Learning to Query: Leveraging Experience Memories

Eric Liang
EECS
UC Berkeley
ericliang@berkeley.edu

Zongheng Yang
EECS
UC Berkeley
zongheng@berkeley.edu

Corey Zumar
EECS
UC Berkeley
czumar@berkeley.edu

Extended Abstract

In this project, we investigate designing reinforcement-learning agents that efficiently utilize a memory of past expert experiences.

Our motivation stems from multi-agent and highly dynamic environments. Consider a network of autonomous drones or self-driving cars, each such agent acting independently; a natural question arises: how can one agent's findings or experiences (e.g., encountering a newly formed pothole) be quickly leveraged by all other agents? The naive approach would be to re-train all agents' internal policies on the newly observed data, which is clearly computationally expensive. To tackle this challenge, one promising direction is the use of *memory* [3, 6, 5, 1]. For the remainder of this paper we restrict our attention to memories consisting of a list of past transitions, i.e., $(s, a, s', V(s))$.

We seek to answer the following practical design choices that need addressed when using a memory, but which seem under-explored in related literature:

- Can a purely read-only memory, populated with experiences from experiences of other agents, improve performance? (Neural Episodic Control [6] accepts writes.)
- Can even a small number of expert demonstrations in memory improve performance (§2.3)? (NEC stores 5×10^5 memories per action.)
- When querying the memory, should we use a hand-chosen distance function (e.g., Gaussian, Euclidean) or a trainable neural network (§2.2.1)?
- When querying with a learned distance network, should the observations be in the original space, or in an embedded space (§2.3.3)?

To evaluate these questions, we designed a hybrid KNN (k-nearest-neighbors) and model-free agent based on the idea of a learnable *distance function* between observations. We evaluate its performance with just the KNN-module enabled (KNN-only), performance after fine-tuning the distance function with policy gradient updates, and performance of the KNN output combined with a standard policy network (KNN-Hybrid). The agent is evaluated against both a simple gridworld environment (FrozenLake) and one with image observations (PongDeterministic-v4).

We find surprisingly strong performance from the KNN-only agent in the gridworld and Pong environments, showing scalability to larger grids than supported by simple model-free agents and near one-shot learning on Pong respectively. In both environments, we also saw indications that the distance function could be fine-tuned via RL training after supervised pretraining on a target metric.

In the gridworld, our KNN-Hybrid agent showed promising results, with faster convergence to a higher reward compared to PPO in some settings. However, we were unable to reproduce these gains in the Pong environment, with our hybrid network architectures (§3.3) converging to a lower reward compared to the baseline and KNN-only agents.

CS294 Deep RL final project, Fall 2017, UC Berkeley.

1 Querying Experience Memory

We present a differentiable action selection policy for environments with discrete state and action spaces. Agents select actions by leveraging an experience memory consisting of state-action-value transition tuples.

State-action notation: We define the d_s-dimensional, discrete state space as $S \subset \mathbb{R}^{d_s}$. Similarly, we define the discrete action space as $A \subset \mathbb{R}^{d_a}$. We also assume the existence of a value function $V : S \to \mathbb{R}$.

Representing memory: The associative experience memory, M, of size $|M| = m$ is defined as

$$M = \{ T_j \mid j \in [1, m] \}$$

where $T_j = (s_j, a_j, s'_j, V(s'_j))$ such that s'_j is the state resulting from taking the action a_j in the state s_j. We proceed to consider action-specific subsets of this experience memory. For an action $a \in A$, its action-specific memory subset of length l_a is defined as

$$M_a = \left\{ T_{j_a} = (s_{j_a}, a, s'_{j_a}, V(s'_{j_a})) \;\middle|\; j \in [1, l_a] \right\}$$

Procedure for action selection: Given a state $s \in S$, the policy applies a learned distance function $D_\theta : S \times S \to \mathbb{R}$ (in practice, this is a feed-forward neural network) to each action-specific memory subset in order to obtain a weighted distribution over all possible actions. We make the simplifying assumption that $M_a \neq \emptyset$ for all $a \in A$, which is easy to ensure in practice.

