**High-Level Summary of the Large Outline of the Preprint of the Paper**

**Version 2**

Key Findings:

We’ve shown that encoding and retrieving memory at the hidden state space instead of input space still holds comparable performance, can be more resistant to input noises (which we measured in contextual bandit tasks), and can work better in partially observable situations as it is able to integrate information over time steps (shown in maze solving tasks). This is due to our model learning the internal task representations better, which we visualize via t-SNE on the keys stored in memory for our version compared to Ritter. We’ve tracked this effect over a variety of different configurations of the Bandit Task successfully, with intriguing results in situations with higher numbers of tasks and increased task identifier size.

Methods:

We start from the standard Ritter 2018 epLSTM model, which takes an input, passes it through a single cell LSTM, and makes action choices using an A2C policy network. While Ritter took their inputs and stored portions of them in memory to be searched over during the LSTM update step, our model inserts a new secondary model in between the LSTM and the memory.

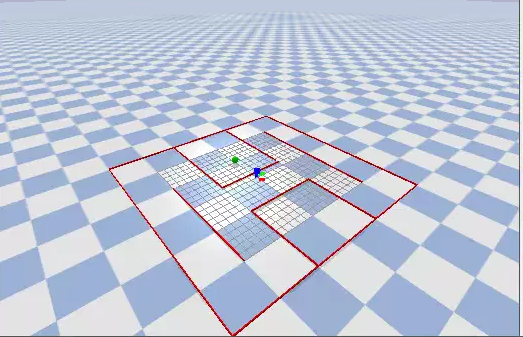
We take the hidden state of the LSTM and transform it into a larger embedding, which we then use as the key for the memory instead of Ritter's direct storage system. We still utilize the same method of storing the cell state of the LSTM as the value attached to these keys, and reintegrate those cell states using the R-Gate as proposed by Ritter.

Currently, our embedding model is as naive as possible, with just a single dense layer between the hidden state and the embedding we store in memory. We train the model using the contexts presented to the LSTM, in order to let the embeddings separate based on these task identifications. We wanted to keep the model as simple as possible to make sure we didn’t add in any unexpected effects from the model we couldn’t account for yet.

For the Contextual Bandit task, we follow the same procedure as outlined by Ritter, with the unique barcodes being held constant over training, but switching the mapping of which barcode corresponds to the arm with the higher probability of reward. In order to add noise after the model is trained, we randomly choose a percentage of the bits in the barcodes to flip, and then keep that choice of noise constant over the course of a single episode. One addition we make to the bandit task is the initial clustering of the barcodes before training. Since we represent the barcodes as binary vectors, we use Hamming distance to create small intracluster separation, but allow for high intercluster separation. This clustering allows us to show how Ritter has significantly more difficulty in learning an ideal policy compared to our model.

Main Questions:

* Suggestions for visualizations of the keys in memory, given how t-SNE is turning into accidental abstract art for some of our larger test cases
* Reasons behind long-term accuracy decrease over massively extended training (3-5x longer than our graphs currently are showing)
* How many tasks are required to show the solution we have (will Context Bandits and our Maze task be enough?)
* General paper writing tips for ICLR specifically?

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 passing hidden states of the LSTM through a single dense layer 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. This key would be the barcode for that specific episode in our chosen task.
      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. Task1: 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
      3. 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.
      4. We define an episode as a series of 10 pulls, with rewards generated by the ground truth (unnoised) barcode.
      5. 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.
      6. The mapping of arms to barcode is shuffled at the end of every epoch, with clusters each individually shuffling their assigned arms
         1. ~~Assume A, B, and C were a single cluster, and D, E, and 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. Task2: Maze2D
      1. Task definition: 5 different mazes are shown to the agent within an epoch
      2. Within an epoch, the start and goal locations in each maze will remain fixed.
      3. Across epochs,the same set of 5 mazes will be shown the goal location will change
      4. 
      5. Adapting the supervised model to this task: the supervised model will be trained to predict the maze label from the hidden state to make the keys more distinct and make retrieval of the corresponding values more accurate
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 identifiers 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.
   4. 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?
6. Other methods attempted
   1. Unsup Learning
      1. Created a 4 layer model to obtain the embedding space and a two layer model to predict the whether 2 embeddings belong to the same class - unsupervised learning using Siamese network
      2. The tasks are shown in a batched manner such that all tasks within a batch are distinct
   2. Unsup learning train settings
      1. To make the input distribution more stable for the embedding model, the training was started only after training of the task stabilized. Essentially, the training of the embedding model began only after 90 epochs out of 100 epochs of training
      2. The above training approach on the random vector task showed a clear increase in inter-cluster distance and a decrease in intra-cluster distance. Samples from the same cluster are embeddings of hidden states from the same episode. The reward returns were also comparable to the Ritter memory model
      3. While this trend in embedding similarity showed up in the barcode-bandit task, the memory model was too noisy to allow the A2C to learn and the returns did not improve