Programming Assignment-3: Recurrent Neural Networks

Kuldeep Purohit, ee14s007@ee.iitm.ac.in
Praveen Kandula, praveen.kandula94@gmail.com

Due date: October 13th, Sunday, 11:55pm

Note:

1. For any questions, please schedule a time with TAs before deadline according to their convenience. Please use moodle discussion threads for posting your doubts and also check it before mailing to TAs, if the same question has been asked earlier.
2. Submit a single zip file in the moodle named as PA3_Rollno.zip containing report and folders containing corresponding codes.
3. Read the problem fully to understand the whole procedure.
4. Late submissions will be evaluated for reduced marks and for each day after the deadline we will reduce the weight-age by 10%.

An RNN is a function that applies the same transformation (known as the RNN cell or step) to every element of a sequence. The output of an RNN layer is the output of the RNN cell applied on each element of the sequence. In the case of text, these are usually successive words or characters. In addition to this, RNN cells hold an internal memory that summarize the history of the sequence it has seen so far.

1 MNIST digit classification using RNN

In this experiment, we will do classification on MNIST digits dataset using a Recurrent Neural Network and its variants. The approach will be deliberately simple i.e. reading the image row by row and treating that as one long sequence rather than a single entity which is more suited to networks using convolutions.

1.1 Model

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 $\mathbf{x_i}$ 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. This will be a 10 dimensional vector $\hat{\mathbf{y}}_\mathbf{i}$. Now train the network using cross entropy loss between $\hat{\mathbf{y}}_\mathbf{i}$ and $\mathbf{y_i}$, where $\mathbf{y_i}$ is the one hot representation of ground truth label. You 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 use Adam optimizer.

1.2 Data preparation

You can make use of the binary files that you downloaded for the first assignment. You can also make use of the python library sklearn and download the MNIST data using sklearn:datasets. If using sklearn then make sure that you shuffle the data properly and divide the whole data into training (50000 images), validation (10000 images) and test (10000

images) sets. To load the MNIST data, follow links like this [1] in matlab or this [2] in python. It has standard train(60,000) and test samples(10,000). You have to use them accordingly for training and testing.

1.3 Experiments

1. Using a network with vanilla RNN cells
2. Using a network with LSTM/GRU cells instead of vanilla RNN cells
3. Using a bidirectional RNN/LSTM network

1.4 Submissions

1. 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.
2. 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. You should also plot randomly selected test images showing the true class label as well as predicted class label. Use standard training techniques as described in the previous assignment. Use a l2 regularization over only the weights of the neural network. and submit the training plots and the final test accuracy. Compare the two convergences.

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.

[1] http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset

[2] https://github.com/mnielsen/neural-networks-and-deep-learning/blob/master/src/mnist_loader.py

Below examples show few cases where the number to be remembered is at position 2

$6, 9, 8, 1, 8, 3, 4 \rightarrow 9$

$1, 2, 3, 4 \rightarrow 2$

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

2.1 Dataset

You will need to create a function that can create a rndom input-output pair on the fly. 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 $L \times 10$: 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

2.2 Model

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 timestep the input to the network is of dimension=10.

At the last timestep, 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 k^{th} 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 hyperparameter for the model is the size (dimensionality) of the hidden state. Bigger state size implies higher capacity.

2.3 Training

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 fixed 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 over-fitting.

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 difficult to learn and the suggested data and the model are small, you can train it on the CPU itself (GPUs are not required).

2.4 Submissions

1. 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.

2. 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 k^{th} position.

3 Adding two binary strings

In 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.
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.

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

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 $y - axis$ with length L on $x - axis$.