We define the weighting function $W : M \to \mathbb{R}$ such that, for a given entry $T_{j_a} \in M_a$,

$$W(T_{j_a}) = \frac{V(s'_{j_a})}{D_\theta(s, s_{j_a})^2 + \epsilon}$$

where $\epsilon \in \mathbb{R}^+$ is a small constant to prevent division by zero. In practice, we used $\epsilon = 10^{-3}$. Using W, memory-based logits are obtained for each action $a \in A$ as

$$L = \left\{ \frac{1}{l_a} \sum_{j=1}^{l_a} W(T_{j_a}) \;\middle|\; a \in A \right\}$$

These logits are then used to select an action in one of two ways, depending on the policy configuration:

(a) **Immediate softmax and random sampling**: In this case, a temperature approximation constant, $\alpha \in \mathbb{R}^+$, is applied to the memory-based logits, yielding

$$L^* = \{ \alpha x \mid x \in L \}$$

The softmax activation function is applied to L^*. This creates a probability distribution over all actions. Using the probability distribution, weighted sampling is then employed to obtain an action.

(b) **Learned incorporation**: Alternatively, the policy applies a learned incorporation function $I_\theta : \mathbb{S} \times \mathbb{R}^{|A|} \to \mathbb{R}^{|A|}$ (in practice, this is a feed-forward neural network) that uses the memory-based action logits, L, as supplemental information to obtain a final set of action logits. These logits are obtained as $L_{dec} = I_\theta(s, L)$. Finally, the softmax activation function is applied to L_{dec}, and weighted sampling then leverages the resulting probability distribution to select an action.

Agent-Policy terminology: This procedure for action selection based on an experience memory is a differentiable generalization of K-Nearest-Neighbors (KNN). Accordingly, we refer to the policy that selects actions using subprocedure (a) as the **KNN-only policy**. Agents using this policy are subsequently referred to as **KNN-only agents**. Consequently, the policy that selections actions using subprocedure (b) is referred to as the **KNN-Hybrid policy**. Agents using this policy are subsequently referred to as **KNN-Hybrid agents**.

Pretraining the distance network ($\mathbf{D}_\theta$): Preliminary experiments indicated that randomly initializing distance network weights leads to poor performance. From this random state, the network cannot learn a useful distance metric. Therefore, we chose to explore strategies for performing supervised pretraining of the distance network. We examined pretraining strategies based on two different distance metrics: **L2 distance** and **timestep-based distance**.

In both cases, pretraining begins with the collection of timestep-tagged transition data by performing rollouts in the discrete environment. These rollouts should explore a large fraction of the state space. In practice, rollouts were performed using expert agents, as defined in (§2.1). The first state element s_i of each transition is then extracted, and the cartesian product is applied to the set formed by all of these state elements. This yields a training set of sample state pairs, $T \subset S \times S$.

(a) **L2 distance:** For each pair $(s_i, s_j) \in T$, we compute the distance label as

$$d_{ij} = \log\left(1 + \|s_i - s_j\|_2\right)$$

T is then augmented with distance labels.

(b) **Timestep-based distance:** For timestep-based training, we compute the distance label associated with each pair $(s_i, s_j) \in T$ as

$$d_{ij} = \log\left(1 + |timestep(s_i) - timestep(s_j)|\right)$$

where $timestep(s)$ returns the timestep associated with the transition data element from which the state s was obtained. We then filter out sample state pairs whose associated distance is too large (we used 10 as the threshold for FrozenLake, and 100 for Pong). Intuitively, this filtering is used because timestep differences become less meaningful the further apart the observations are (e.g. due to loops in the MDP). The filtered training set T is then augmented with distance labels.

Once the training set T is fully constructed, it is used to train the distance network for a fixed number of epochs. Afterwards, network weights continue to be adjusted during the SGD iterations of PPO.

2 Evaluation

2.1 Experimental Setup

We evaluate several variants of our proposed memory-utilizing agent on two different OpenAI Gym [2] environments: navigating through a 2D grid with hidden obstacles (modified `FrozenLake`), and the Atari game Pong (`PongDeterministic-v4`).

