COMP480 Term Project Checkpoint2 Report

Edward Feng (jf44), Xinhao Liu (xl59)

As previously described in our checkpoint one progress report and our initial project proposal, our goal was to use machine learning to create adversarial traffic such that we can cause a DDoS attack for a server that uses a hash table to check for malicious traffics. The idea is that the server has a hash table that is keeping track of all of the malicious traffics, and whenever the traffic sent to the server causes a cache hit, it will take the server a longer period of time than normal to get the respond since it needs to check whether it is really malicious. Thus, our goal now becomes to use machine learning to generate traffic such that the traffic will cause a cache hit on server's hash table.

More generally, the real problem here is: can machine learning learn the pattern of a hash function? To find out the answer, we did a simple simulation using a hash function and trained a classifier with feed forward neural network.

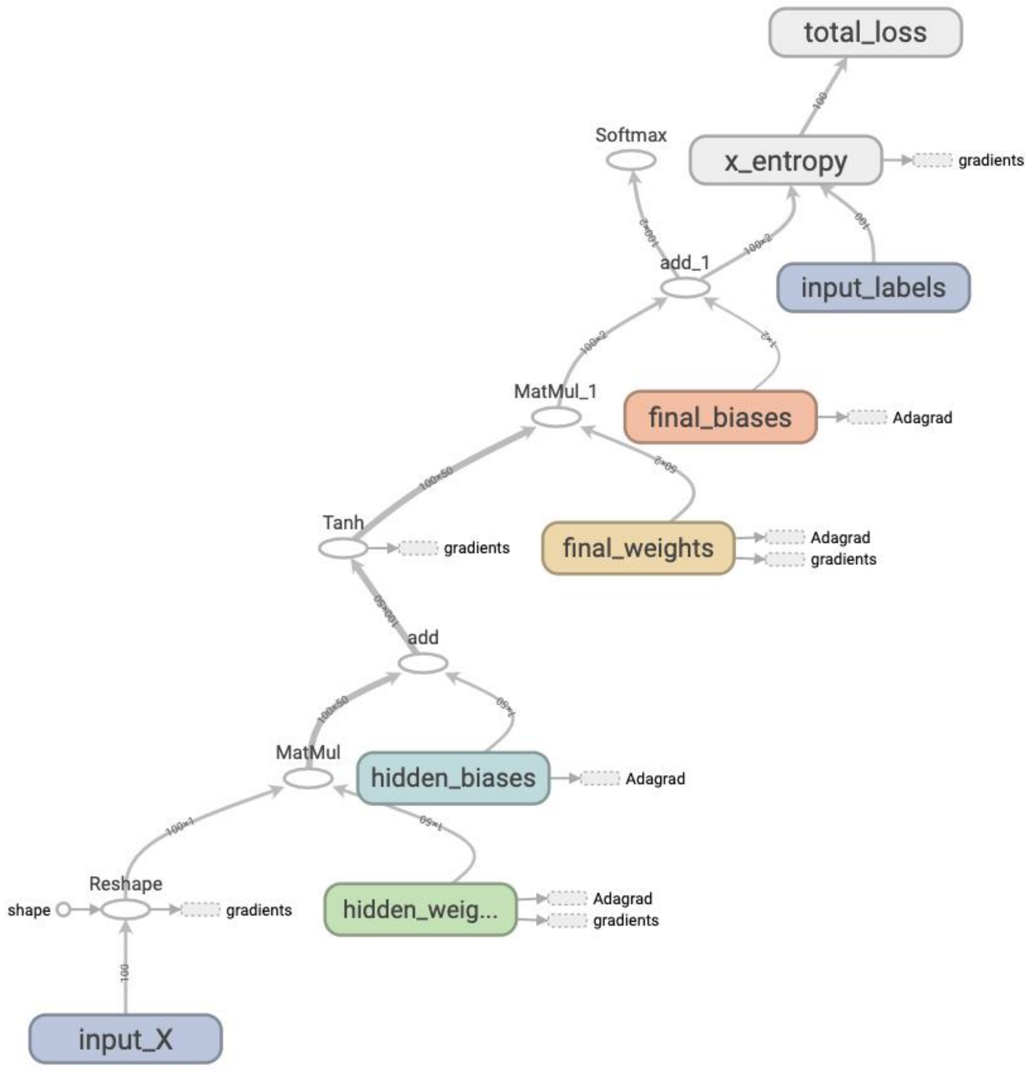
Background and setup:

