Memory-Consistent Neural Networks for Imitation Learning

Kaustubh Sridhar[1], **Souradeep Dutta**[1], **Dinesh Jayaraman**[1], **James Weimer**[2], **Insup Lee**[1]
[1]University of Pennsylvania, [2]Vanderbilt University
{ksridhar, duttaso, dineshj, lee}@seas.upenn.edu, james.weimer@vanderbilt.edu

Abstract

Imitation learning considerably simplifies policy synthesis compared to alternative approaches by allowing access to expert demonstrations. For such imitation policies, errors outside the training samples and away from the expert distribution are particularly critical. Even rare slip-ups in the policy action outputs can compound quickly over time, since they lead to unfamiliar future states where the policy is still more likely to err, eventually causing task failures. Prior approaches to address this issue have relied on online experience, reward-labeled transitions, and queryable experts, all of which entail more complicated policy learning. We instead revisit simple supervised "behavior cloning" for conveniently training the policy from nothing more than pre-recorded demonstrations, but carefully design the model class to counter the compounding error phenomenon. Our "memory-consistent neural network" (MCNN) outputs are hard-constrained to stay within clearly specified permissible regions anchored to prototypical "memory" training samples. Using MCNNs on 14 imitation learning tasks spanning dexterous robotic manipulation and driving, proprioceptive inputs and visual inputs, and varying sizes and types of demonstration data, we find large and consistent gains in performance, validating that MCNNs are better-suited than vanilla deep neural networks for imitation learning applications. Our code can be found at https://github.com/kaustubhsridhar/MCNN.

1 Introduction

For sequential decision making problems such as robotic control, imitation learning is an attractive and scalable option for learning decision making policies when expert demonstrations are available as a task specification. Such demonstrations are typically easier to provide than the typical task specification requirements for reinforcement learning and model-based control, namely, dense rewards and good models of the environment. Furthermore, imitation learning is also typically less experience-intensive than reinforcement learning and less expertise-intensive than model-based control.

We consider the simplest and perhaps most widely used imitation learning algorithm, behavior cloning (BC) [20], which reduces policy synthesis to supervised learning over the expert demonstration data. For example, a neural network policy for an autonomous car could be trained to mimic human driving actions [2]. While the policy is synthesized with supervised learning, the evaluation setup is very different: rather than merely achieving low average error on new states from the training distribution as common in supervised learning, the trained policy must, when rolled out in the world, successfully accomplish the demonstrated task.

This sequential deployment makes the behavior of imitation policy functions outside their training data particularly critical. To see this, observe that during rollout, the policy's own output actions determine its future input states. Task performance is most closely tied to the policy's behavior on this self-induced state distribution, which usually differs from the training expert distribution. In

Preprint. Under review.

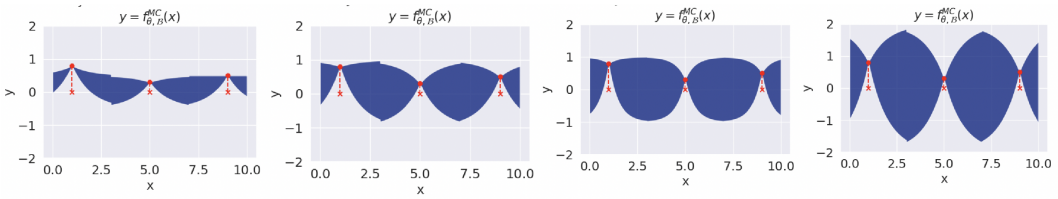

Figure 1: Our memory-consistent neural network (MCNN) formulation can express model classes that are constrained to various extents, depicted in the dark blue permissible regions, around "memories" subsampled from the training dataset shown in red circles. When used for imitation learning, this design of the model class reduces errors away from the expert distribution, boosting task performance.

particular, a minor error in the policy's action output at any time t may induce a future input state s_{t+1} that is subtly different from expert states. If the policy behaves erratically under such small deviations, as it often does in practice, the situation quickly snowballs into a vicious cycle of compounding errors leading to task failure.

Past solutions to this compounding error problem have focused on modifying the behavior cloning setup, such as by permitting online experience [8, 25], reward labels [19], queryable experts [29], or modifying the demonstration data collection procedure [12]. Instead, we retain the conveniences of the plain BC setup and focus on designing a model class that encourages better behavior beyond the training data, which in turn could boost task performance by mitigating the compounding error phenomenon discussed above.

We argue that vanilla deep neural networks are too powerful for behavior cloning — they can generate large errors when evaluated away from the training distribution, and even rare errors could derail an entire task rollout. To tame them, we propose semi-parametric "memory-consistent neural networks" (MCNN). MCNNs first subsample the dataset into representative prototype "memories" to form the scaffold for the eventual function. They then fit a parametric function to the rest of the training data that is hard-constrained by the very formulation of the model class, to exactly fit the training data at all the memories, and further, to stay within double-cone-like zones of controllable shapes and sizes centered at each memory. As a result, an MCNN behaves mostly like a nearest-neighbor function close to memories, and mostly like a deep neural network (subject to the double-cone constraints) far from them.

Fig 1 visualizes the MCNN model class for a fixed 1-D dataset, with fixed memories shown in the red circles. Consider first the leftmost plot. All functions in this MCNN model class lie within the dark blue double-cone-like "permissible regions" centered on each memory, meaning that function values away from the training data are bounded. Under mild assumptions on the expert policy, we show that this property of MCNNs induces an upper bound on the suboptimality of the learned BC policy. Further, it is easy to modulate the shape of the permissible regions to trade off model class capacity and the error bound — indeed, the four plots in Fig 1 show four such MCNN model families with increasing capacity, indicated by the growing permissible regions. Using MCNNs on 14 imitation learning tasks spanning dexterous robotic manipulation and driving, proprioceptive inputs and visual inputs, and varying sizes and types of demonstration data, we find large and consistent gains in performance, validating that MCNNs are better-suited than vanilla deep neural networks for imitation learning applications.

2 Related Work

In imitation and reinforcement learning, non-parametric methods such as RBFs [26], SVMs [13], and nearest neighbours [30] have shown competitive performance on various robotic control benchmarks. But, only recently, a combination of neural networks for representation learning and k-nearest neighbors for control was proposed in Visual Imitation through Nearest Neighbors (VINN) [18]. This is the closest paper to our work and in Section 5, we compare with VINN and demonstrate that we outperform their Euclidean kernel weighted (k) nearest neighbors comprehensively. Other semi-parametric methods include SPIN (Semi-Parametric Inducing point Networks) [27] and Self-Attention between Datapoints [10]. These works propose to use parametric models to approximate non-parametric methods. Our work instead smoothly interpolates between non-parametric and

parametric models and demonstrates an improved generalization capability. Finally, our theorem on the sub-optimality gap in imitation learning with MCNNs builds on earlier work on reductions for imitation learning in [28, 22, 2, 23]. It also leverages intuitions from [17] on bounding the width of the model class, which was useful in a model-based RL setting. To meet space constraints, we leave a more detailed discussion of related work to the Appendix.

3 Problem Formulation

A *Markov Decision Process (MDP)* is a tuple $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{I})$, where $\mathcal{S} \subseteq \mathbb{R}^n$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}(s'|s,a)$ is the probability of transitioning from state s to s' when taking action a, $\mathcal{R}(s,a)$ is the reward accrued in state s upon taking action a, $\gamma \in [0,1)$ is the discount factor, and $\mathcal{I}$ is the initial state distribution. We assume that the MDP operates over trajectories with finite length H, in an episodic fashion. Additionally, we want the set of states $\mathcal{S}$ to be closed and compact. A deterministic policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ maps elements in the state-space to actions, which induces a behavior in the evolution of the MDP. The expected cumulative reward accrued over the duration of an episode, is given by the following,

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=1}^{H} \mathcal{R}(s_t, a_t)\right]. \tag{1}$$

In imitation learning, we assume that there exists an expert policy π^* unknown to the learner. This policy induces a distribution d_{π^*} on the state-action space $\mathcal{S} \times \mathcal{A}$ obtained by rollouts on the MDP. The learner agent has access to an expert trajectory dataset $D = \{(s_0, a_0), (s_1, a_1), \ldots, (s_N, a_N)\}$ drawn from distribution d_{π^*}. The goal of imitation learning is to estimate a policy $\hat{\pi}$, which mimics the expert's policy and reduces the *sub-optimality* gap: $J(\pi^*) - J(\hat{\pi})$.

