1. Abstract
   1. Barcodes in memory allow for search and recall, but what if the barcodes are very close?
   2. In an ideal world, distinctions can always be made, but there cannot be an assumption of a lack of noise being introduced at some point in the process
   3. By transforming the barcodes into a different space via an intermediate model, we are able to show resilience to noise, while maintaining better performance than previous model attempts (specifically Ritter et al. 2018)
2. Background
   1. Ritter Summary
      1. Based on L2RL (Wang et al. 2016)
      2. Inputs are passed into a single cell LSTM and get transformed into a hidden state, which is passed into an A2C policy maker to choose the next action for input into the LSTM
      3. At the end of an episode, a copy of the cell state for the LSTM is stored in a memory buffer, referenced by a key. In our chosen task, this key would be the barcode for that specific episode.
      4. Every input is now also matched against the key in memory, and the prior LSTM cell state found by a 1NN match over the keys is reintegrated via the R-Gate proposed by Ritter.
3. Methods
   1. Task Definition
      1. Contextual Bandits with optional initial clustering of barcodes
         1. X number of arms, with a reward probability of 0.9 for one arm, and 0.1 for all others
         2. The high probability arm is randomly chosen per barcode, with a guarantee that at least one barcode is assigned to every possible arm
      2. Using Hamming distance as a clustering metric, we are able to artificially clump barcodes together and assign/shuffle arm assignments to these clusters over the course of training and evaluation
         1. Intracluster Hamming distance would be less than some chosen threshold
         2. Intercluster Hamming distance would be greater than (barcode\_size – 2\* hamming\_threshold)
         3. Example: 3 arms, 6 distinct barcodes, each barcode is a 16-dimensional binary vector, and we choose a Hamming threshold of 3.
            1. We would create 2 clusters of 3 barcodes (mapping arms 1, 2, and 3 to a barcode in each cluster)
            2. The differences between the barcodes of a single cluster would all be at most 3
            3. The difference between the centers of the clusters would be at least (16-2\*3) 10
            4. This ensures even under noise, it is improbable for a barcode in cluster 1 to be mistaken for a barcode in cluster 2, but there is a high chance for a cluster 1 barcode to be misinterpreted as another cluster 1 barcode
            5. The misinterpretation of the barcode would lead to the agent choosing to pull the wrong arm as the next action.
      3. We define an episode as a series of 10 pulls, with rewards generated by the ground truth (unnoised) barcode.
      4. An epoch is a randomized run through all of the N barcodes, with repetition of each barcode N times.
         1. If there are 6 distinct barcodes, we would have 36 episodes, where each individual barcode is presented 6 times.
      5. The mapping of arms to barcode is shuffled at the end of every epoch, with clusters maintaining their unique arm choices
         1. Assume A, B, C were a single cluster, and D, E, F were the other cluster.
         2. If the first epoch had the following mapping: {A:1, B:2, C:3, D:1, E:2, F:3} then a reshuffle would possibly be {A:2, B:3, C:1, D:3, E:1, F:2}
         3. We don’t allow the following reshuffle: {A:1, B:1, C:3, D:2, E:2, F:3} since that makes cluster 1 have 2 instances of arm 1, and cluster 2 have 2 instances of arm 2
   2. Noise Definition
      1. At the beginning of an episode, we choose N bits from the barcode and randomly decide to flip or keep those bits the same. The rest of the barcode is unchanged. This applied noise is held constant over the course of the episode
      2. We evaluated over 4 different levels of noise, specifically 25%, 50%, 75%, and 87.5%
         1. For a 16-dim barcode, this would correspond to random flips on 4, 8, 12, or 14 of the bits of the barcode.
   3. Supervised Learning
      1. Model takes the hidden state from the LSTM and passes it through a single dense layer to produce a larger embedding.
      2. Embedding is then passed through another single dense layer which is softmaxed and used to predict the barcode for that episode
      3. Model is trained with cross-entropy loss against the true barcodes from the input
      4. The embedding is taken and stored in memory as a key, with the stored value being the ending cell state of the LSTM (the state after all 10 pulls of the episode have been completed)
      5. Outside of this change in memory key (from the original barcode to our new embedding) we make no other substantive changes to the Ritter model or the reward processing
4. Results
   1. Our model is more resilient to noise than Ritter when barcodes are clustered, and approximately equivalent when barcodes are more randomly distributed.
   2. Due to inaccuracy in the embedding model, our overall performance in non-noisy situations for higher numbers of barcodes can be less than Ritter, but that same inaccuracy gives us more favorable results during noisy evaluations
   3. Results Labeling Shorthand (#arms, #barcodes, #pulls per episode)
      1. 4a8b10p
         1. 4 arms
         2. 8 unique barcodes split into two clusters
         3. 10 pulls per episode
   4. Clustered Results
      1. We don’t beat ritter on higher unique barcode counts during training, but do beat them on the same number of distinct barcodes once noise is introduced
         1. See 4a8b10p 1 Hamming Results graph
         2. Need to run a 4a12b10p 1 Hamming to see if that causes any problems
      2. We beat Ritter on lower unique barcode counts during training, and continue to outpace them when noise is introduced
         1. See 2a4b10p 1 Hamming and 2a6b10p 1 Hamming
   5. Random Distribution
      1. See 2a4b10p 5 Hamming
         1. Need to run a 2a6b10p, 4a8b10p, and 4a12b10p at 5 Hamming?
      2. Need to do a full run on the original non clustered version to show parity in totally random starting barcodes
5. Conclusions
   1. Given the simplistic nature of our embedding model, any deficiencies in performance can most likely be smoothed out by a better choice of model layers or hyperparameters.
   2. The resilience to noise that is shown by this simple model does warrant further investigation, however, due to the real nature of task identifier never being perfect or consistently denoised.
   3. Showing the parity of our model even in unclustered situations, we think this might be a viable alternative for storing keys in memory, and look forward to seeing how we and others can continue to investigate this new avenue of thought.
6. Future work
   1. Make the embedding model better
      1. Currently, it never does better than ~90% when predicting barcodes from embeddings
      2. There were some indications of long-term problems in the learning capacity of the model, where it would be stable for ~2000 epochs, but then performance would decrease due to a loss in accurate predictions. We didn’t investigate the cause behind this for this paper.
         1. However, this was over extended training, far past the point where the model performance had plateaued, so we aren’t sure if it’s a problem or a hiccup.
   2. Harder Tasks and Weirder Noise
      1. Our inputs and noise are all binary vectors of varying sizes.
      2. Would this performance hold up under a continuous domain of input values?
      3. Does our choice of the Contextual Bandit Task lend some unknown advantage to our noisy approach?