Deep Learning
Assignment 2 Report
M22MA003

Question 1: You have been provided a DATASET, which contains pairs of the words (x,y) i.e. akhbaar
अख़बार in which the first word is a Latin word(words we usually type while chatting with friends in WhatsApp) and the second word is its corresponding word in native script. Your main goal is to train a seq2seq model which takes as input the romanized string and produces the corresponding word in native script.
For Example, Jabki yah Jainon se km hai. ⇒ जबकि यह जनै ◌ों सेकम है।
a) Build a seq2seq model which contains the following layers -

(i) input layer for character embeddings
(ii) one encoder which sequentially encodes the input character sequence (Latin)
(iii) one decoder which takes the last state of the encoder as an input and produces one character output at a time (native).
Please note that the dimension of input character embeddings, the hidden state of encoders and decoders, the cell(LSTM), and the number of layers in the encoder and decoder should be passed as an argument.

a) Building Seq2Seq Model :

Solution 1(a) :-

Steps Followed:-

1. **Importing the data and preprocessing the data.**
    a. Using a compatible Pytorch version and torchtext version to be able to utilize BucketIterator in further process. Versions used are torch=1.8.0 and torchtext=0.9.0
    b. Using TabularDatasets to import the data into train, validate and test datasets.
2. **Creating the vocabulary from the data.**
    a. Tokenizing each data row by returning a list of the strings in the dataset.
    b. Using Fields to specify how the data should be tokenized.
    c. Here are some examples of tokenized data:-

```
<class 'torchtext.legacy.data.dataset.TabularDataset'>
Number of training examples: 44204
Number of validation examples: 4358
Number of testing examples: 4502
{'trg': ['अ', 'ं'], 'src': ['a', 'n']}
```

d. Number of unique token with min frequency three:-

```
Unique tokens in source hindi vocabulary: 65
Unique tokens in target english vocabulary: 30
```

e. Using BucketIterator to create train_loader, valid_loader and test_loader.

3. **Building the Model.**
   a. Defining the Encoder class with forward function. Utilizing the embedding and LSTM from torch.nn.
   b. Defining the Decoder class with forward function. Here also, utilizing the embedding and LSTM from torch.nn.
   c. Defining the Seq2Seq LSTM model with forward function in which we call the Encoder class and keep its output and pass it to the Decoder class.

```
Seq2Seq(
  (encoder): Encoder(
    (embedding): Embedding(30, 16)
    (rnn): LSTM(16, 16, dropout=0.5)
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (decoder): Decoder(
    (embedding): Embedding(67, 16)
    (rnn): LSTM(16, 16, dropout=0.5)
    (fc_out): Linear(in_features=16, out_features=67, bias=True)
    (dropout): Dropout(p=0.5, inplace=False)
  )
)
```

   d. Build the vocabulary and get the input and output dimension using the library.
   e. Define parameters according to the usage and create a LSTM model by calling Seq2Seq class, inside call Encoder and Decoder class.

```
The model has 7,043 trainable parameters
```

b) Now train your model using the standard train, test, and val data provided in the dataset.
Try below mentioned hyperparameters and draw the correlation table along with the plot (loss/accuracy VS. hyperparameter) for LSTM.
(i) Input embedding size: 16, 64
(ii) number of encoder layers: 1,3
(iii) number of decoder layers: 1,3
(iv) hidden layer size: 16,64

Solution 1(b) :-
Steps Followed:-

1. Import the required libraries - numpy, pandas, torch, torchtext, torchvision
2. Since we have already loaded the dataset and created the model in the previous part of the questionnaire, we will now define the required functions to call the model.
3. Define the train LSTM method and use the train dataloader and dropout value of 0.5.
4. Define the test method and evaluate using test_dataloader keeping the same dropout value.
5. Defining the utility functions such as init_weight and plot_graph.
6. Using below two combinations to train the LSTM models.

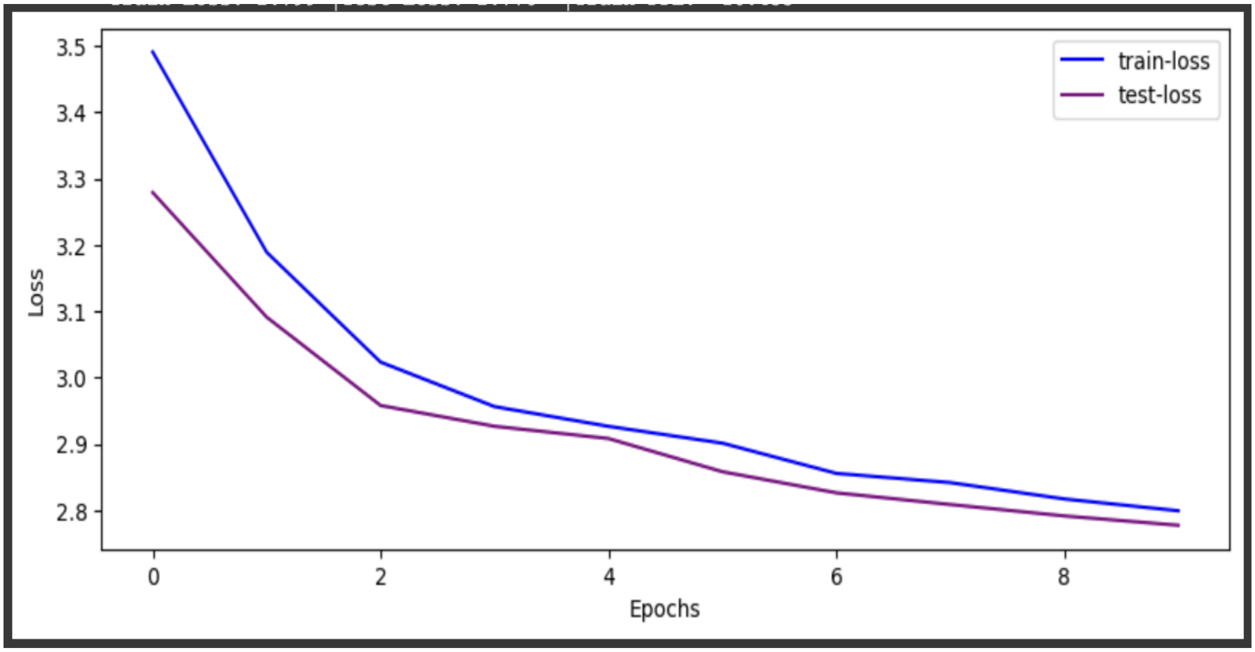
(i) Input embedding size: 16	**(i) Input embedding size: 64**
(ii) number of encoder layers: 1	**(ii) number of encoder layers: 3**
(iii) number of decoder layers: 1	**(iii) number of decoder layers: 3**
(iv) hidden layer size: 16	**(iv) hidden layer size: 64**