4 Approach

Our approach involves developing a new model class, memory consistent neural networks (MCNN), and training it with supervised learning to clone the expert from the demonstration data. We start by setting up the MCNN model class in Sec 4.1, analyze its theoretical properties for imitation learning in Sec 4.2, and finally describe our behavior cloning algorithm that uses MCNNs in Sec 4.3.

4.1 The Model Class: Memory-Consistent Neural Networks

First, we develop the semi-parametric MCNN model class for imitation learning. MCNNs rely on a "memory" code-book set $\mathcal{B} := \{(s_i, a_i)\}_{i=1}^{K}$ which are subsampled from the expert training dataset and summarize it. In practice, such a memory code-book can be created using one of various off-the-shelf approaches. We describe our algorithmic choices later in Sec 4.3. For notational convenience, we describe the approach for a scalar action space, but it is easily generalizable to the vector action spaces we evaluate in our experiments.

Given this memory code-book $\mathcal{B}$, we now define a "nearest memory neighbor policy". For a set $S \subset \mathcal{S}$, and an input $x \in \mathcal{S}$, we first define its closest element in S as, $C_S(x) = \arg\min_{s \in S} d(s, x)$, where d is some distance metric defined on the space $\mathcal{S}$. We denote by $\mathcal{B}|_S$, and $\mathcal{B}|_A$ as the set of all states and actions captured by the memory code-book $\mathcal{B}$. With slight abuse of notation, we denote $\mathcal{B}(s)$ as the action assigned by the codebook for a state input s. Using the above, we now define a nearest neighbor regression function f^{NN} as the following,

Definition 4.1 (Nearest Memory Neighbor Function)**.** For an input $x \in \mathcal{S}$, assume that $s' = C_{\mathcal{B}|_S}(x)$, then $f^{NN}(x) := \mathcal{B}(s')$.

In other words, the nearest memory neighbor function assigns actions according to a nearest neighbor look-up in the memory code-book $\mathcal{B}$.

We are now ready to define *memory-consistent neural networks* (MCNN), which permit interpolating between the nearest neighbor and parametric deep neural network (DNN)-based function approximation settings. Let f^{θ} denote a DNN function parameterized by θ, which maps from MDP states to actions.

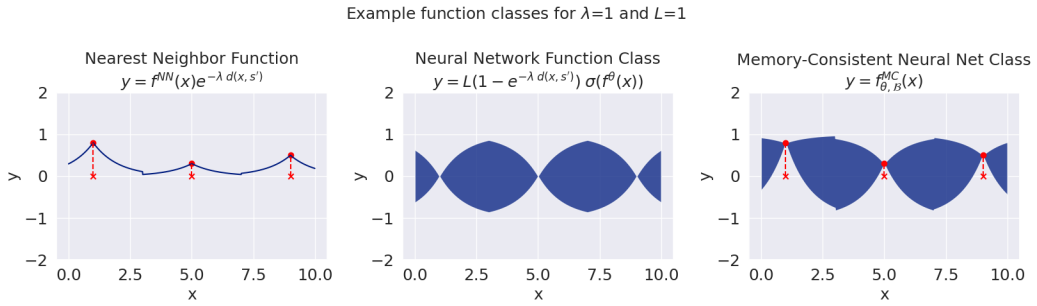

Figure 2: **The elements of the MCNN model class.** The left panel shows the nearest memory neighbour component from Equation 2. The middle panel depicts the constrained neural network function class, where the shaded regions represent the permissible regions; by design, the function cannot take values outside these shaded regions. Finally, the right panel shows the combined MCNN model class. For these plots, we set $\lambda = 1, L = 1$. We demonstrate the effects of changing λ and L in the Appendix A.

Definition 4.2 (Memory-Consistent Neural Network)**.** A memory-consistent neural network f^{MC} is defined using the pair $(\mathcal{B}, f^{\theta})$, and hyperparameters $\lambda \in \mathbb{R}^{+}$, and $L \in \mathbb{R}$

$$f^{MC}_{\theta,\mathcal{B}}(x) = \underbrace{f^{NN}(x)\left(e^{-\lambda\, d(x,s')}\right)}_{\text{Nearest Memory Neighbour Function}} + \underbrace{L\left(1 - e^{-\lambda\, d(x,s')}\right)\ \sigma(f^{\theta}(x))}_{\text{Constrained Neural Network Function Class}} \tag{2}$$

where, $s' = C_{\mathcal{B}|_S}(x)$ is the nearest memory to x, and $\sigma : \mathbb{R} \mapsto [-1, 1]$ is a compressive non-linearity that imposes hard limits on the outputs of f^{θ}. In practice, we use the $\tanh$-like function.

We refer the reader to Figure 2 to drive the intuition. For inputs which are close to the points in the code-book $\mathcal{B}$, the function predicts values which are similar to the one observed in the training dataset. More concretely, the value predicted by the function is a simple mixture: $\alpha f^{NN}(x) + (1 - \alpha)\ L\ \sigma(f^{\theta}(x))$, where the mixing factor $\alpha \in [0, 1]$ changes in proportion to the distance to the nearest memory. Thus, for points further away more weight is placed on the neural network and the memories have little influence. The degree of deviation from nearest neighbor prediction f^{NN}, that the function can have when in the vicinity of a memory point is controlled by the parameter λ. Thus, we obtain a purely nearest neighbor function for $\lambda = 0$, and a vanilla deep neural network function for $\lambda = \infty$. Note, that the MCNN function values in regions far away from memories are in the set $[-L, L]$. This is a realistic assumption in practice, and is often readily available in the form of actuator bounds. We demonstrate the effects of changing λ and L in the Appendix A.

4.2 Theoretical Analysis of MCNNs for Imitation Learning

For fixed hyperparameters L, λ and memory codebook $\mathcal{B}$, we denote by $\mathfrak{F}$, the class of memory-consistent functions outlined in Equation 2. Note, that a choice of the DNN function parameters θ fixes a specific function in this class as well.

Assumption 4.3 (Realizability)**.** We assume that the expert policy π^* belongs to the function class $\mathfrak{F}$.

This assumption trivially holds at the memories, where the MCNN exactly reproduces expert actions. For all other points, we assume that there exist some parameters θ, which can capture the policy π^* with sufficient accuracy, for a choice of L and λ. For a point x at a distance of $d(s', x)$, the vanilla DNN can affect the predictions only by an amount of $\pm L\left(1 - e^{-\lambda\, d(x,s')}\right)$. Without this restriction, we might have been able to capture behaviors which went well beyond these ranges. However, expert policies often have such bounded behavior, and do not overshoot a certain threshold. What we propose here is a way to enforce this bound using a zeroth order nearest neighbor estimate.

We analyze the behavior of this function class, and note some useful lemmas along the way. For a set of memories present in $\mathcal{B}|_S$, we wish to capture the maximum value that the distance term : $d(x, s')$ can take in Equation 2. To that end, we define the *most isolated state* as the following:

Definition 4.4 (Most Isolated State)**.** For a given set of memory points $\mathcal{B}|_S$, we define the most isolated state $s^I_{\mathcal{B}|_S} := \arg\max_{s \in S} \left(\min_{m \in \mathcal{B}|_S} d(s, m) \right)$, and consequently the distance of the most isolated point as $d^I_{\mathcal{B}|_S} = \min_{m \in \mathcal{B}|_S} d(s^I_{\mathcal{B}|_S}, m)$

The distance of the most isolated state captures the degree of emptiness that persists with the current knowledge of the state space due to memory code-book $\mathcal{B}$.

Lemma 4.5. *Assume two sets of memory code-books $\mathcal{B}_i$, $\mathcal{B}_j$, such that $\mathcal{B}_i \subseteq \mathcal{B}_j$, then $d^I_{\mathcal{B}_i|_S} \geq d^I_{\mathcal{B}_j|_S}$*

Proof The proof of the above lemma is straightforward, since the infimum of a subset ($\mathcal{B}_i$), is larger than the infimum of the original set ($\mathcal{B}_j$). □

This observation is useful when we study the effects of increasing the size of the code-book $\mathcal{B}$. Note, when learning a memory-consistent neural network $f^{MC}_{\theta,\mathcal{B}}$, we deploy the standard SGD based training to adjust the parameters θ. The choice of the number of memories in $\mathcal{B}$ is kept as a hyperparameter. This allows us to bound the maximum width of the function class, first described in [17] .