Task: `FrozenLake-v0` variants (easy). The `FrozenLake` Gym environment is a 2D grid world containing a single goal cell. The agent's task is to navigate to the goal without falling into any number of hidden "holes" in the lake ice. We chose this environment as a simple baseline to validate memory performance. Since FrozenLake requires the memorization of many hidden obstacles, we expect an agent able to leverage memory would have a much easier time as the size of the grid increases.

We make the following changes to FrozenLake: it (1) yields incremental rewards, which makes learning on very large grids tractable, (2) uses Cartesian coordinates (x, y) as the observation rather than the one-hot encoding which scales quadratically with grid size, (3) samples the agent's starting position randomly instead of starting from a fixed position, and (4) allows scaling the grid size by a controllable parameter (e.g. 10×10). In our modified task, we consider a mean reward of > 9 to be task-solving, and any positive reward means it sometimes solves the task.

The experience memory for FrozenLake was initialized using transition data from expert trajectories. To generate these trajectories, we implemented an expert agent with the capacity to check the safety of an action prior to selecting it. Using this capability, the expert simply selected safe moves that proceeded in the direction of the goal.

Task: `PongDeterministic-v4` (harder). For a more challenging environment which has no concretely memorizable objects (e.g., holes), we pick the Atari game Pong. Specifically, the unmodified `PongDeterministic-v4` is used, where the observations are preprocessed into 42×42 grayscale images. To calibrate expectation, we note that a baseline DQN [4] agent achieves 20.5 reward in 2.9M timesteps, while a baseline PPO [7] agent achieves the maximum 21.0 reward in 840k timesteps.

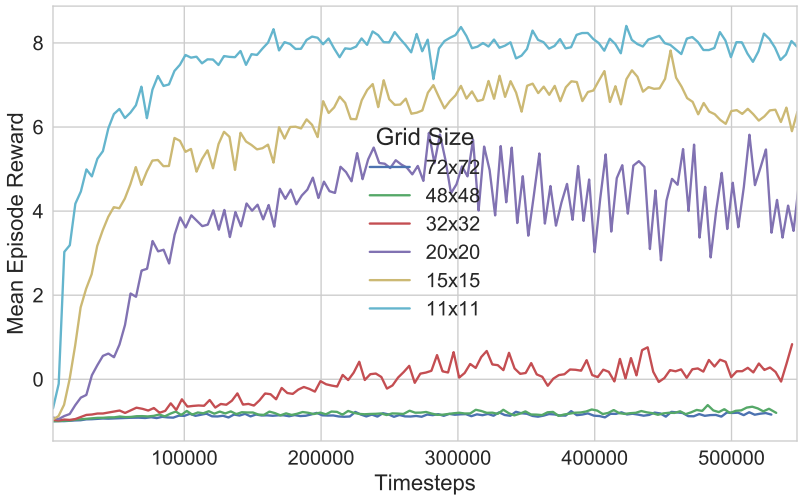


Figure 1: Baseline PPO agent, FrozenLake. Discussion in §2.2.

Distance	10×10	15×15	20×20	32×32	48×48	72×72
L2	6.33 ± 0.42	5.33 ± 0.64	4.32 ± 1.13	0.11 ± 0.97	3.88 ± 1.53	1.09 ± 1.34
Timestep	1.38	1.46	0.26	-0.58	-0.89	-0.96
T.S., tuned	7.89	8.11	3.62	-0.52	-0.90	-0.92

Table 1: Mean reward of the L2 and timestep-based KNN-only agents on FrozenLake (§2.2), and also the timestep-based KNN-only agent after fine-tuning with RL training. Results averaged over multiple episodes, and standard deviation reported where available.

Again, the experience memory was initialized using transition data from expert trajectories. To generate these trajectories for Pong, we trained an expert agent using PPO and performed rollouts with the expert agent.

All experiments are conducted on Amazon EC2 `p2.8xlarge` instances with Tesla K80 GPUs. Models are implemented in TensorFlow 1.4.0 and Ray RLlib 0.2.2.