7. Training the LSTM model for Hyperparameters values:-
   Epochs = 10, Batch Size = 128 and Learning Rate = 0.001

8. Results :-
   Traditional LSTM Combination 1 : (16,1,1,16)
   Output of the Train/Test method :-
   Epoch: 01 | Time: 0m 27s
   Train Loss: 3.491 |Test Loss: 3.279
   Epoch: 02 | Time: 0m 16s
   Train Loss: 3.189 |Test Loss: 3.091
   Epoch: 03 | Time: 0m 16s
   Train Loss: 3.023 |Test Loss: 2.958
   Epoch: 04 | Time: 0m 17s
   Train Loss: 2.956 |Test Loss: 2.927
   Epoch: 05 | Time: 0m 17s
   Train Loss: 2.927 |Test Loss: 2.908
   Epoch: 06 | Time: 0m 17s
   Train Loss: 2.901 |Test Loss: 2.858
   Epoch: 07 | Time: 0m 17s
   Train Loss: 2.856 |Test Loss: 2.826
   Epoch: 08 | Time: 0m 17s
   Train Loss: 2.842 |Test Loss: 2.809
   Epoch: 09 | Time: 0m 18s
   Train Loss: 2.817 |Test Loss: 2.792
   Epoch: 10 | Time: 0m 16s
   Train Loss: 2.799 |Test Loss: 2.778

Observation : In 10 epochs, the lowest loss value observed is 2.79 training loss.

Plotting the results for Traditional LSTM Combination-1 (16,1,1,16) :-



Traditional LSTM Combination-1 (16,1,1,16) Correlation Table:-

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	318046	46	288	22	0	11	0	0	0	12	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	40969	1521	849	328	0	247	0	0	0	259	0	18	0	0	0	0	13	0	0	0
0	0	0	5815	11490	2126	6241	537	719	0	0	0	39	0	214	0	0	0	0	1	0	0	0
0	0	0	2033	2337	9030	3276	409	395	6	1605	0	485	0	258	0	0	0	36	73	0	0	1
0	0	0	1530	4475	2434	9658	774	106	0	0	0	11	0	69	0	0	0	0	0	0	0	0
0	0	0	1048	5073	901	4457	636	108	0	0	0	8	0	35	0	0	0	0	1	0	0	0
0	0	0	2216	2916	1960	881	128	1519	9	1624	0	287	0	576	0	0	0	27	93	0	0	0
0	0	0	943	1219	3622	1539	90	354	8	2782	0	292	0	209	0	0	0	47	116	0	0	0
0	0	0	1087	716	2774	1181	91	242	9	4169	0	187	0	132	0	0	0	111	64	0	0	0
0	0	0	1132	1555	4049	1639	90	384	5	968	0	270	0	181	0	0	0	17	85	0	0	1
0	0	0	3196	1385	2206	1121	238	500	6	0	0	1168	0	442	0	0	0	0	63	0	0	0
0	0	0	3548	3734	673	1567	95	251	0	0	0	25	0	51	0	0	0	0	0	0	0	0
0	0	0	1215	2023	2138	543	65	1352	8	846	0	319	0	798	0	0	0	5	86	0	0	2
0	0	0	624	1075	2222	1076	151	352	3	2773	0	152	0	205	0	0	0	46	47	0	0	0
0	0	0	1871	3965	470	1826	188	250	0	0	0	14	0	46	0	0	0	0	0	0	0	0
0	0	0	366	868	3277	1197	74	336	5	1908	0	141	0	190	0	0	0	36	37	0	0	0
0	0	0	214	304	2056	879	61	169	2	3591	0	142	0	95	0	0	0	100	15	0	0	0
0	0	0	1446	562	3256	393	26	627	3	373	0	269	0	246	0	0	0	3	321	0	0	0
0	0	0	1420	3606	425	1507	110	168	0	0	0	7	0	24	0	0	0	0	0	0	0	0
0	0	0	484	584	1843	680	42	341	12	1320	0	137	0	137	0	0	0	31	21	0	0	0
0	0	0	460	624	1143	250	27	574	3	1815	0	129	0	228	0	0	0	16	21	0	0	1
0	0	0	566	758	1169	346	32	475	7	1075	0	138	0	251	0	0	0	17	9	0	0	0
0	0	0	656	486	2157	355	50	148	0	623	0	162	0	153	0	0	0	7	39	0	0	0
0	0	0	177	1860	357	1959	293	24	0	0	0	0	0	24	0	0	0	0	0	0	0	0
0	0	0	155	497	737	292	18	388	2	2287	0	64	0	104	0	0	0	21	16	0	0	0
0	0	0	407	1016	550	396	79	320	2	933	0	71	0	187	0	0	0	8	12	0	0	0
0	0	0	130	242	1425	448	27	164	2	1008	0	127	0	59	0	0	0	25	18	0	0	0
0	0	0	495	564	1006	209	15	221	2	546	0	131	0	208	0	0	0	8	20	0	0	0
0	0	0	195	1389	255	1008	122	28	0	0	0	1	0	5	0	0	0	0	0	0	0	0
0	0	0	76	328	823	317	14	169	2	886	0	50	0	82	0	0	0	21	7	0	0	2
0	0	0	6	3	26	10	0	119	0	2206	0	0	0	6	0	0	0	40	0	0	0	0
0	0	0	113	128	1016	231	29	91	2	353	0	67	0	71	0	0	0	14	20	0	0	0
0	0	0	455	751	166	215	6	57	0	0	0	2	0	47	0	0	0	0	0	0	0	0
0	0	0	14	112	418	210	19	108	5	701	0	29	0	43	0	0	0	17	2	0	0	0
0	0	0	85	122	353	112	5	79	2	802	0	34	0	35	0	0	0	11	5	0	0	1
0	0	0	99	44	1251	114	4	24	0	8	0	44	0	23	0	0	0	1	21	0	0	0
0	0	0	38	944	48	435	123	10	0	0	0	3	0	3	0	0	0	0	0	0	0	0
0	0	0	320	79	147	10	1	318	0	611	0	23	0	59	0	0	0	7	7	0	0	0
0	0	0	21	141	475	93	10	103	0	539	0	42	0	96	0	0	0	7	0	0	0	1
0	0	0	87	17	41	2	0	129	0	1183	0	12	0	17	0	0	0	21	0	0	0	0
0	0	0	384	296	433	201	5	58	0	0	0	25	0	36	0	0	0	0	13	0	0	0

