Calculate the number of directions using the formula *bidirectional + 1* and store this in num_directions. NOTE: Bidirectional is a boolean variable so you need to convert it to a suitable type before adding 1 to calculate directions

CODE_BLANK_3 : If the connection_possibility_status flag is true then return input as it is, else return the output that is obtained after applying linear layer (self.linear_connection_op) on the input.

3. train.py

This file contains the training loop for the model.It is partially evaluative, look for code blanks carefully and fill them in without changing any other code.

EVALUATIVE

I. train(input_tensor, target_tensor, mask_input, mask_target, encoder, decoder, bridge, encoder_optimizer, decoder_optimizer, bridge_optimizer,device, criterion, max_length,batch_size=32,bidirectional=False,teacher_forcing=True): ***[1.5 mark]***

Train function called for each batch for each epoch. Fill in the blanks carefully

CODE_BLANK_1 - call the initHidden method of encoder class.

CODE_BLANK_2 - initialise the encoder outputs to zero tensor, use appropriate arguments to get the correct shape using batch_size, max_length, encoder.hidden_size. (change comment in train.py)

CODE_BLANK_3 - call the encoder function with appropriate arguments

CODE_BLANK_4 - assign the hidden and cell state

CODE_BLANK_5 - Assign the decoder input as a torch tensor with value [SOS] and make sure you load it on proper device -HINT : SOS_token is the name of the variable

CODE_BLANK_6 - Assign the initial decoder hidden states as the last hidden states of encoder retrieved previously

CODE_BLANK_7 - call the decoder function with appropriate arguments

CODE_BLANK_8 - calculate loss normalised with batch size

CODE_BLANK_9 - same as CODE_BLANk_7

Code_BLANK_10 - same as CODE_BLANK_8

CODE_BLANK_11 - call backprop on loss.

CODE_BLANK_12 - update encoder optimizer weights

CODE_BLANK_13 - update decoder optimizer weights

CODE_BLANK_14 - return loss value normalised with target length

II. trainIters(trainloader,encoder, decoder, bridge,device,bidirectional=False,teacher_forcing=True,num_epochs=600,batch_size=32,max_length=20, print_every=1000, plot_every=100, learning_rate=0.1): ***[0.5 mark]***

Training iterations while keeping track of everything

CODE_BLANK_1 - replace "_,_ " the for loop below correctly

CODE_BLANK_2 - assign training data to training pair

CODE_BLANK_3 - assign input tensor

CODE_BLANK_4 - assign target tensor

<u>4. validate.py</u>

This file contains code to evaluate/validate the model. Its partially evaluative, fill in the CODE_BLANKS appropriately without changing any other code

I. evaluate(encoder, decoder, bridge, input_tensor,device,index2word_hin, max_length=20,bidirectional=False): ***[0.75 marks]***

CODE_BLANK_1 - call the encoder for input tensor

CODE_BLANK_2 - assign the hidden state and cell state from the last layer

CODE_BLANK_3 - Assign the decoder input as a torch tensor with value [SOS]

CODE_BLANK_4 - Call decoder function with appropriate arguments

CODE_BLANK_5 - Append EOS token to list of decoded words

CODE_BLANK_6 - Append the converted word from the tensor using appropriate dictionaries to the list of decoded words

II. evaluateRandomly(encoder, decoder, bridge,device,testset,idx2word_en,idx2word_hin, n=10): ***[0.75 marks]***

CODE_BLANK_1 - assign the value of data to pair

CODE_BLANK_2 - get non-zero values from the input tensors

CODE_BLANK_3 - get non-zero values from the out_put_tensor

CODE_BLANK_4 - Call the SentenceFromTensor_ method using the idx2word dictionaries and the input t tensor

CODE_BLANK_5 - Call the SentenceFromTensor_ method using the idx2word dictionaries and the output tensor

#CODE_BLANK_6 - call evaluate method with appropriate arguments