EE5179 : Deep Learning for Imaging

Programming Assignment 3: Recurrent Neural Networks

Due by: October 9th, 2022

Instructions

- Program in python for this assignment.
- Post any doubts you have on moodle. This will be helpful to your peers as well.
- Submit the codes (the actual notebook and a PDF version) and your assignment report in a zip file titled PA3_RollNumber.zip in the submission link provided on moodle.

Preliminaries

- It is recommended to use Google Colab (as in the tutorials) or Jupyter/iPython notebooks for the assignment. The notebook format is very convenient to work (and evaluate!) with and additionally, Colab provides GPU access as well. Click here to learn more about how GPUs can be accessed using Colab.
- The dataset can be downloaded from here or you can make use of the inbuilt dataset from Pytorch.
  Note: Check if the labels are in one-hot format, or appropriately convert them to one-hot format before training and testing the network.

---

1 MNIST classification using RNN

Build a simple network to classify MNIST data using the following architecture

- Input to the network is a digit image (28×28) which will be converted to a sequence in order to enable usage of RNN, which is a many to one network. Each image will be a sequence of 28 vectors each of dimension 28 (these can be columns or rows of the image). Hence, the RNN has to be unrolled for 28 steps.
- Each input vector is fed to the first hidden layer of a RNN in a fully connected manner. The output unit consists of 10 units for each digit. Let the output activation be linear. Since you want to do classification use soft-max to get probabilities of digit belonging to 10 classes. can start with a network with hidden state size (state vector size) of 128. You are encouraged to tune it to get better performance. You can make use of optimizers like adam.
- Divide the whole data into training (50000 images), validation (10000 images) and test (10000 images) sets. It has standard train(60,000) and test samples(10,000). You have to use them accordingly for training and testing.
- You can conduct the following experiments seperately
  - Using a network with vanilla RNN cells
  - Using a network with LSTM/GRU cells instead of vanilla RNN cells
  - Using a bidirectional RNN/LSTM network
- You would have to go through related literature and conduct experiments to decide appropriate values of hyper-parameters like the number of such hidden layers and number of neurons in each hidden layer.
- For each of the experiments above, you are required to show the plot of training error, validation error and prediction accuracy over the training progress. For each of the experiments, at the end of training report the average prediction accuracy for the whole test set of 10000 images.
- Provide sample inputs from the test data and predictions made by your best model

- check with at least 10 custom input images from your own handwriting and predictions made by your best model.Make sure to pre process the custom input format of the image. It should be same as the format of mnist data input (28X28 resolution).
- please comment your observations of above predictions of custom inputs

2 Remembering the number at a particular index in a given sequence

This exercise pertains to the capability of RNNs to handle variable length sequences. Here, you need to train a recurrent network to always remember the number appearing in a particular position in any given sequence of integers. Below examples show few cases where the number to be remembered is at position 2

$$6,9,8,1,8,3,4 \rightarrow 9 \tag{1}$$

$$1,2,3,4 \rightarrow 2 \tag{2}$$

$$9,7,2,3,4,5,6,1,1,1 \rightarrow 7 \tag{3}$$

1. You will need to create a function that can create a random input-output pair . Write a function that takes a sequence length L as input and returns a training sample to be fed to the RNN. For a given length K, a training sample is a 2-tuple of input-x and output-y where input is a tensor of size L×10: Second dimension is 10 since each integer is represented as a one-hot vector of dimension=10 (since we consider only sequence of integers in [0-9]). Output is just a single number . Its the number at the Kth position which need to be remembered. So y is a tensor with just one element of size 10
2. The RNN model design is not too different from the one you used for the mnist classification exercise above. The input vectors are fed one-by-one, starting from the first, proceeding all the way to the last vectors. At each time step the input to the network is of dimension=10. At the last time step, the model should output the number at position k. For this purpose we have an output layer of 10 neurons. Each corresponding to integers 0-9. So if the kth integer is say 7, then the 7th neuron should fire the most. Or in other words output of this node should have the maximum value . The only hyper parameter for the model is the size (dimensionality) of the hidden state. Bigger state size implies higher capacity.
3. You need to train your model on sequencess of different lengths lying L in range of 3 and 10. For this exercise, you can make your training code run for a fxed number of epochs (or iterations) eg. 20. Although in practice, the training should be monitored with performance on a held-out or validation set, in order to avoid overfitting. Note that you can train a RNN model for only a particular value of the position. Hence, during test time, the position for remembering will be fixed, but the input sequence length can vary. Since the task is not too dificult to learn and the suggested data and the model are small, you can train it on the CPU itself (GPUs are not required).
4. (a) Train 3 different RNN models: with hidden state size 2, 5, 10. For each of the experiments, you are required to show the plot of training error and prediction accuracy over the training progress. For each of the experiments, at the end of training, report the average prediction accuracy for the test set you created.
   (b) For your best model, you need to submit the results of testing the model for various input sequence lengths. For each length, print a few input sample sequence and show the network output and the number at the kth position.

3 Adding two binary strings

İn this experiment, we will explore the simple problem of teaching an RNN to add binary strings. Recall, like grade-school addition, binary addition moves from the right-most bit (least- significant bit or LSB) towards the left-most bit (most-significant) bit, with a carry bit passed from the previous addition. Table-1 shows the truth table for a full-adder.

i_1	i_2	Carry-in	Sum	Carry-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1: Truth-table for a full adder

Time	i_1	i_2	Output
1	0	1	1
2	1	1	0
3	0	0	1

Table 2: Sample sequence during training

The RNN is fed two input bit-sequences and the target "sum" sequence. The sequence is ordered from LSB to MSB, i.e., time-step 1(t = 1) corresponds to LSB, and the last time-step is the MSB.

For example, if the bit strings 010 (integer value = 2) and 011 (integer value = 3) are to be added to produce the sum 101 (integer value 5), the Table 2 is the sequence of inputs and targets fed to the RNN when training. Note that we are not providing the carry-bit explicitly as the input and the RNN is expected to learn the concept of carry-bit.

3.1 Data preparation

To train the model, we need a batch size of two binary strings of a given length L (say 5) and the corresponding output as training data. Be cautious that the output string can have one extra bit and make sure you pad your input strings with 0's appropriately. You can either generate the whole dataset before hand and store it in a file or you can generate it on the go during training.

3.2 Model

Your model should take two inputs at every time-step starting from the LSB to MSB in the generated data. State vector can be of size say 5 and the output is single value, either 0 or 1. To constraint the output between 0 and 1, use sigmoid function. Use LSTM as your RNN and the initial state vector as all zero.

3.3 Experiments

1. Vary the state vector size and check if there is any improvement in bit-accuracy.
2. Compare the accuracies on using MSE and Cross-Entropy loss functions separately.
3. Train only on a fixed length inputs say L = 3, L = 5, L = 10 and check the bit-accuracies on different length. What do you observe?

By bit-accuracy, we mean the number of bits predicted correctly by the network. You can keep the threshold as 0.5 and any value above it can be considered 1 and any value below it can be considered as 0.

3.4 Submissions

1. You will have to fix the number of epochs, batch size and other hyper-parameters like learning rate etc., for the training process.

2. For each experiment above, you are required to show the plot of training loss and test loss as the training progresses.

3. For each of the experiments, at the end of training, report the average bit-accuracies on say 100 samples each for input string length (L) in the range 1 to 20 and show the plot with the average bit-accuracy on yaxis with length L on xaxis.

–end–