# CS 330 Autumn 2020 Homework 1

SUNet ID: 06009508

Name: Brandon McKinzie

Collaborators: N/A

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

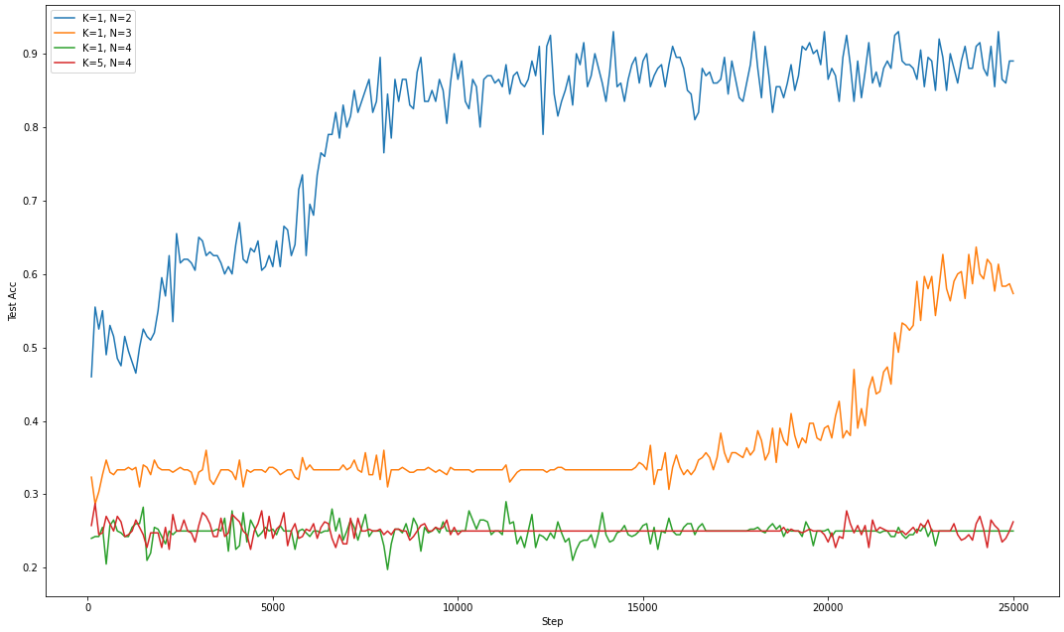
---

[LINK TO COLAB NOTEBOOK]

Raw URL: https://colab.research.google.com/drive/1uhEGHZrJ2x9HZyph0KQTeNYp2V0lPEMT?usp=sharing

## Problem 3: Analysis

Below is my plot of results for each requested setting of N and K.




Observations:

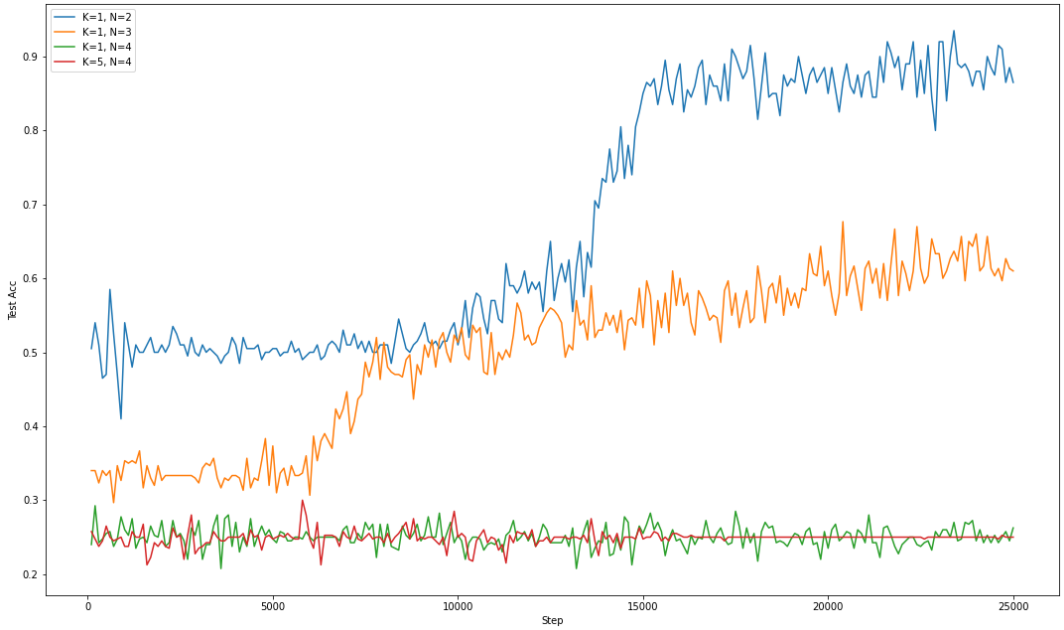
- The model was able to achieve about 90 percent test accuracy for the simplest case (K=1 and N=2). This was expected to be the easiest trial since it only had to distinguish between two classes.
- For $K = 1, N = 3$, the model was able to reach about 66 percent accuracy near the final iterations, meaning it was able to correctly distinguish two out of three classes on average for the test examples. Given the performance boost near the end, it would be interesting to see whether training for more iterations could improve test accuracy even more (did not have time to verify unfortunately).
- The model did no better than random chance for the two trials with $N = 4$. Considering

that I verified the contents returned by my `sample_batch` function were as expected and the model call function is relatively simple, my main hypothesis for this result is that I did not shuffle the order of meta training examples (label-wise). Specifically, my implementation randomized the order labels sampled, but then this order was just repeated for each shot (note that the order of my meta-test examples were independently shuffled to prevent the model from just memorizing the label ordering). I also thought it may have been due to the relatively small model capacity, but increasing the state size didn't seem to improve much (albeit I only tried it for a few thousand iterations and moved on).

- The model seems to struggle greatly as $N$ increases, and even moreso when $K$ increases. Although, conceptually, increasing $K$ should make it easier for a model to learn the task, it also requires that we increase the model capacity. This is because, as we increase $K$, we are asking the model to cram more and more information into its hidden state, which is a fixed size. Intuitively, we should be able to at least improve training accuracy by increasing the LSTM state size beyond 128.

## Problem 4: Experimentation

The main tweak I explored was changing the first LSTM to be bidirectional. I assumed bidirectionality would put less stress on the model in regards to preserving information from the initial meta training examples, especially considering the relatively small state size of 128. Furthermore, the task isn't inherently temporal – the ordering of the training examples is arbitrary. For this reason, I think it would make far more sense to switch to something like a transformer architecture that operates on sets, without the inductive bias imposed by a unidirectional recurrent architecture. Below are the results I observed by just wrapping tf.keras.layers.Bidirectional around the first LSTM:




Unfortunately, these results don't appear substantially different from the previous trials. The only notable qualitative difference is that the N=3 case improved more steadily than before, although this could've been an artifact of the random seed we're using too. The N=2 case seemed to actually have a harder time than before, but was able to still converge to a similar test accuracy.

Although my implementation is correct as far as I can tell, I'm still not sure why the N=4 case is unable to improve.