Lemma 4.6. *(Width of Function Class) The width of the function class $\mathfrak{F}$, $\forall\ \theta_1, \theta_2 \in \Theta$, and $\forall s \in \mathcal{S}$, defined as $\max_{\theta_i, \theta_j} |\left(f^{MC}_{\theta_i,\mathcal{B}} - f^{MC}_{\theta_j,\mathcal{B}}\right)(s)|$ is upper bounded by : $2L \times \left(1 - e^{-\lambda\ d^I_{\mathcal{B}|_S}}\right)$*

Proof Using Equation 2, we can express the difference as the following :

$$\|f^{MC}_{\theta_i,\mathcal{B}} - f^{MC}_{\theta_j,\mathcal{B}}\|(x) = \|L\left(1 - e^{-\lambda\ d(x,s')}\right)\ \sigma(f^{\theta_i}(x)) - L\left(1 - e^{-\lambda\ d(x,s')}\right)\ \sigma(f^{\theta_j}(x))\| \tag{3}$$

$$= L\left(1 - e^{-\lambda\ d(x,s')}\right) \|\sigma(f^{\theta_i}(x)) - \sigma(f^{\theta_j}(x))\| \tag{4}$$

$$\leq L\left(1 - e^{-\lambda\ d^I_{\mathcal{B}|_S}}\right) \|\sigma(f^{\theta_i}(x)) - \sigma(f^{\theta_j}(x))\| \tag{5}$$

Now, observing that $\|\sigma(f^{\theta_i}(x)) - \sigma(f^{\theta_j}(x))\| \leq 2$, completes the proof. □

Theorem 4.7. *The sub-optimality gap $J(\pi^*) - J(\hat{\pi}) \leq min\{H, H^2|\mathcal{A}|L\left(1 - e^{-\lambda\ d^I_{\mathcal{B}|_S}}\right)\}$*

Proof In order to take advantage of well-known results in the imitation learning literature [28, 22, 2, 23], we restrict ourselves for the purpose of this analysis to the discrete action-space $\mathcal{A}$ scenario, where the policy $\pi : \mathcal{S} \mapsto \Delta(\mathcal{A})$. The actions picked in the expert dataset D, induces a dirac distribution over the actions space , corresponding to an input state.

Recall that in imitation learning, if the population total variation (TV) risk $\mathbb{T}(\hat{\pi}, \pi^*) \leq \epsilon$, then, $J(\pi^*) - J(\hat{\pi}) \leq min\{H, H^2\epsilon\}$ (See [22] Lemma 4.3).

We note the following for population TV risk,

$$\begin{aligned}
\mathbb{T}(\hat{\pi}, \pi^*) &= \frac{1}{H}\sum_{t=1}^{H} \mathbb{E}_{s_t \sim f^t_{\pi^*}}\left[TV(\hat{\pi}(\cdot|s_t), \pi^*(\cdot|s_t))\right] \\
&\leq \frac{1}{H}\sum_{t=1}^{H} \mathbb{E}_{s_t \sim f^t_{\pi^*}}\left[|\mathcal{A}|L\left(1 - e^{-\lambda\ d^I_{\mathcal{B}|_S}}\right)\right] \\
&\leq |\mathcal{A}|L\left(1 - e^{-\lambda\ d^I_{\mathcal{B}|_S}}\right)
\end{aligned} \tag{6}$$

Where, $f^t_{\pi^*}$ is the empirical distribution induced on state s^t obtained by rolling out policy π^*. For the first inequality, in the above derivation we use Lemma 4.6. Using this, in the performance gap lemma gives us the following :

$$J(\pi^*) - J(\hat{\pi}) \leq min\{H, H^2|\mathcal{A}|L\left(1 - e^{-\lambda\ d^I_{\mathcal{B}|_S}}\right)\}$$

□

Corollary 4.8. *Using Lemma 4.5 we know that if $\mathcal{B}_i \subseteq \mathcal{B}_j$, then $\left(1 - e^{-\lambda\ d^I_{\mathcal{B}_i|_S}}\right) \geq \left(1 - e^{-\lambda\ d^I_{\mathcal{B}_j|_S}}\right)$. This can result in lower performance gap according to Theorem 4.7, when $H \geq H^2|\mathcal{A}|L\left(1 - e^{-\lambda\ d^I_{\mathcal{B}|_S}}\right)$. Hence, reflecting the utility of adding more memories in such cases.*

Algorithm 1 Learning Memories

Input: Offline dataset $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^{N}$, number of memories m
Output: A nearest neighbor based function $f^{NN} : \mathcal{S} \mapsto \mathcal{A}$

1: Nodes $\mathcal{N}$, edges $\mathcal{E} \leftarrow$ NeuralGasClustering($\mathcal{S}$, m) // learns the distribution induced by $\mathcal{D}$
2: Nodes $\mathcal{N}'$, $\mathcal{D}(\mathcal{N}') \leftarrow$ For each node in $\mathcal{N}$, find the closest observation in $\mathcal{D}$, and call this $\mathcal{N}'$. Additionally, return the corresponding action taken by the expert in $\mathcal{D}$, denoted by the map $\mathcal{D}(\mathcal{N}')$
3: $\mathcal{G} \leftarrow$ Define neural-gas with nodes $\mathcal{N}'$, and edges $\mathcal{E}$.
4: Define a memory code-book $\mathcal{B}$ using $\mathcal{B}|_{\mathcal{S}} = \mathcal{G}$ and $\mathcal{B}|_{\mathcal{A}} = \mathcal{D}(\mathcal{N}')$. Pairing nodes in the neural-gas to its corresponding actions.
5: Define a nearest neighbor function $f_{\mathcal{B}}^{NN}$, along the lines described in Definition 4.1 using $\mathcal{B}$.
6: **return** $f_{\mathcal{B}}^{NN}$

We summarize the insights from the above theoretical analysis before moving on. First, our MCNN class of functions is bounded in *width* even though it uses a high-capacity function approximator like DNNs. This translates to the fact that, when trying to imitate a policy which belongs to this function class we only suffer a bounded amount in estimation error (Lemma 4.6). Finally, it might sometimes be useful to add more memories, to gain better imitation learning performance. We do not yet have an exact analysis of the performance gap for continuous state and action space MDPs, but the intuitions developed through this analysis guide our algorithmic choices even in such environments.

4.3 Algorithm: Imitation Learning With Memory-Consistent Neural Network Policies

We now describe our algorithm to use MCNNs for imitation learning. The first step in our method is to learn the memory code-book $\mathcal{B}$ from the expert trajectory dataset D. The goal of Algorithm 1 is to build the nearest memory neighbor function f^{NN}. This is followed by details on the training aspects of the MCNN parameters from the imitation dataset $\mathcal{D}_e$ in Algorithm 2.

For building the memory code-book, we leverage an off-the-shelf approach, Neural Gas [6], that selects prototype samples to summarize a dataset. For completeness, we summarize this approach briefly below.

Definition 4.9 (Neural Gas)**.** A neural gas $\mathcal{G} := (\mathcal{N}, \mathcal{E})$, is composed of the following two components,

1. A set $\mathcal{N} \subset \mathcal{S}$ of the nodes of a network. Each node $m_i \in \mathcal{N}$ is called a memory in this paper.
2. A set $\mathcal{E} \subset \{(m_i, m_j) \in \mathcal{N}^2, i \neq j\}$ of edges among pairs of nodes, which encode the topological structure of the data. The edges are unweighted.

Neural gas. The neural gas algorithm [6, 21, 15] is primarily used for unsupervised learning tasks, particularly for data compression or vector quantization. The goal is to group similar data points together based on their similarities. The algorithm works by creating a set of prototype vectors, also known as codebook vectors or neurons. These vectors represent the clusters in the data space. The algorithm works by adaptively placing prototype vectors in the data space and distributing them like a *gas* in order to capture the density. For more details we refer the reader to [6]. We use this in Algorithm 1 (Line 1) to get the initial clustering. We can now go ahead and outline how "memories" are picked in our case.

