SFNN Status Update

So far the SFNN project has proved to be somewhat challenging due to the lack of detail specified in the Tang paper. After meticulously debugging the algorithm it’s clear that there are problems but it is difficult to say what they are. I get a sense that we are not following exactly what the paper has explained yet it is difficult to tell because it is written ambiguously. I believe one thing we really need to focus on is the importance sampling. These equations are giving me the most confusion and what area I’m least confident in for our implementation.

Up to this point I have fully implemented and tested the biases and the learning rate. I am currently working on mini-batching and restructuring the code for testing purposes. The testing file will continue to grow as we clear up some confusion about the algorithm and continue to implement neural network methods to enhance accuracy. Due to the lengthiness and laborious processes of debugging I feel very behind schedule. Going through 16x30 and 16x16 matrices trying to do calculations with other matrices is very slow even with the help of a computer.

I think looking at the code we have, we are close to how a natural SFNN would be implemented. However, they claim a novel Monte Carlo Generalized Expectation Maximization algorithm for learning. This novelty is my hypothesis of why we cannot replicate their results. There are also some questions about variance and normalization where it is unclear if the authors want to sample from a normal distribution or saying that the results are already normalized. The variance is the main reason why I believe our implementation is incorrect. The paper claims that the variance should be small however, in our implementation, the variance seems to be extremely small such that the distances away from one another are almost non-existent.

SFNN RoadMap

I believe our first step is figuring out what is wrong with our current implementation. We should be much closer than we are and simple accuracy enhancement tools are not going to make the difference we need. For this step I believe it would be beneficial to work through this with someone else. If we can logically work out the equations in the paper, then hopefully we can produce them in code form. Also, since a lot of this project revolves around working with large matrices it very quickly becomes a couple day task to run through manual error checking. I think having a partner on this section would help the process flow much more smoothly and efficiently.

Once we have confirmed the correctness of our implementation, the next step will be adding to the test file a concrete demonstration of the properly working algorithm. This will be essential for error checking in the future. Although I have already started this process, it is difficult to continue due to my lack of confidence in the correctness of our implementation. Testing can be bias and it is easy to create test cases which are incorrect, yet give the impression of correctness to the algorithm.

The concrete testing platform will allow us to move on to implementing any methods for increasing accuracy. These methods would include a more substantial learning rate adaptation as well as mini-batching. Lastly, we should be able to verify the validity of our implementation via graphical images Tang’s paper as well as answer and lingering questions Marty posted on Github. Once the implementation is completed and fully tested we could potentially begin changing parameters in an attempt to get better results. Once we experimentally find this optimal parameterization we can begin testing with abstracted vectors for source code in an attempt to find refactoring opportunities and potential replacements.

Roadmap Simplified

Look at Tang paper in extreme detail to answer some ambiguous questions

↓

Systematically go through Implementation to determine where we differentiate from the Tang implementation

↓

Complete concrete testing suite to ensure the correctness of the implementation and use that testing suite as a baseline for when we add new methods

↓

Add additional methods to our implementation to increase the accuracy of our results (mini-batching, learning adaptation, parameterization)

↓

Add additional test methods to ensure we didn’t not break our implementation by adding new methods

↓

Compare our results with Tang’s results on the same data to make sure we are relatively close.

↓

Begin testing implementation with abstracted vectors representing source code

↓

Gather results and finish writing paper for submission