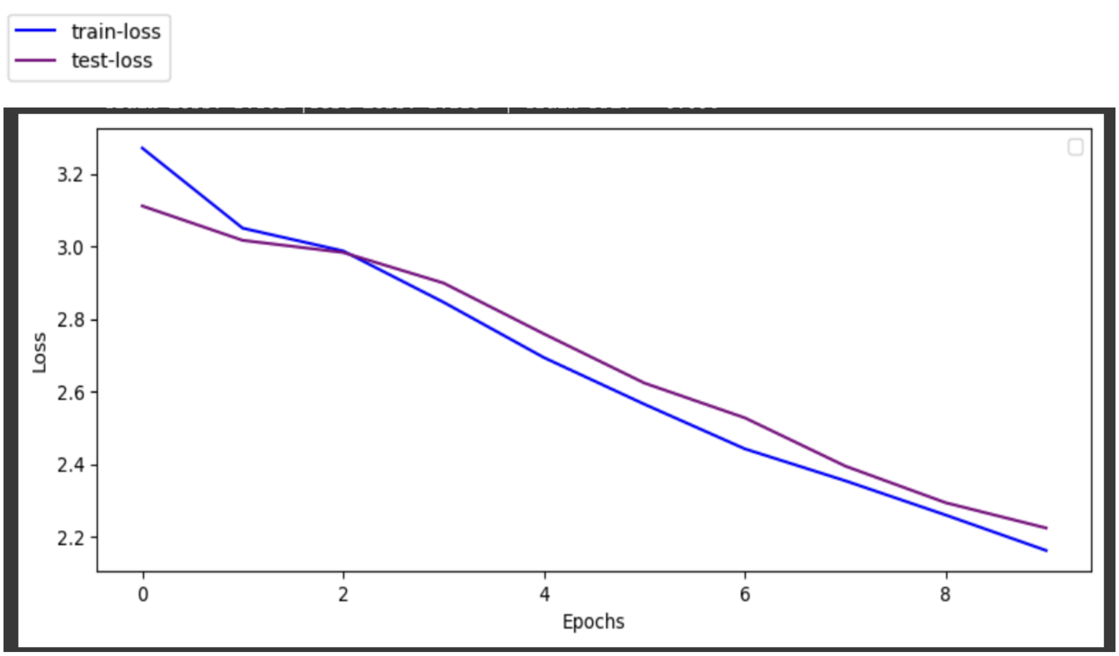
Traditional LSTM Combination-2 : (64,3,3,64)

Output of the Train/Test method :

Epoch: 01 | Time: 1m 22s
Train Loss: 3.266 |Test Loss: 3.105
Epoch: 02 | Time: 1m 19s
Train Loss: 3.038 |Test Loss: 2.978
Epoch: 03 | Time: 1m 20s
Train Loss: 2.969 |Test Loss: 2.889
Epoch: 04 | Time: 1m 19s
Train Loss: 2.849 |Test Loss: 2.739
Epoch: 05 | Time: 1m 20s
Train Loss: 2.691 |Test Loss: 2.553
Epoch: 06 | Time: 1m 20s
Train Loss: 2.537 |Test Loss: 2.412
Epoch: 07 | Time: 1m 18s
Train Loss: 2.431 |Test Loss: 2.299
Epoch: 08 | Time: 1m 19s
Train Loss: 2.324 |Test Loss: 2.178
Epoch: 09 | Time: 1m 21s
Train Loss: 2.214 |Test Loss: 2.039
Epoch: 10 | Time: 1m 21s
Train Loss: 2.104 |Test Loss: 1.916

Observation : In 10 epochs, the lowest loss value observed is 2.162 training loss.