2.2 FrozenLake

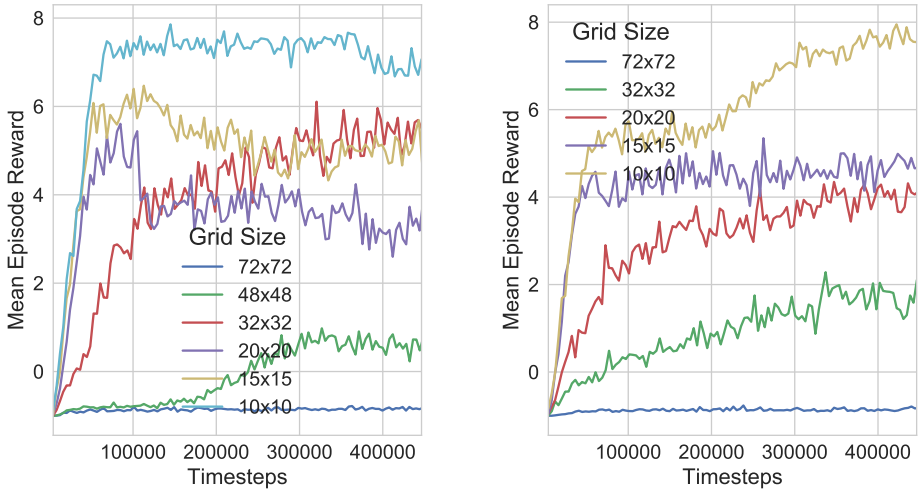
The baseline DQN and PPO scale reasonably well up to 32×32, but stop achieving positive rewards after that point even after hundreds of thousands of timesteps (Figure 1; DQN is not shown, but has similar reward to PPO with reward of 3.6 on 32×32, and similarly negative past that). We found that in contrast, a simple L2 KNN-only agent (Table 1) querying 500 expert trajectories is able to achieve positive rewards on grid sizes up to 72×72.

Our end goal was to demonstrate a hybrid model-free agent that can take advantage of the KNN output to learn more quickly. Towards this end, we next validated the following questions:

2.2.1 What is the right distance function?

Two candidate distance functions are L2 distance between observation vectors, and a *timestep-based* distance (§1), where the distance is an estimated timestep difference between observations. We evaluated both in Table 1, and found that for this task the L2 distance yielded superior results out of the box. However, both had some positive results.

Figure 2: The original supervised distance loss increases along with mean reward over 1m timesteps as RL gradient updates are applied to the KNN-only Pong agent. The distance network here is pretrained to emit $\log(1 + l_2)$ between two Pong image observations. Discussion in §2.2.2.



(a) L2 KNN-Hybrid agent

(b) Timestep-based KNN-Hybrid agent

Figure 3: Incorporation of distance KNN outputs ("hybrid" agents), FrozenLake. Discussion in §2.2.3.

2.2.2 Can the distance function be fine-tuned by policy gradients?

In order to further train the distance functions, we encoded them both as neural networks (note that the timestep-based distance is always estimated), by pretraining them on randomly sampled pairs of observations. We then trained these KNN-only agents with Proximal Policy Optimization. In the timestep-based case, the policy gradient updates substantially improves the KNN-only performance on top of the supervised pre-training (last row of Table 1).

We provide an analysis of this fine-tuning in Figure 2. The average reward increases over PPO training of the KNN-only agent, even though the policy gradient updates appear to degrade the loss of the distance function on the original training set. This suggests that the RL training is able to further improve the distance neural network beyond the original supervised pretraining.

2.2.3 Can the distance KNN outputs be incorporated into a model-free agent?

We tried incorporating the value estimates from the Q-network into a baseline PPO agent by concatenating the value estimates with the observation (§1). We evaluated both L2 KNN-Hybrid and Timestep-based KNN-Hybrid agents in Figure 3. To reduce training times, we also stopped the gradient between the incorporation network and the distance network, therefore updating only the policy.