Learning memories. Algorithm 1 first uses the neural-gas algorithm to pick candidate points (nodes) $\mathcal{N}$ in the state space. However, these points could be potentially absent in the dataset $\mathcal{D}$, making it hard to associate the correct action. To remedy this situation, we replace these points with the closest states from the training set as *memories*. Such *memory* states come with the corresponding actions taken by the expert. This is then used to define a nearest neighbor function by building the memory codebook $\mathcal{B}$ and defining a function as outlined in Definition 4.1.

Training MCNNs. Finally, we train MCNN policies through gradient descent on the parameters θ of the neural network over the expert dataset D_e. For the compressive non-linearity, we define the σ_β function in Algorithm 2, to smoothly interpolate between a linear function and a $\tanh$-like function. This ensures that gradients are non-zero even when the $\tanh$ is flat.

Algorithm 2 Behavior Cloning with Memory-Consistent Neural Networks: Training

Input: Dataset $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^{N}$, nearest neighbor function $f_{\mathcal{B}}^{NN}$, neural network function $f^{\theta}(.)$, batch size, total training steps T, parameters λ and L.

Output: Learned policy $f_{\theta,\mathcal{B}}^{MC}$

1: **for** step$= 1$ to T **do**

2: Sample batch B from $\mathcal{D}$.

3: Forward propagate $(s_i, a_i) \sim B$ as

$$f_{\theta,\mathcal{B}}^{MC}(x) = f_{\mathcal{B}}^{NN}(x)\left(e^{-\lambda\ d(x,s')}\right) + L\left(1 - e^{-\lambda\ d(x,s')}\right)\ \sigma(f^{\theta}(x))$$

where, s' is the nearest neighbor of x in $\mathcal{B}|_S$, $\sigma_\beta(x) = 2\left[\text{LeakyReLU}_\beta\left(\frac{x+1}{2}\right) - (1-\beta)\text{ReLU}\left(\frac{x-1}{2}\right)\right] - 1$ and $\beta = \max\left(0, 1 - \lfloor\frac{step}{100}\rfloor\right)$.

4: Update $\theta \leftarrow \theta - \nabla\, \mathbb{E}_{(s_i,a_i)\sim B}\, \mathcal{L}(f_{\theta,\mathcal{B}}^{MC}(s_i), a_i)$ where $\mathcal{L}$ is the negative log-likelihood or mean squared loss.

5: **end for**

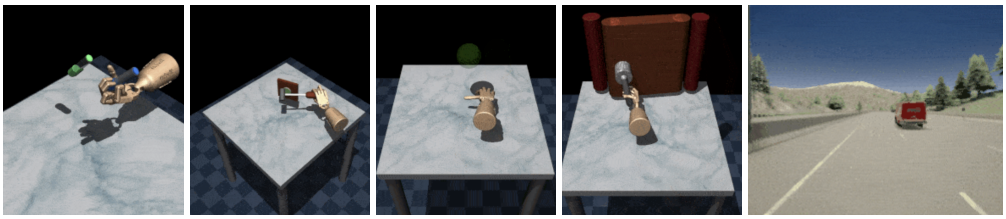

Figure 3: The D4RL [7] environments where we evaluate our trained policies: Adroit Pen, Hammer, Relocate, and Door [24], and CARLA's Town03 and Town04 [3]. The four Adroit environments have proprioceptive observations and the two CARLA environments have image observations.

5 Experimental Evaluation

We now perform a thorough experimental evaluation of MCNN-based behavior cloning in a large variety of imitation learning settings.

Environments and Datasets: We test our approach on 14 tasks, in 6 environments: 4 Adroit dextrous manipulation and 2 CARLA environments pictured in Figure 3. Demonstration datasets are drawn from D4RL [7]. For each Adroit task, we evaluate imitation learning from 3 different experts: (1) small human demonstration datasets ('human') with less than 50 trajectories per task, (2) large demonstration datsets from a well-trained RL policy ('expert'), and (3) mixed data from demonstrations and an imitation policy trained on the demonstrations ('cloned'). In CARLA, we train on demonstrations from a hand-coded expert. For observations, we use high-dimensional states in Adroit pen, hammer, relocate, and door, and 48×48 images in CARLA. Action spaces are 24-30 dimensional in the Adroit dexterous manipulation environments , and 2-D in CARLA.

Baselines: We run the following baselines for comparison. (1) Behavior Cloning: We obtain results with a vanilla neural network architecture. The details of the architecture can be found in the Appendix. We use the same architecture across all parametric methods. We report our results for BC as well as those from Fu et al. [7] under the name 'BC (D4RL)' and 'BC' which is our reimplementation. (2) 1 Nearest Neighbours (1-NN): We set up a simple baseline where the action for any observation in the online evaluation is the action of the closest observation in all the training data. In the expert and cloned datasets for each environment, this amounts to having to perform a search amongst a million datapoints online at every step (which is highly inefficient). (3) Visual Imitation with (k) Nearest Neighbours (VINN) [18]: VINN is a recent method that performs a Euclidean kernel weighted average of some k nearest neighbors. In the Adroit case, we direct perform the k nearest neighbors on the raw observation vectors. In the CARLA case, we perform it in the same embedding space that we use to create memories (we discuss this embedding space more below).

Learning memories and MCNN: We learn neural gas nodes with the incremental neural gas algorithm for 10 epochs starting from 2.5% of the total dataset to 10% of the total dataset for each task. We update all the transitions in each dataset by appending the closest memory observation and its corresponding target action (see more in Algorithm 1). We train the MCNN on this dataset

Table 1: Environment Total Normalized Scores: Comparison of sum of normalized scores (each averaged across 20 episodes and 3 random seeds) between baselines and our method (MCNN+BC) on the Adroit and CARLA environments. Our method, with fixed hyperparameters across all tasks ($\lambda = 10$, $L = 1$, 10% memories), outperforms the baselines in all but one environment.

Environment	Baselines			Ours	
	BC (D4RL) [7]	**1-NN**	**VINN** [18]	**BC** (ReImplemented)	**MCNN + BC** (Fixed Hyperparams)
Adroit pen	176.40	224.70	210.26	228.73	**294.32**
Adroit hammer	127.90	81.27	83.51	109.36	**129.17**
Adroit relocate	101.20	25.96	30.38	105.31	**107.83**
Adroit door	35.30	102.38	105.42	105.24	**106.82**
CARLA lane	31.80	34.03	29.09	35.03	**43.14**
CARLA town	**-1.80**	-13.45	-7.85	-14.95	-15.52

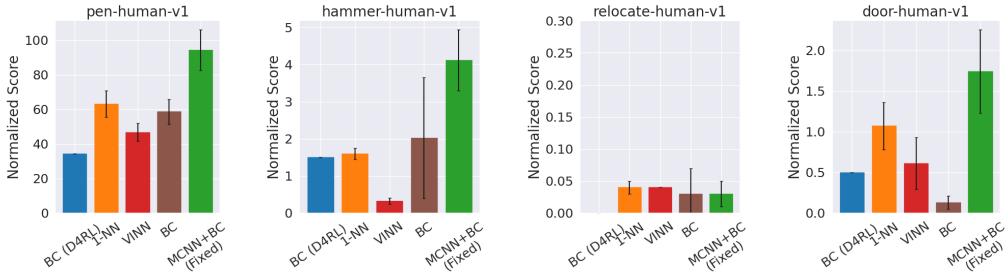

Figure 4: Adroit Human Tasks: Comparison of normalized scores (across 20 evaluation trajectories and 3 random seeds) between baselines and our method (MCNN+BC). For the results shown here, our MCNN approach uses the same fixed set of hyperparameters across all tasks.

following Algorithm 2 for $1e6$ training steps and evaluate on 20 trajectories after training. We also repeat each experiment for a minimum of 3 seeds. We expand on the experimental setup and all hyperparameters in the Appendix.

Embedding CARLA images: In the two CARLA tasks, we use an off-the-shelf ResNet34 encoder [2] that has been shown to be robust to background and environment changes in CARLA to convert the 48×48 images to embeddings of size 512. We use this embedding space as the observation space for learning memories and policies.

Performance Metrics: All our environments come with pre-specified dense task rewards which we use to define performance metrics. We report the cumulative rewards for each task in the form of a normalized score, following prior work[7]. The normalized score for a given policy π measures: how much does π improve cumulative rewards over a random policy, normalized by that same improvement for an expert policy over a random policy. A score of 100 corresponds to expert-level performance, and higher is better.