Plotting the results Traditional LSTM Combination-2 (64,3,3,64) :-



Traditional LSTM Combination(64,3,3,64) Correlation Table:-

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	317616	164	135	41	23	29	41	432	50	375	23	18	9	4	0	8	31	61	17	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	38857	1655	578	323	156	271	154	211	148	561	369	97	78	106	1	15	179	228	38	24
0	0	0	3382	14220	1702	2379	1195	838	139	132	179	229	240	180	552	142	122	94	319	466	105	42
0	0	0	1666	2217	8647	2382	504	41	637	440	767	78	85	165	113	28	1006	279	453	20	93	5
0	0	0	1000	3143	1411	9524	1486	168	174	321	203	71	215	174	197	48	105	171	95	148	64	28
0	0	0	792	2158	472	2496	4039	141	183	111	107	139	252	183	66	149	25	26	83	455	37	7
0	0	0	1047	2373	220	362	291	4464	105	111	26	667	60	703	267	92	26	26	462	463	84	33
0	0	0	763	1161	1494	804	443	38	3995	378	356	89	128	402	33	19	117	72	194	31	181	4
0	0	0	595	412	682	1017	155	52	235	5653	197	48	16	97	25	30	17	1048	74	20	4	1
0	0	0	820	1408	3253	1138	211	35	677	455	1144	40	79	66	40	17	291	181	159	8	57	1
0	0	0	1166	1184	228	287	385	593	223	197	54	3427	44	945	230	69	14	20	273	216	128	89
0	0	0	3120	1388	364	1227	1129	173	104	86	84	87	1035	166	43	258	15	15	91	308	30	15
0	0	0	770	888	197	327	498	553	336	169	17	445	290	3565	69	47	4	14	351	249	227	34
0	0	0	427	1709	135	237	76	337	43	41	19	231	12	88	4086	7	13	18	201	26	91	412
0	0	0	1777	1617	289	1295	1410	251	95	115	55	119	402	165	34	299	10	16	37	400	26	13
0	0	0	222	1029	2715	793	79	20	186	34	233	11	21	38	48	6	2482	98	216	3	59	2
0	0	0	93	211	651	773	45	35	37	1255	150	9	4	23	5	6	18	4009	48	6	2	0
0	0	0	1187	868	792	182	81	535	248	100	78	401	24	395	184	5	149	43	1519	85	285	3
0	0	0	785	1184	154	665	1114	396	39	86	23	223	200	134	28	247	1	12	119	1630	11	8
0	0	0	312	720	372	266	153	127	295	22	43	135	57	456	96	4	64	3	359	11	1641	12
0	0	0	247	662	62	93	42	166	22	101	3	146	10	132	1868	0	5	21	71	20	58	927
0	0	0	229	553	57	138	135	296	37	19	7	326	34	311	224	14	6	4	100	79	119	143
0	0	0	375	317	1595	312	94	15	460	552	252	41	24	127	14	16	54	87	50	13	6	6
0	0	0	151	1054	188	1138	1068	69	52	40	39	76	74	53	74	59	11	23	21	107	12	5
0	0	0	68	401	40	106	95	143	17	14	0	153	18	199	208	8	2	0	30	26	65	211
0	0	0	243	1347	240	35	11	172	25	21	23	55	2	34	346	0	32	11	79	8	32	20
0	0	0	84	350	757	307	37	19	53	641	89	22	0	17	9	3	74	349	84	3	2	0
0	0	0	335	398	161	112	106	61	261	37	26	100	66	621	32	1	12	2	72	18	574	13
0	0	0	156	632	141	668	552	80	29	47	19	59	96	46	22	87	9	16	17	153	9	4
0	0	0	42	473	271	107	47	24	455	13	41	15	1	73	19	4	38	13	105	5	93	2
0	0	0	3	7	14	5	2	23	3	5	0	1	0	3	14	0	1	5	6	0	1	5
0	0	0	70	375	94	57	4	65	18	3	6	21	0	21	173	0	32	2	189	0	276	2
0	0	0	375	554	75	158	49	31	50	9	10	8	115	50	29	14	4	1	13	36	34	5
0	0	0	5	266	8	28	3	35	2	0	2	27	0	4	348	0	0	1	22	0	10	112
0	0	0	61	49	139	96	54	6	263	187	13	12	6	54	1	16	0	101	15	4	2	1
0	0	0	59	75	642	146	9	24	24	106	108	6	0	2	9	0	18	198	62	0	0	2
0	0	0	19	771	77	151	230	27	6	29	3	54	9	26	16	41	2	2	7	84	5	2
0	0	0	252	51	26	11	3	226	26	86	2	38	6	97	27	8	0	10	88	6	41	15
0	0	0	14	207	140	32	10	8	330	1	1	2	1	21	9	0	24	4	49	1	54	1
0	0	0	58	15	24	9	2	35	6	9	2	1	1	19	10	0	4	6	16	0	9	2
0	0	0	196	406	110	64	38	172	9	3	9	85	6	31	34	1	5	7	201	37	12	3

c) Now add an attention network to your LSTM seq2seq model and train your model along with the hyperparameter tuning. (you can use single or multiple attention layers)
(i) Plot the accuracy/loss.
(ii) Report the test accuracy.
(iii) Does the attention-based model perform better than the LSTM? Proof to support your answer.

Solution 1(c) :-

Steps Followed:-

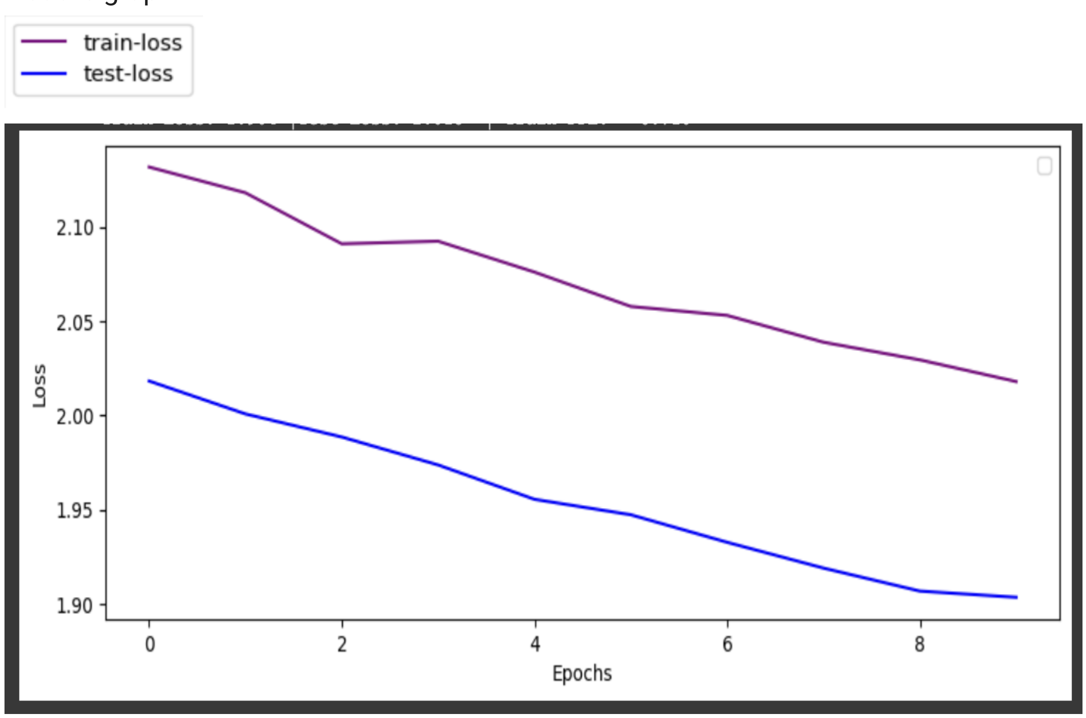
1. Building the LSTM Attention Model by adding an attention network.
2. First, we define two fully connected layers in the Encoder to keep the hidden state (short-term) and cell state (long-term). Then, we will use torch.cat which we help in concatenating the hidden state to the features dimension.
3. Now, defining the Decoder with attention. Energy value is calculated by concatenating the hidden state and encoder output and then applying activation function on top of this value.
4. Permute the attention weights and encoder output to align them.
5. Modifying the LSTM layer to accept the attention weights as input and compute a weighted sum of the input sequence using the attention weights.
6. Defining the Seq2Seq LSTM model with forward function in which we call the Encoder class and keep its output and pass it to the Decoder class.
7. Training the LSTM model for Hyperparameters values:-
   Epochs = 10, Batch Size = 128 and Learning Rate = 0.001