L2 KNN-only	L2 KNN-only, tuned	L2 KNN-Hybrid
6.91 ± 2.40	16 ± 2	-13.2
Timestep KNN-only	Timestep KNN-only, tuned	Timestep KNN-Hybrid
-20.6	-18.7	7.84
L2 PCA KNN-only	L2 PCA KNN-only, tuned	L2 PCA KNN-Hybrid
12.32 ± 3.05	7.84 ± 7.88	$18.83 \pm .94$

Table 2: Mean reward of the L2, timestep-based, and L2 + PCA agents on the Pong task (§2.3). Each agent was evaluated in KNN-only mode, KNN-only with tuning (meaning a neural network was used for the distance function), and in KNN-Hybrid mode. Standard deviations provided when available.

The hybrid agents were able to achieve substantially higher rewards on the 32×32 task, though the improvements were not visible on all tasks. In particular, on the large 48×48 and 72×72 grids, the L2 KNN-only agent (Table 1; 3.88 ± 1.53 and 1.09 ± 1.34, respectively) still out-performed the hybrid agents. We believe that with more training this gap might be closed since in principle the hybrid agent should do strictly better than with KNN-only. Figure 3 also shows that the KNN-hybrid agents were able to learn in fewer timesteps and converged to higher rewards compared to the baseline PPO agent (Figure 1).

2.3 Pong

We ran the same set of experiments on PongDeterministic-v4 as with FrozenLake. We report two surprising findings during our investigation: (1) The KNN-only agent works surprisingly well, achieving near one-shot learning on a single expert demonstration; (2) negative results for the KNN-Hybrid agents on the Pong task.

To better compare our proposed learned distance function with the embedding-based approach of NEC [6], we also implemented a PCA-based embedding of the image observation, reducing its dimensionality from 1764 to 150. Unfortunately our results here are also negative on the Pong task: while RL fine-tuning of the learned distance functions did improve performance over time, our best KNN-only result was from a PCA-KNN agent, and the best KNN-Hybrid result was also with PCA (without further distance function learning).

We summarize our results in the following sub-sections. Due to memory and compute limits (§3.2), these agents used only one expert demonstration trajectory in their experience memory.

2.3.1 L2 distance

Our naive L2 KNN-only agent did surprisingly well on Pong, achieving a mean episode reward of 6.9 ± 2.4. In many episodes, it managed to obtain a perfect score of 21, as an example of one-shot learning. Fine-tuning of this agent with PPO (pretraining a distance network on the L2 distance) yielded an average reward of 16.

However, hybridizing this agent yielded poor results, with a best reward of -13 after 5.5 million timesteps of training. We tried several neural network architectures for incorporating the distance value estimates, discussed further in §3.3.

2.3.2 Timestep-based distance

The timestep-based distance network performed poorly in all cases, achieving at most 7.84 reward with a standard deviation of 7.88 in the hybrid setting (much more unstable than the baseline PPO).

2.3.3 PCA L2 distance

Using a PCA-based embedding of the observations worked the best, achieving 12.32 reward with L2 KNN-only (one-shot performance). With RL fine-tuning the KNN-only agent achieved 17 mean

reward, and 18 mean reward with hybridization. We also tried using the timestep-based distance function, but PCA did not appear to help in that situation.

3 Implementation

3.1 Network architecture

Distance network. We used the same distance network architecture for both L2 and timestep-based distance metrics: two 64×64 hidden layers with tanh activation.

Policy network. The policy network used for all FrozenLake baseline and KNN-hybrid experiments had 256×256 layers with tanh activation. For Pong, we used a convolutional architecture with two convolutional layers followed by two fully-connected layers (from the reference PPO implementation).

Computing KNN action weight estimates. Obtaining per-action weight estimates requires computing the distance between an agent's state and every other state stored in the experience memory. This is a computationally-intensive evaluation task for the distance network. When using batch training (e.g. for computing gradients), we have to apply the network to all pairs between the batch and memory (e.g. batch size $\times$ memory size evaluations). This was the most complex piece of TensorFlow code we implemented, requiring more than 300 lines of Python to properly define this part of the graph.