Results: Table 1 shows the main results. We report the performance at a single set of hyperparameters across all tasks and environments. For the Adroit tasks, we summarize results with the average score over the 3 expert datasets. MCNN+BC outperforms the best baselines in all Adroit environments and one of two CARLA environments, achieving particularly large gains in Adroit pen, hammer, and CARLA lane. We now dive further into the performance with different training distributions in Adroit. With the small human demonstration datasets, on the pen, hammer, and door tasks, we can observe a clear improvement with MCNN as shown in the rightmost bar of each plot in Figure 4. All methods perform poorly in relocate. With "expert" demonstrations in Figure 5, we again observe that MCNN improves the performance of BC uniformly on all tasks while outperforming baselines. The availability of lots of data in a narrow training distribution enables our nearest neighbor component to provide a strong scaffold for the neural network component in MCNN. This is particularly evident in the plots for pen and hammer. With the cloned dataset, in the pen task, we notice a significant improvement with MCNN but all methods perform poorly on the other three tasks. On CARLA (Figure 7), MCNN outperforms the closest baseline by a 31% in the lane task but all method perform poorly in the town task. Finally, if provided limited online interaction (here, 20 episodes), simply

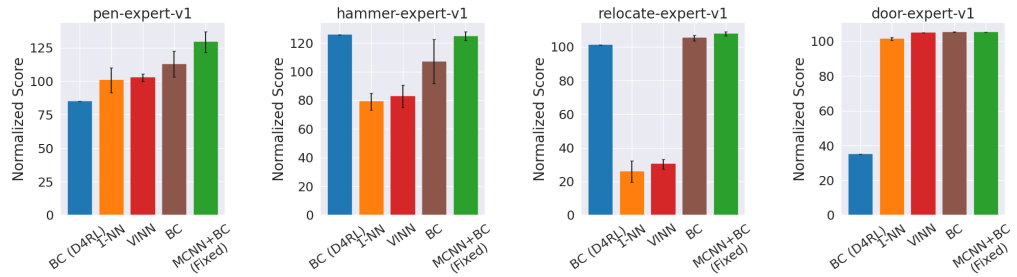

Figure 5: Adroit Expert Tasks: Comparison of normalized scores (across 20 evaluation trajectories and 3 random seeds) between baselines and our method (MCNN+BC). Our method uses the same fixed set of hyperparameters across all tasks.

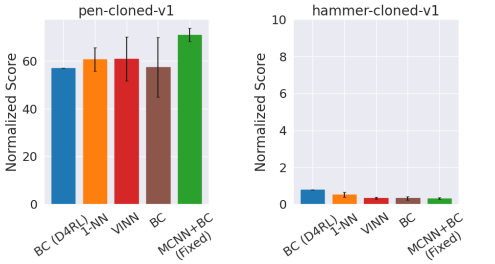

Figure 6: Adroit Cloned Tasks: Comparison of normalized scores (across 20 evaluation trajectories and 3 random seeds) between baselines and our method (MCNN+BC). Our method uses the same fixed set of hyperparameters across all tasks.

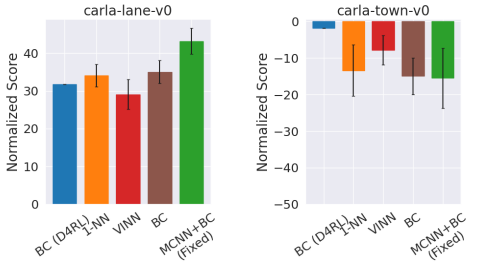

Figure 7: CARLA Tasks: Comparison of normalized scores (across 20 evaluation trajectories and 3 random seeds) between baselines and our method (MCNN+BC). Our method uses the same fixed set of hyperparameters across all tasks.

selecting the best-performing hyperparameters (λ, L, and a number of memories) further improves MCNN performance, as Figure 8 shows. Additional figures and all means and standard deviations of our results can be found in the Appendix.

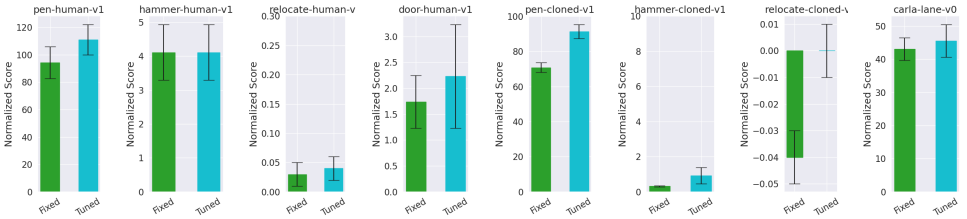

Figure 8: Fixed vs Tuned Hyperparameters in MCNN+BC: Comparison of normalized scores (across 20 evaluation trajectories and 3 random seeds) between our method with fixed hyperparameters across all tasks ($\lambda = 10$, $L = 1$, 10% memories) and with hyperparameters tuned online. This shows that limited online interaction (20 episodes) for finetuning hyperparameters can further improve our method.

Conclusions and Limitations. Imitation learning forms one the promising approaches when it comes to transferring complex robotic manipulation tasks from experts to automated algorithms. Here, we demonstrate that restricting the function class using *memories* can help us control the imitation gap on points outside the training dataset. This can potentially have large ramifications on the performance of the subsequent tasks. While our theoretical and empirical results support the idea that appropriately constraining the function class based on training data memories improves imitation performance, MCNNs are only one heuristic, if sensible, way to accomplish this; it is very likely that there are even better-designed model classes in this spirit, that we have not been explored in this work. Finally, we would like in future work to explore MCNNs as model classes beyond just behavior cloning, in standard supervised learning settings, and perhaps even in reinforcement learning.

References

[1] A. Bertsch, U. Alon, G. Neubig, and M. R. Gormley. Unlimiformer: Long-range transformers with unlimited length input. *arXiv preprint arXiv:2305.01625*, 2023. (Cited on 13)

[2] F. Codevilla, E. Santana, A. M. López, and A. Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019. (Cited on 1, 3, 5, 8, 14)

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017. (Cited on 7)

[4] S. Dutta, Y. Yang, E. Bernardis, E. Dobriban, and I. Lee. Memory classifiers: Two-stage classification for robustness in machine learning. *arXiv preprint arXiv:2206.05323*, 2022. (Cited on 14)

[5] P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021. (Cited on 14)

[6] B. Fritzke. A growing neural gas network learns topologies. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, page 625–632, Cambridge, MA, USA, 1994. MIT Press. (Cited on 6)

[7] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020. (Cited on 7, 8, 16)

[8] J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016. (Cited on 2, 14)

[9] X. Ji, H. Choi, O. Sokolsky, and I. Lee. Incremental anomaly detection with guarantee in the internet of medical things. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, IoTDI '23, page 327–339, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700378. doi: 10.1145/3576842.3582374. URL https://doi.org/10.1145/3576842.3582374. (Cited on 14)

[10] J. Kossen, N. Band, C. Lyle, A. N. Gomez, T. Rainforth, and Y. Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *Advances in Neural Information Processing Systems*, 34:28742–28756, 2021. (Cited on 2, 13)

[11] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020. (Cited on 14)

[12] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on robot learning*, pages 143–156. PMLR, 2017. (Cited on 2, 14)

[13] M. Lee and C. W. Anderson. Robust reinforcement learning with relevance vector machines. *Robot Learning and Planning (RLP 2016)*, page 5, 2016. (Cited on 2, 13)

[14] J. Lyu, X. Ma, X. Li, and Z. Lu. Mildly conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2206.04745*, 2022. (Cited on 14)

[15] T. Martinetz, S. Berkovich, and K. Schulten. 'neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, 1993. doi: 10.1109/72.238311. (Cited on 6)

[16] M. Nakamoto, Y. Zhai, A. Singh, M. S. Mark, Y. Ma, C. Finn, A. Kumar, and S. Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *arXiv preprint arXiv:2303.05479*, 2023. (Cited on 14)

[17] I. Osband and B. V. Roy. Model-based reinforcement learning and the eluder dimension. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, page 1466–1474, Cambridge, MA, USA, 2014. MIT Press. (Cited on 3, 5, 14)