Attention LSTM Combination 1 : (16,1,1,16)

Output of the Train/Test method :

Epoch: 01 | Time: 0m 20s
    Train Loss: 2.018 |Test Loss: 2.132
Epoch: 02 | Time: 0m 19s
    Train Loss: 2.001 |Test Loss: 2.118
Epoch: 03 | Time: 0m 20s
    Train Loss: 1.989 |Test Loss: 2.091
Epoch: 04 | Time: 0m 20s
    Train Loss: 1.974 |Test Loss: 2.092
Epoch: 05 | Time: 0m 19s
    Train Loss: 1.956 |Test Loss: 2.076
Epoch: 06 | Time: 0m 22s
    Train Loss: 1.947 |Test Loss: 2.058
Epoch: 07 | Time: 0m 21s
    Train Loss: 1.933 |Test Loss: 2.053
Epoch: 08 | Time: 0m 19s
    Train Loss: 1.919 |Test Loss: 2.039
Epoch: 09 | Time: 0m 20s
    Train Loss: 1.907 |Test Loss: 2.029
Epoch: 10 | Time: 0m 19s
    Train Loss: 1.904 |Test Loss: 2.018

Plot the graph :-

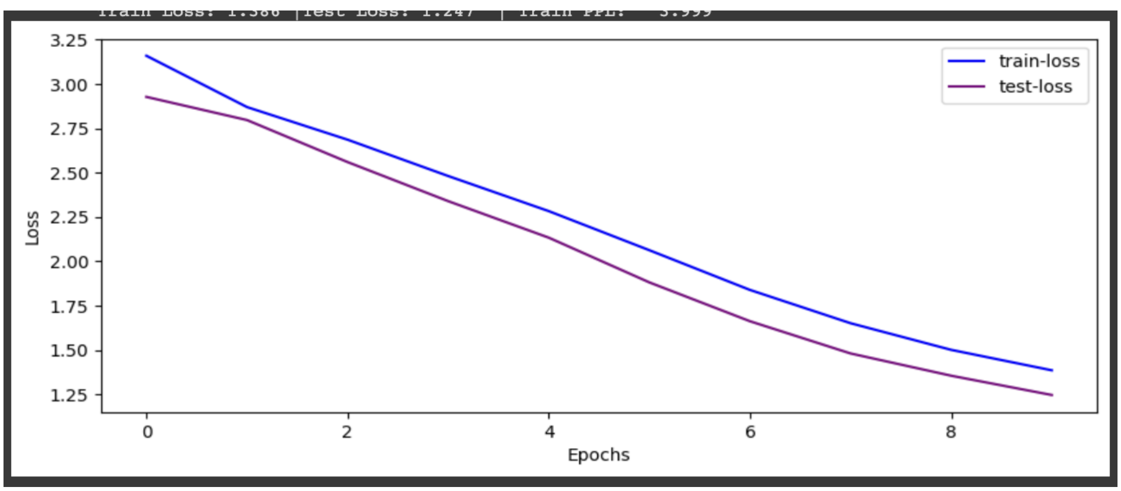


Attention LSTM Combination 2 : (64,3,3,64)

Output of the Train/Test method :

Epoch: 01 | Time: 1m 4s
 Train Loss: 3.159 |Test Loss: 2.927
Epoch: 02 | Time: 1m 3s
 Train Loss: 2.870 |Test Loss: 2.797
Epoch: 03 | Time: 1m 0s
 Train Loss: 2.686 |Test Loss: 2.560
Epoch: 04 | Time: 1m 2s
 Train Loss: 2.481 |Test Loss: 2.340
Epoch: 05 | Time: 1m 1s
 Train Loss: 2.283 |Test Loss: 2.134
Epoch: 06 | Time: 1m 1s
 Train Loss: 2.063 |Test Loss: 1.882
Epoch: 07 | Time: 1m 5s
 Train Loss: 1.839 |Test Loss: 1.663
Epoch: 08 | Time: 1m 1s
 Train Loss: 1.651 |Test Loss: 1.481
Epoch: 09 | Time: 1m 11s
 Train Loss: 1.501 |Test Loss: 1.356
Epoch: 10 | Time: 1m 4s
 Train Loss: 1.386 |Test Loss: 1.247

Plot the graph:-



Observation : In 10 epochs, the lowest loss value observed is 1.9 training loss.

LSTM with Attention Network has given better results compared to the traditional LSTM models. This is because with the use of an attention mechanism, attention-based LSTM models could differentiate between distinct input sequence segments based on how relevant they are. By updating the hidden and cell states, the attention LSTM decides on either highlight or de-emphasize certain parts of the input sequence.

[Colab Link Question 1]

Question 2 : The dataset you have been given is Individual household electric power consumption dataset.
(i)Split the dataset into train and test (80:20) and do the basic preprocessing.
(ii) Use LSTM to predict the global active power while keeping all other important features and predict it for the testing days by training the model and plot the real global active power and predicted global active power for the testing days and comparing the results.
(iii) Now split the dataset in train and test (70:30) and predict the global active power for the testing days and compare the results with part (ii).