- To simplify the problem at this point, we used integers to simulate the network traffic being sent through the network. Each individual data packet is represented by one integer.
- For this experiment, we used a simple hash function that is h(x) = ((37x + 47) % 2038072819) % TABLE_SIZE, where TABLE_SIZE = 2500.
- To load the table with caches, I randomly sampled 1625 = (0.65 * 2500) integers from 0 to 5 * TABLE_SIZE and inserted them into the hash table. Due to hash collision, there are approximately 1200 buckets that are marked as 1 after the insertion.
- We used a similar technique to generate training and testing data. Specifically, we randomly sampled TABLE_SIZE * 2 integers from 0 to 3 * TABLE_SIZE and labeled them 1 or 0 based on whether they cause a cache hit or cache miss respectively. After removing duplicates, there are approximately the same amount of cache hit data and cache miss data (around 1800 each). Then I took the first 80% of the data as training data and the rest for testing.

Machine Learning Model:

- The basic model architecture we used is a simple feed forward neural network with one input node, one hidden layer of 50 nodes and 2 output nodes.
- For the hidden layer, we used the tanh activation function to calculate the activated value for each hidden node.
- We calculated the loss by calculating the cross entropy and we measure the prediction with a softmax layer at the end
- Then we used the adagrad optimizer provided by TensorFlow which automatically adapts the learning rate to the parameters, we set the initial learning rate to 0.01

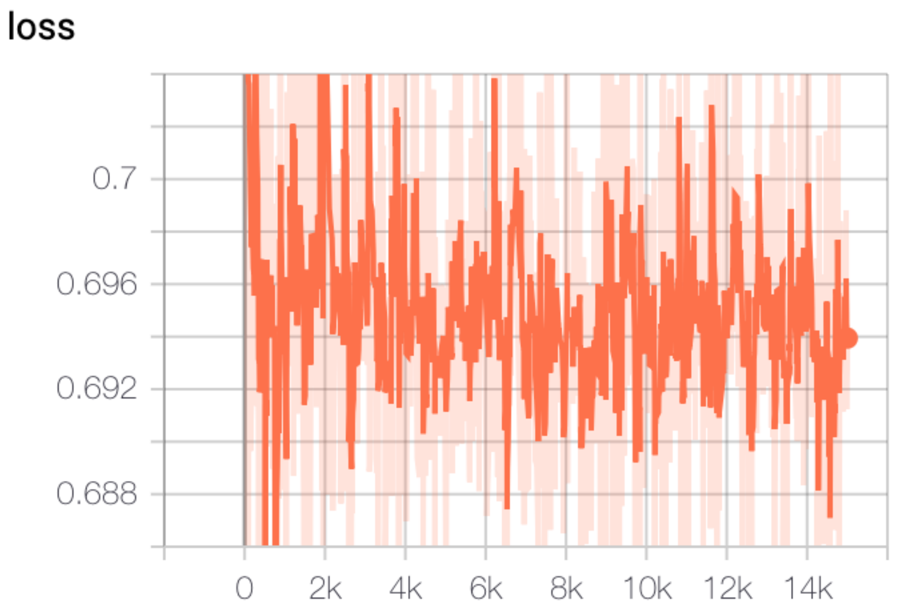
- We ran 15000 training iterations with 100 data as a batch for each iteration, each batch is randomly drawn from the whole training dataset
- Here's the visualization of the model architecture (from TensorBoard):



Current results and observations:

- For now, as we ran 15000 iterations, the total loss in each iteration doesn't vary much and there's no obvious tendency of loss getting lower along number of iterations.
- The loss is pretty stable and fluctuate at around 0.7.
- The number of correct labels for each batch is fluctuating at around 50-55 out of 100.
- For the 735 testing data, number of correct labels is around 350 for different runs.

- Here's a graph visualization of loss vs number of iterations from TensorBoard:



  As we can see the loss doesn't show a decreasing tendency.

Summary:

- For the current experiment, we can basically claim that the machine learning model didn't really learn the pattern of our hash function.
- What we expect to do for final report:
  - Tune the model using various combinations of parameters to see if the pattern can be learned.
  - Try different model architectures since learning the hash function is not an easy task and a simple one-hidden-layer feed forward neural network is probably not powerful enough.
  - Increase the amount of randomly generated training data, since the overall quality of the machine learning model largely depends on the amount of training data provided during the training step.