This implementation also had several limitations; we materialized the large batch size $\times$ memory sized tensor directly in memory, which limited both the batch and memory size. We believe this could be further optimized by using the `tf.While` control-flow operator to avoid large intermediate results.

3.2 Training

One main downside of using attempting to fine-tune the distance network with RL was that it made training very compute intensive. This is because the forward and backward passes need to be computed over a large dataset for each experience sample. We found that this severely limited our ability to evaluate scenarios – we had to downscale the image observations to $42 \times 42 \times 1$, limit the memory size, and parallelize over multiple GPUs in order to run the experiments we did.

In contrast, published external memory approaches like NEC tended to use external data structures such as KD-trees to limit the amount of experience data queried. In principle the advantage of our approach was that we could train the distance function. Though we didn't try this, it may be possible to still train the distance function by sampling random subsets of the memory for evaluation instead of using all of it.

3.3 Hybrid network architectures for Pong

In Section 2.3 we reported negative results for KNN-Hybrid agent with Pong. Here we describe several unsuccessful incorporation architectures.

- Concatenate the action-value estimates with the flattened image observation and feed it to a fully-connected network.
- The above but with an additional skip-layer connection to feed the action-value estimates to the last layer.
- Concatenate the convolutional network output with the action-value estimates, and feed to a linear layer (and also variants with some additional hidden layers with skip-layer action-value inputs).
- Feeding just the action-value estimates to a fully connected network also didn't work, which was surprising since all the network had to learn to do was copy the inputs to the outputs. It is possible that given more time the network would have eventually learned, but we terminated all runs after 12 hours due to time constraints.

3.4 PCA for Pong image observations

To reduce the dimensionality of Pong image inputs, we begin by performing PCA on a state matrix constructed from expert trajectory data. Expert states are first flattened and stacked to form a matrix. Then, we perform PCA on the state matrix to obtain a component vector basis consisting of 1764 (42×42) unique vectors. We then extract the 150 vectors with the largest corresponding eigenvalues (associated variance) to obtain a transformation matrix that projects batches of origin state inputs into a 150-dimensional subspace. This projection matrix is also applied to memory states and pretraining data. As a result, our policy architecture operates on $1/10^{th}$ the number of original features, removing noise from low-variance features (which are abundant in the context of Pong images) and reducing the latency of SGD iterations.

4 Related Work

The most closely related piece of work to us is NEC [6], which evaluates a similar hybrid model-free and KNN agent. However, they assumed an embedding and did not attempt to learn a distance metric (which they stated as possible future work). There are also several papers that use the idea of incorporating extra information into a model-free policy (e.g. DeepMind's I2A [8]).

The idea of learning a distance metric is also closely related to the idea of *attention* in neural networks, used in frameworks such as Neural Turing Machine [3] and Memory Networks [9], as well as many other network architectures.

5 Conclusion

In this project we investigated the design of reinforcement-learning agents based on KNN action values as well as a hybrid KNN and model-free policy. While we were able to demonstrate the feasibility of learning distance functions, with concrete gains in some scenarios, it is still unclear if they have any advantages over embedding-based approaches. To proceed further would require development of new techniques to train more efficiently over large memories, as we found the computational cost of incorporating memory to be prohibitive even with the use of GPUs. In addition, it is unclear why the hybrid networks we implemented for Pong did not learn well.

Finally, we chose to investigate learning a distance metric because we believe it could serve as a useful primitive for more sample-efficient RL algorithms. While we were not able to demonstrate performance gains over embedding-based approaches in the tasks we investigated, further investigation or applications to other architectures may be fruitful.

References

[1] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, and D. Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[3] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] C. Olah and S. Carter. Attention and augmented recurrent neural networks. *Distill*, 2016.

[6] A. Pritzel, B. Uria, S. Srinivasan, A. Puigdomènech, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell. Neural episodic control. *arXiv preprint arXiv:1703.01988*, 2017.

[7] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[8] T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

[9] J. Weston, S. Chopra, and A. Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.