Albert Lee and Tom Wu

Neural Turing Machine Assignment

The goal of this assignment is to help you get a more concrete understanding of the NTM architecture, as well as the individual roles different components of the NTM play in its learning performance. We will focus specifically on the copy task described as the first example in the paper.

Some general comments on the architecture and training implemented in the provided code:

- For the copy task, the NTM is supposed to copy a sequence of *seq_length* units, each of which is a vector of 8 dimensions. The input to the NTM is a sequence of *seq_length * 2 + 1* vectors, where the first *seq_length* vectors are the sequence it is suppose to copy, the proceeding vector is the delimiter, and the last *seq_length* vectors are empty. The expected output of the NTM is also a sequence of the *seq_length * 2 + 1* vectors, but with the last *seq_length* vectors the sequence to copy and all other vectors are 0.
- The controller is a simple network with one hidden layer between the inputs and outputs.
- The heads can be thought of as another layer following the controller's hidden layer. They output the key k, the key strength β, the interpolation gate g, the shift weighting s, and the sharpening factor γ, as well as the erase and add vectors e and a used for writing to memory.
- The NTM is currently trained with curriculum learning (shorter inputs are presented first before longer ones, gradually increasing the "difficulty" of copying tasks the NTM has to learn). The input sequence to be copied has an initial length of 1, and gradually increases to 20 during training. During testing, it will be asked to generalize the copy task for sequence lengths far beyond 20.
- The NTM can be trained for the copy task by running **"python train_copy.py <filename>"** where <filename> is the name of the file to save the trained model to.
- Plots can be generated by running **"python test.py <filename> <seq_length>"** where <filename> is the model file and <seq_length> is the length of the test sequence that will be randomly generated.
- The average error rate of the output sequence produced by the NTM is calculated as the average of the absolute difference between each data point in the expected output sequence and the actual output sequence.

Problems:

1. Fill in the code for `make_model` in `run_model.py` to evaluate the NTM for testing.

2a. NTM Addressing Mechanism

i. Implement content-based addressing (Hint: use the provided `cosine_sim` function as the similarity measure)

ii. Implement the interpolation step used for location-based addressing.

iii. Implement the convolutional shift step used for the next step in location-based addressing (Hint: use the provided `shift_convolve` function)

iv. Run the NTM for the copy task for sequences of length 10, 20, 50, and 120 and report your average error rates and include the generated plots. Comment on the general performance of the NTM for different length sequences.

b. Implement the sharpening step used in the final step of location-based addressing. Run the NTM for the copy task for the same length sequences above and compare your average error rates and plots with Part 2aiv. What effect does sharpening have on the NTM's results?

3a. What role does content-based addressing play specifically for the copy task? Can you think of another simpler way to capture this functionality for the copy task without a learned weight vector?

b. Implement your alternative to content-based addressing, run the NTM for the copy task for sequences of length 10, 20, 50, and 120 and compare your results with Part 2b.

c. Try running the NTM for the copy task for a sequence of length 150. What do you observe? What are possible explanations for these results?

4. Try experimenting with variations in:

- the controller architecture (number of hidden units, number of hidden layers)
- the cost function
- the update rule
- different training methods (e.g. curriculum vs. non-curriculum learning or mixture of both, see http://arxiv.org/abs/1410.4615 for ideas, or changing the maximum input sequence length the NTM is trained on and commenting on generalization performance)
- the memory size (if you increase the memory size, test on longer sequences and comment on your generalization performance)
- any other ideas you can come up with

Try at least two of the above and compare and analyze your results.