Solution 2:-

Steps Followed:-

1. Download the data from the website using 'wget' command and unzipping it.

```
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20640916 (20M) [application/x-httpd-php]
Saving to: 'household_power_consumption.zip'

household_power_con 100%[===================>]  19.68M  9.73MB/s    in 2.0s

2023-04-08 11:12:37 (9.73 MB/s) - 'household_power_consumption.zip' saved [20640916/20640916]

Archive:  household_power_consumption.zip
  inflating: /content/data/household_power_consumption.txt
```

2. Importing the data using pandas and replacing '?' with nan value.

```
length of train dataloader: 6404
length of test dataloader: 1601
```

3. Splitting the dataframe into train and test data with a ratio of 80:20.

datetime	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
2006-12-16 17:24:00	4.216	0.418	234.84	18.4	0.0	1.0	17.0
2006-12-16 17:25:00	5.360	0.436	233.63	23.0	0.0	1.0	16.0
2006-12-16 17:26:00	5.374	0.498	233.29	23.0	0.0	2.0	17.0
2006-12-16 17:27:00	5.388	0.502	233.74	23.0	0.0	1.0	17.0
2006-12-16 17:28:00	3.666	0.528	235.68	15.8	0.0	1.0	17.0

4. Now, splitting the data into input data and labels by copying the Global Active Power column to y-train and dropping the Global Active Power column from input X_train. Perform the same operation with X_test and y_test.

Features in the X_train after dropping the Global_active_power column:-

```
features: Index(['Global_reactive_power', 'Voltage', 'Global_intensity',
       'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3'],
      dtype='object')
target: Global_active_power
```

X_train data without Global_active_power column.

datetime	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
2010-11-02 20:38:00	0.236	244.14	5.2	0.0	1.0	0.0
2010-06-16 15:14:00	0.084	243.73	1.8	0.0	0.0	1.0
2010-04-18 06:00:00	0.098	240.21	1.6	0.0	0.0	0.0
2009-03-06 15:20:00	0.074	244.49	7.0	0.0	0.0	19.0
2007-04-12 00:26:00	0.130	239.10	1.4	0.0	1.0	0.0

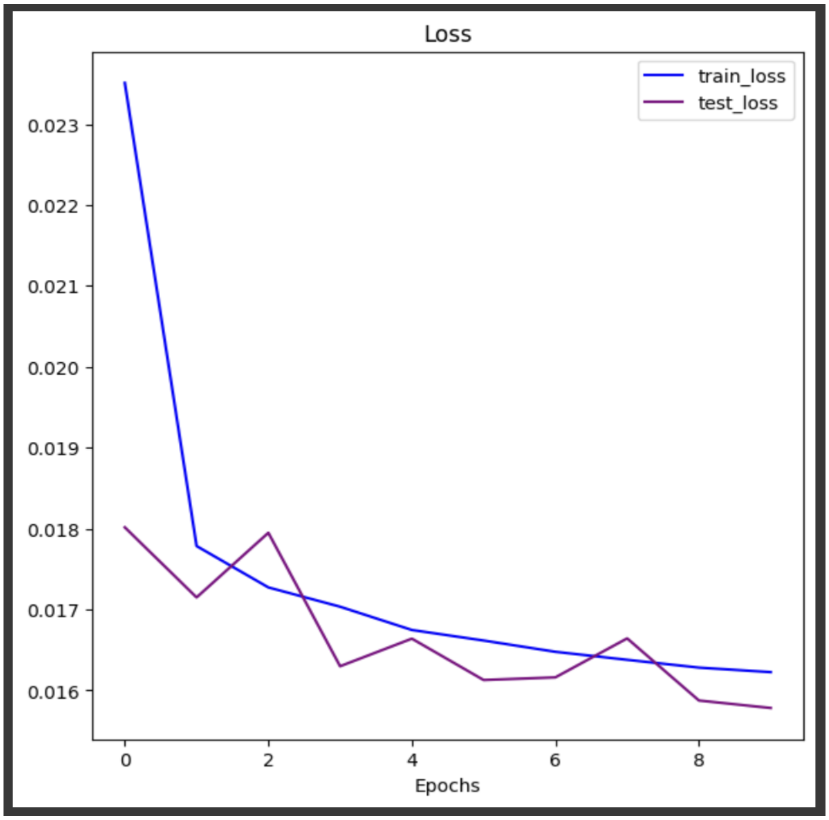
X_train , y_train data

	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	Global_active_power
0	0.931507	0.802564	0.52	0.0	1.0	-0.058824	0.526230
1	-0.109589	0.697436	-0.16	0.0	0.0	0.000000	-0.124590
2	-0.013699	-0.205128	-0.20	0.0	0.0	-0.058824	-0.195082
3	-0.178082	0.892308	0.88	0.0	0.0	1.058824	0.885246
4	0.205479	-0.489744	-0.24	0.0	1.0	-0.058824	-0.242623

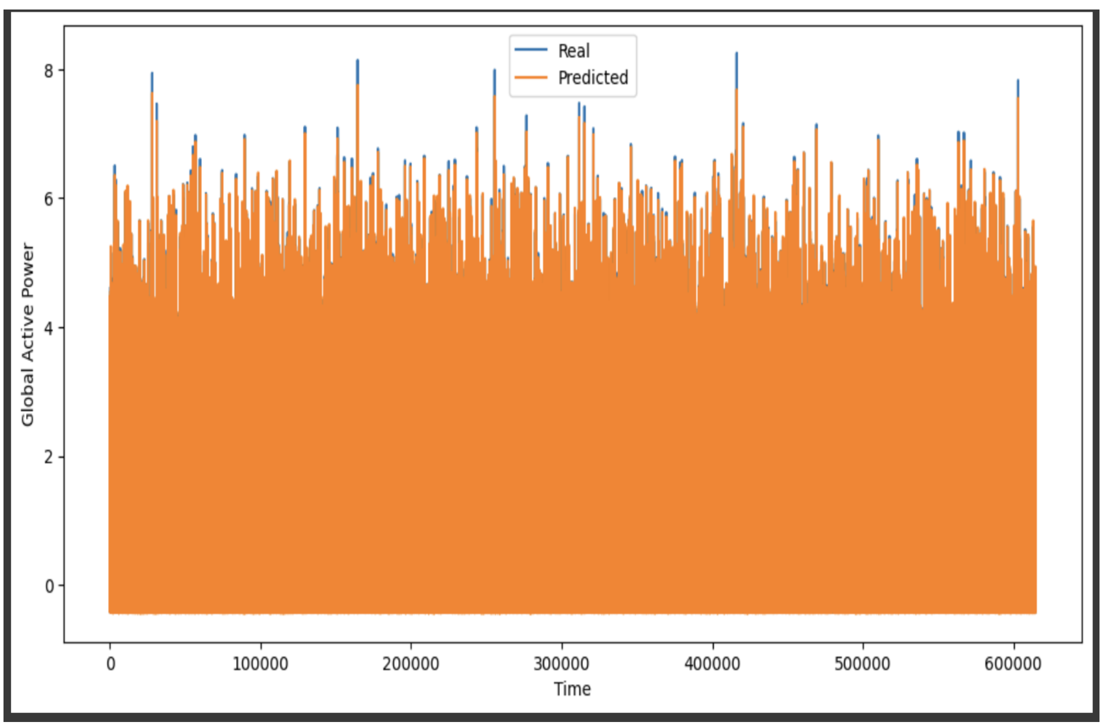
5. Build the traditional LSTM model and train the model. Hyperparameters used are Epochs= 10, Learning Rate = 0.001, Batch Size = 256. Loss Function - nn.L1Loss (Mean Absolute Error).