[18] J. Pari, N. M. Shafiullah, S. P. Arunachalam, and L. Pinto. The surprising effectiveness of representation learning for visual imitation. *arXiv preprint arXiv:2112.01511*, 2021. (Cited on 2, 7, 8, 13, 15, 16)

[19] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019. (Cited on 2, 14)

[20] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991. (Cited on 1)

[21] Y. Prudent and A. Ennaji. An incremental growing neural gas learns topologies. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 1211–1216 vol. 2, 2005. doi: 10.1109/IJCNN.2005.1556026. (Cited on 6)

[22] N. Rajaraman, L. F. Yang, J. Jiao, and K. Ramchandran. Toward the fundamental limits of imitation learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546. (Cited on 3, 5, 14)

[23] N. Rajaraman, Y. Han, L. Yang, J. Liu, J. Jiao, and K. Ramchandran. On the value of interaction and function approximation in imitation learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 1325–1336. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/09dbc1177211571ef3e1ca961cc39363-Paper.pdf. (Cited on 3, 5, 14)

[24] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017. (Cited on 7)

[25] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017. (Cited on 2, 14)

[26] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade. Towards generalization and simplicity in continuous control. *Advances in Neural Information Processing Systems*, 30, 2017. (Cited on 2, 13)

[27] R. Rastogi, Y. Schiff, A. Hacohen, Z. Li, I. Lee, Y. Deng, M. R. Sabuncu, and V. Kuleshov. Semi-parametric inducing point networks and neural processes. In *The Eleventh International Conference on Learning Representations*, 2023. (Cited on 2, 13)

[28] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010. (Cited on 3, 5, 14)

[29] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011. (Cited on 2, 14)

[30] D. Shah and Q. Xie. Q-learning with nearest neighbors. *Advances in Neural Information Processing Systems*, 31, 2018. (Cited on 2, 13)

[31] K. Sridhar, S. Dutta, J. Weimer, and I. Lee. Guaranteed conformance of neurosymbolic models to natural constraints. *arXiv preprint arXiv:2212.01346*, 2022. (Cited on 14)

[32] K. Sridhar, O. Sokolsky, I. Lee, and J. Weimer. Improving neural network robustness via persistency of excitation. In *2022 American Control Conference (ACC)*, pages 1521–1526. IEEE, 2022. (Cited on 14)

[33] Y. Sun. Offlinerl-kit: An elegant pytorch offline reinforcement learning library. https://github.com/yihaosun1124/OfflineRL-Kit, 2023. (Cited on 15)

[34] A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. (Cited on 14)

[35] Y. Wu, M. N. Rabe, D. Hutchins, and C. Szegedy. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022. (Cited on 13)

[36] Y. Yang, R. Kaur, S. Dutta, and I. Lee. Interpretable detection of distribution shifts in learning enabled cyber-physical systems. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 225–235. IEEE, 2022. (Cited on 14)

[37] Y. Yang, S. Dutta, K. J. Jang, O. Sokolsky, and I. Lee. Incremental learning with memory regressors for motion prediction in autonomous racing. In *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*, pages 264–265, 2023. (Cited on 14)

[38] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33: 14129–14142, 2020. (Cited on 14)

[39] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, 34:28954–28967, 2021. (Cited on 14)

Appendix

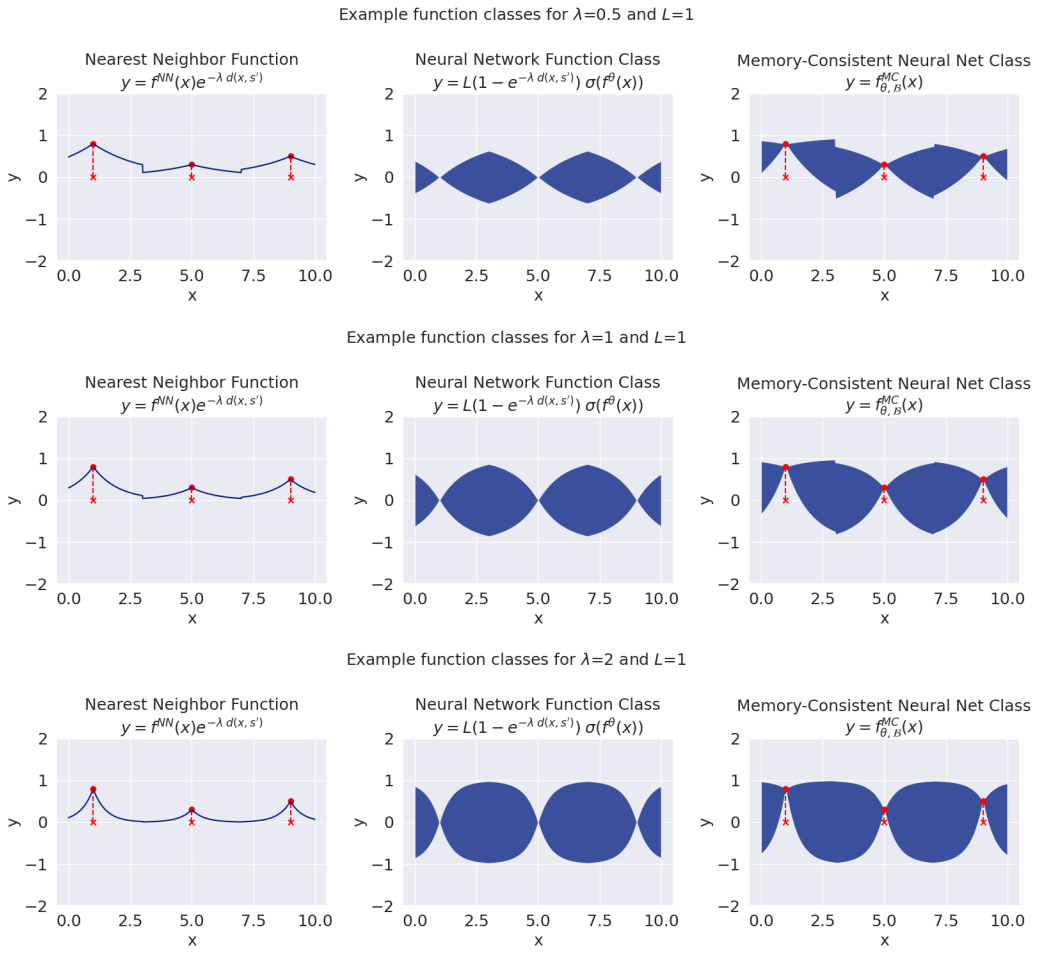

Figure 9: Effects of varying λ (keeping L fixed) on the MCNN function class.

A Examples of the MCNN Function Classes

We demonstrate the effects of varying λ and L in Figures 9 and 10 respectively. By increasing L, we directly increase the width of the model class. By increasing λ, we quicken the interpolation from the nearest neighbor components to the neural network function class. Similarly, by decreasing λ, we slow this transition.

B Detailed Related Work

In imitation and reinforcement learning, non-parametric methods such as RBFs [26], SVMs [13], and nearest neighbours [30] have shown competitive performance on various robotic control benchmarks. But, only recently, a combination of neural networks for representation learning and k-nearest neighbors for control was proposed in Visual Imitation through Nearest Neighbors (VINN) [18]. This is the closest paper to our work and in Section 5, we compare with VINN and demonstrate that we outperform their Euclidean kernel weighted (k) nearest neighbors comprehensively. Other semi-parametric methods include SPIN (Semi-Parametric Inducing point Networks) [27] and Self-Attention between Datapoints [10]. These works propose to use parametric models to approximate non-parametric methods. Other methods like memorizing transformers [35] and Unlimiformer [1] combine non-parametric k nearest neighbors and a parametric attention layer. Here, the kNN is