6. Find predictions and calculate loss and accuracy.
   Results :-
   Epoch: 1 | train_loss: 0.0235 | test_loss: 0.0180 | test_acc: 92.4769
   Epoch: 2 | train_loss: 0.0178 | test_loss: 0.0171 | test_acc: 93.7058
   Epoch: 3 | train_loss: 0.0173 | test_loss: 0.0179 | test_acc: 93.8383
   Epoch: 4 | train_loss: 0.0170 | test_loss: 0.0163 | test_acc: 94.8838
   Epoch: 5 | train_loss: 0.0167 | test_loss: 0.0166 | test_acc: 93.9291
   Epoch: 6 | train_loss: 0.0166 | test_loss: 0.0161 | test_acc: 94.9975
   Epoch: 7 | train_loss: 0.0165 | test_loss: 0.0162 | test_acc: 94.7172
   Epoch: 8 | train_loss: 0.0164 | test_loss: 0.0166 | test_acc: 93.8056
   Epoch: 9 | train_loss: 0.0163 | test_loss: 0.0159 | **test_acc: 94.9272**
   Epoch: 10 | train_loss: 0.0162 | test_loss: 0.0158 | test_acc: 94.7386
   80:20 Ratio Dataset Highest Accuracy : 94.9%

7. Plot the loss graph for Train:Test :: 80:20



8. Comparison between real and predicted values Train:Test :: 80:20

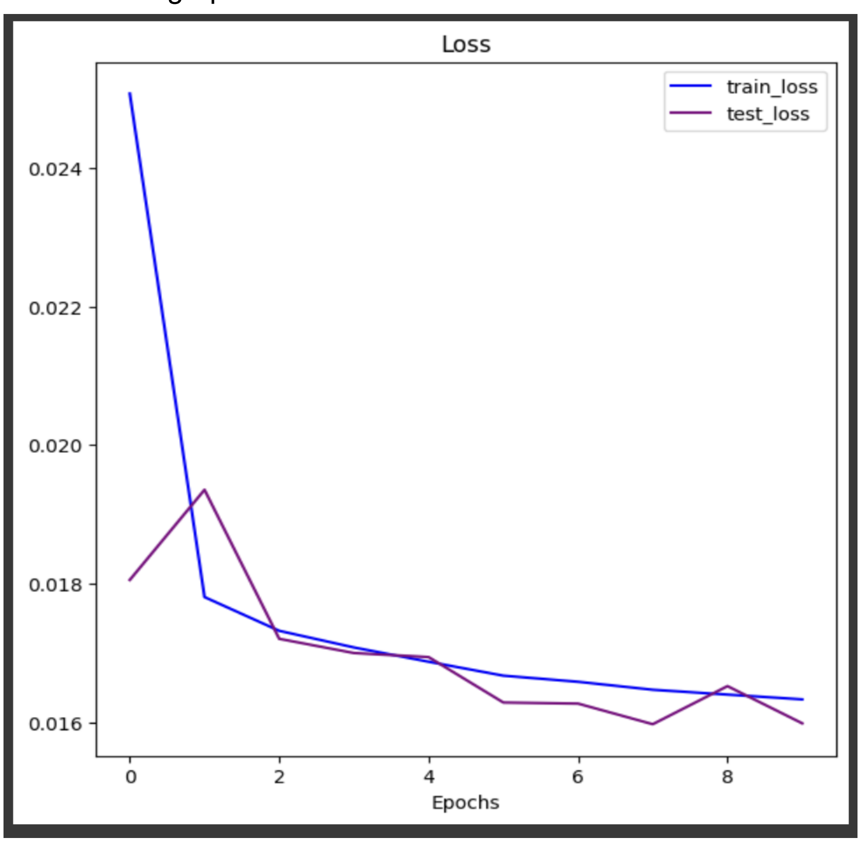


9. Splitting the Dataset in 70:30 Ratio and now the lengths of dataloader has been changed as below:

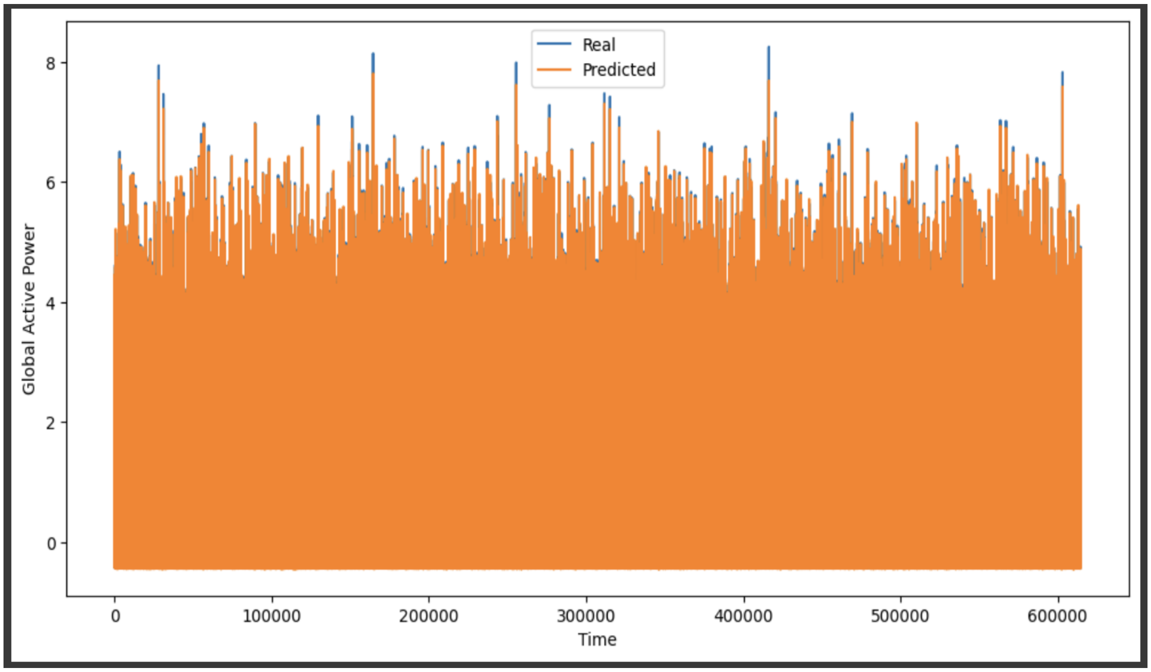
```
Train x_sample shape: torch.Size([256, 25, 6])
length of train dataloader: 5604
length of test dataloader: 2402
```

10. Training the model on this dataset and calculating the loss and accuracy.
    Results :-
    Epoch: 1 | train_loss: 0.0251 | test_loss: 0.0181 | test_acc: 92.4068
    Epoch: 2 | train_loss: 0.0178 | test_loss: 0.0194 | test_acc: 91.2462
    Epoch: 3 | train_loss: 0.0173 | test_loss: 0.0172 | test_acc: 93.4105