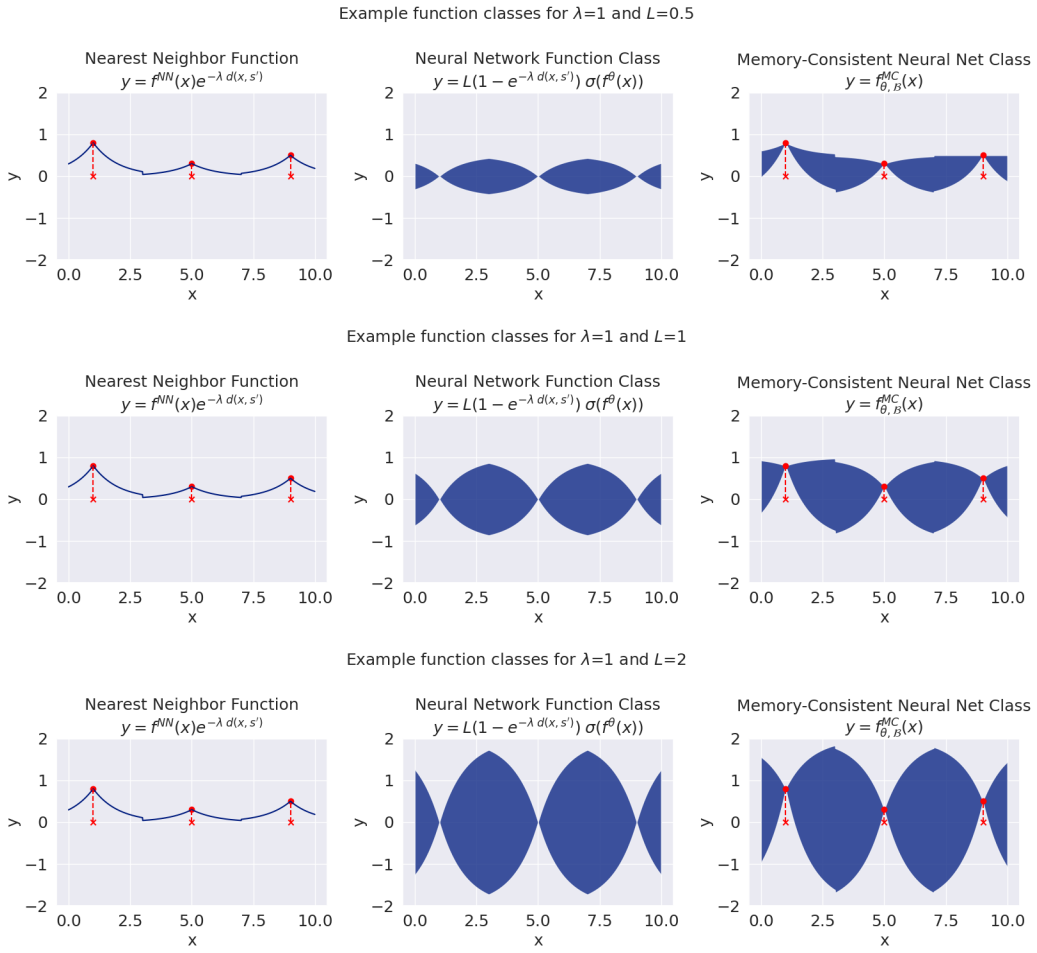

Figure 10: Effects of varying L (keeping λ fixed) on the MCNN function class.

only used to find the top-k inputs for each layer from an input sequence of size greater than k. Our work instead smoothly interpolates between non-parametric and parametric models and demonstrates an improved generalization capability. Finally, our theorem on the sub-optimality gap in imitation learning with MCNNs builds on earlier work on reductions for imitation learning in [28, 22, 2, 23]. It also leverages intuitions from [17] on bounding the width of the model class, which was useful in a model-based RL setting.

Previous work has also explored methods to learn a codebook of (possibly latent) prototypes for physics-constrained learning [31], interpretable OOD detection [36, 9], robust classification [4, 32], motion prediction [37], and image reconstruction [34, 5]. The closest related usage of memories that are representative of the topology of the input space is in [31]. But, here, external information in the form of physics and medical constraints plays a key role in enforcing constraints at these pivotal points. In this paper, our prior is simply that of consistency with no external information utilized.

Finally, the challenge of compounding errors in imitation learning have previous been tackled by permitting online experience [8, 25], reward labels [19], queryable experts [29], or modifying the demonstration data collection procedure [12]. Similarly, in offline RL, a growing body of work deals with conservatism in learning [11, 16, 14, 38, 39]. Instead, in our work, we retain the conveniences of the plain BC setup and focus on designing a model class that encourages better behavior beyond the training data, which in turn boosts task performance in a variety of environments and tasks.

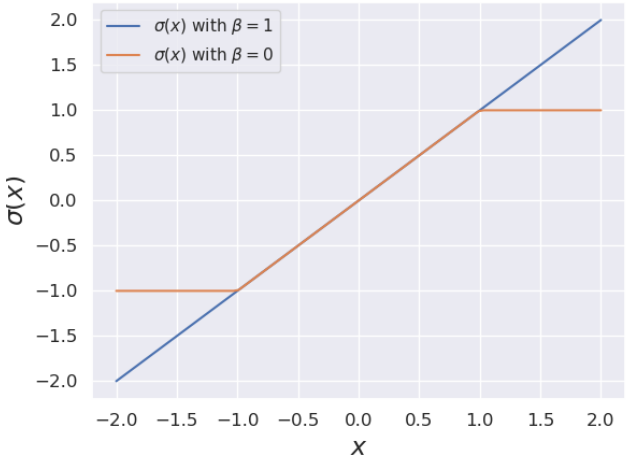

Figure 11: Dynamic $\tanh$-like activation function $\sigma_\beta(x)$ shown for $\beta = 0$ and $\beta = 1$.

C Our Codebase and Compute

We open-source our code at the following link: https://github.com/kaustubhsridhar/MCNN. We ran the experiments on either two Nvidia GeForce RTX 3090 GPUs (each with 24 GB of memory) or two Nvidia Quadro RTX 6000 GPUs (each with 24 GB of memory). The CPUs used were Intel Xeon Gold processors @ 3 GHz.

D Detailed Experimental Setup

Baseline details: We set $k = 10$ in the Euclidean weighted k nearest neighbors algorithm in VINN. This value was recommended by the original paper [18].

Normalization of inputs: We normalized all observations by subtracting the mean and dividing by standard deviation. We didn't have to normalize the actions as they are already in the range $[-1, 1]$ but if we are learning transition models instead, the outputs (next state or reward) would have to be normalized.

Architecture and training details: We use an MLP with two hidden layers (three total layers) of size $[256, 256]$ for Adroit tasks and $[1024, 1024]$ for CARLA. We use an Adam optimizer for BC with and without memories with a starting learning rate of $3e-4$ and train for 1 million steps. We simply minimize the mean squared error for training the policies. We use a batch size of 256 throughout.

Our $\tanh$-like dynamic activation function: We plot our activation function $\sigma_\beta(.)$ described in Algorithm 2 in Figure 11. We use the standard $tanh$ activation function for our reimplementation of BC.

MCNN hyperparameters: We use a value of $L = 1.0$ for all runs. This is suitable for BC because our actions in the range of $[-1, 1]$. We also show a comparison of normalized scores with both number of memories (2.5%, 5%, and 10%) and λ (0.1, 1.0, 10.0, 100.0) values for each task in the 3D bar charts of Figure 12. We tabulate the values for the fixed hyperparameters of $\lambda = 0.1$ and 10% memories under the 'MCNN+BC with Fixed Hyperparameters' columns in both Table 2 (rightmost column) and Table 3 (first column). We also tabulate the highest value of normalized score amongst all runs in the 3D bar charts for each task in the 'MCNN+BC with Tuned Hyperparameters' columns of Table 3 (rightmost column). For all other BC hyperparameters, we use the recommended values for the td3bc implementation at Sun [33].

E Additional Figures and Detailed Results Tables

We tabulate all values from our previous bar charts below. We also plot a comparison of normalized scores with varying number of memories (2.5%, 5%, and 10%) and λs (0.1, 1.0, 10.0, 100.0) for each task in the 3D bar charts of Figure 12.

Table 2: Comparison of normalized scores (across 20 evaluation trajectories and 3 random seeds) between baselines (BC from Fu et al. [7], 1 Nearest Neighbour, Visual Imitation with (k) Nearest Neighbours [18]), our reimplementation of BC, and our method (MCNN+BC) on the Adroit and CARLA domains. Our method outperforms the baselines in all adroit tasks and in one of two CARLA tasks with fixed hyperparameters across all tasks ($\lambda = 0.1$, $L = 1.0$, 10% memories).

Task Name	Baselines			Ours	
	BC [7]	**1-NN**	**VINN** [18]	**BC** (ReImpl.)	**MCNN + BC** (Fixed Hypers)
pen-human-v1	34.4	63.27 ± 7.63	46.77 ± 5.10	58.68 ± 7.14	**94.26** ± 11.71
pen-expert-v1	85.1	100.83 ± 9.23	102.68 ± 2.94	112.69 ± 9.67	**129.20** ± 7.63
pen-cloned-v1	56.9	**60.60** ± 4.96	60.81 ± 9.18	57.36 ± 12.43	**70.86** ± 2.75
hammer-human-v1	1.5	1.60 ± 0.15	0.33 ± 0.08	2.02 ± 1.63	**4.11** ± 0.82
hammer-expert-v1	125.6	79.15 ± 5.89	82.84 ± 7.73	107.01 ± 15.32	**124.74** ± 2.93
hammer-cloned-v1	**0.8**	0.52 ± 0.13	0.34 ± 0.06	0.33 ± 0.10	0.32 ± 0.04
relocate-human-v1	0.0	**0.04** ± 0.01	**0.04** ± 0.00	**0.03** ± 0.04	**0.03** ± 0.02
relocate-expert-v1	101.3	25.97 ± 6.33	30.40 ± 2.89	105.34 ± 1.66	**107.84** ± 1.10
relocate-cloned-v1	-0.1	-0.05 ± 0.05	-0.06 ± 0.01	-0.06 ± 0.01	**-0.04** ± 0.01
door-human-v1	0.5	1.07 ± 0.29	0.61 ± 0.32	0.13 ± 0.08	**1.74** ± 0.51
door-expert-v1	34.9	101.39 ± 0.81	104.88 ± 0.09	**105.21** ± 0.13	**105.18** ± 0.01
door-cloned-v1	-0.1	**-0.08** ± 0.02	**-0.07** ± 0.00	**-0.10** ± 0.01	**-0.10** ± 0.01
carla-lane-v0	31.8	34.03 ± 2.95	29.09 ± 3.96	35.03 ± 3.00	**43.14** ± 3.40
carla-town-v0	**-1.8**	-13.45 ± 7.00	-7.85 ± 4.00	-14.95 ± 5.00	-15.52 ± 8.20

Table 3: Comparison of normalized scores (across 20 evaluation trajectories and 3 random seeds) between our method with fixed hyperparameters across all tasks ($\lambda = 0.1$, $L = 1.0$, 10% memories) and with hyperparameters tuned online. **This table simply demonstrates the ability of our approach to significantly improve with limited online interaction (20 episodes) for hyperparameter tuning.**

Task Name	**MCNN + BC** (Ours)	
	Fixed Hyperparams	Tuned Hyperparams
pen-human-v1	94.26 ± 11.71	**111.01** ± 11.00
pen-expert-v1	129.20 ± 7.63	**132.70** ± 6.55
pen-cloned-v1	70.86 ± 2.75	**91.38** ± 4.00
hammer-human-v1	**4.11** ± 0.82	**4.11** ± 0.82
hammer-expert-v1	124.74 ± 2.93	**127.49** ± 3.00
hammer-cloned-v1	0.32 ± 0.04	**0.92** ± 0.47
relocate-human-v1	**0.03** ± 0.02	**0.04** ± 0.02
relocate-expert-v1	**107.84** ± 1.10	**107.84** ± 1.10
relocate-cloned-v1	-0.04 ± 0.01	**-0.00** ± 0.01
door-human-v1	1.74 ± 0.51	**2.23** ± 1.00
door-expert-v1	105.18 ± 0.01	**105.26** ± 0.24
door-cloned-v1	-0.10 ± 0.01	**-0.07** ± 0.03
carla-lane-v0	43.14 ± 3.40	**45.59** ± 4.71
carla-town-v0	**-15.52** ± 8.20	**-13.70** ± 9.90

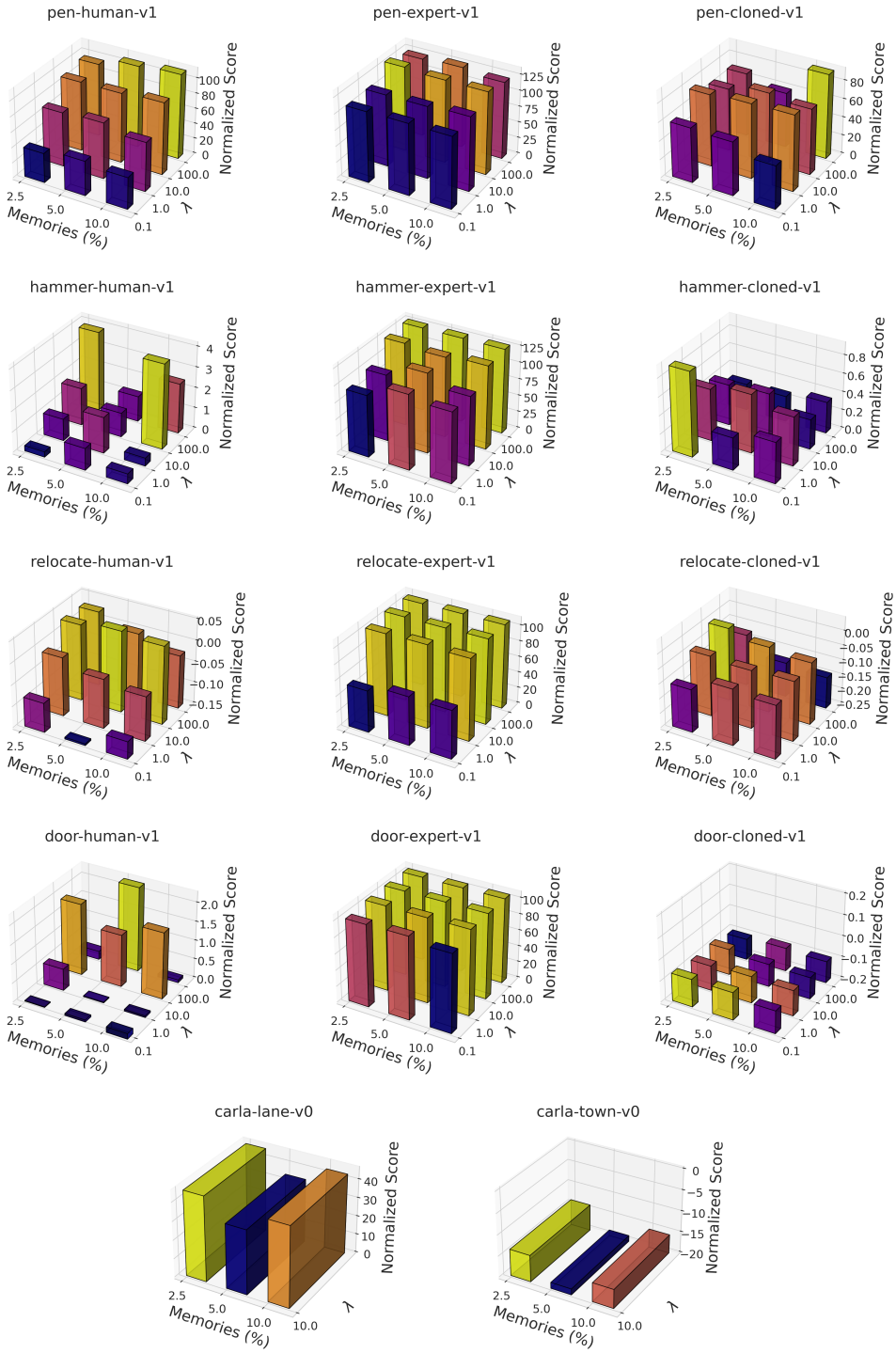

Figure 12: Bar chart of normalized scores for our method against various combinations of λ and memories for each task. Each bar represents the average across 20 evaluation trajectories and three seeds. We notice that the best performance can be obtained for λ values at the middle, *i.e.*, $\lambda \in \{0.1, 1.0\}$, for any number of memories. This is where our method can interpolate, by design, between the nearest memories and vanilla BC. These plots also demonstrate that by training MCNNs on offline data for a few sets of hyperparameters and simply choosing the best hyperparameter with limited online interaction, we can obtain significant improvements in performance.