    Epoch: 4 | train_loss: 0.0171 | test_loss: 0.0170 | test_acc: 93.5468
    Epoch: 5 | train_loss: 0.0169 | test_loss: 0.0169 | test_acc: 93.2942
    Epoch: 6 | train_loss: 0.0167 | test_loss: 0.0163 | test_acc: 94.6246
    Epoch: 7 | train_loss: 0.0166 | test_loss: 0.0163 | test_acc: 94.7541
    Epoch: 8 | train_loss: 0.0165 | test_loss: 0.0160 | test_acc: 93.0566
    Epoch: 9 | train_loss: 0.0164 | test_loss: 0.0165 | test_acc: 93.6643
    Epoch: 10 | train_loss: 0.0163 | test_loss: 0.0160 | test_acc: 94.7987
    70:30 Ratio Dataset Highest Accuracy : 94.7%

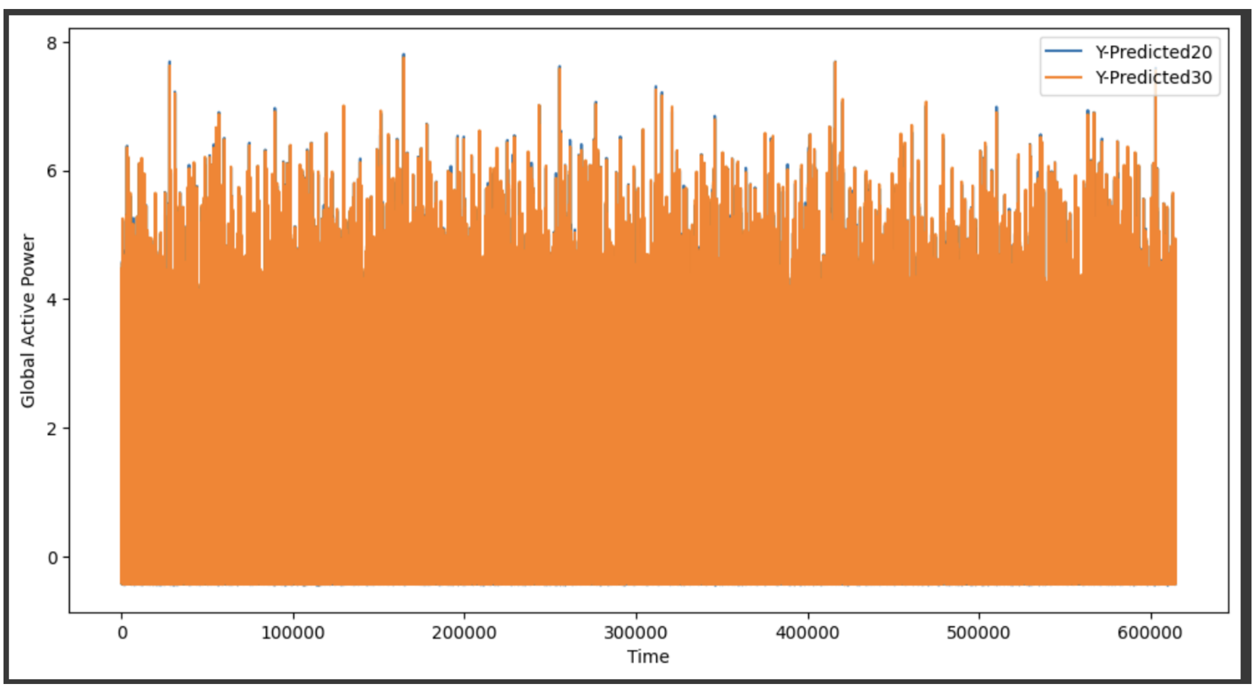
11. Plot the loss graph for Train:Test :: 70:30



12. Comparison between real and predicted values Train:Test :: 70:30



13. Comparison between predicted values when Train:Test split is 80:20 vs 70:30:-



Observation : Though there is not much difference between the values predicted in both the cases, we can say that the accuracy is better in the dataset with Train:Test split ratio 80:20 because we have more training data points available in comparison to the later case.

[**Colab Link Question 2**]

References:-

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html
https://www.crosstab.io/articles/time-series-pytorch-lstm/
https://github.com/UtkarshGarg-UG/Deep-Learning-Projects/tree/main/NLP/Custom%20Dataset
https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html
https://www.kaggle.com/code/tartakovsky/pytorch-lightning-lstm-timeseries-clean-code
https://github.com/bentrevett/pytorch-seq2seq