Discontinuity-Sensitive Optimal Control Learning by Mixture of Experts

Gao Tang
Department of Mechanical Engineering and Material Science
Duke University
Durham, North Carolina 27708
Email: gao.tang@duke.edu

Kris Hauser
Department of Electrical and Computer Engineering
Duke University
Durham, North Carolina 27708
Email: kris.hauser@duke.edu

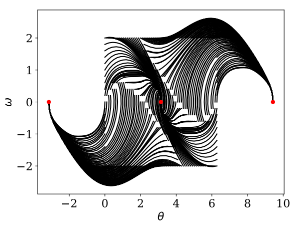
***Abstract*—This paper proposes a discontinuity-sensitive approach to learn the solutions of parametric optimal control problems with high accuracy. Many tasks, ranging from model predictive control to reinforcement learning, may be solved by learning optimal solutions as a function of problem parameters. However, nonconvexity, discrete homotopy classes, and control switching cause discontinuity in the parameter-solution mapping, thus making learning difficult for traditional continuous function approximators. A mixture of experts (MoE) model composed of a classifier and several regressors is proposed to address such an issue. The optimal trajectories of different parameters are clustered such that in each cluster the trajectories are continuous function of problem parameters. Numerical examples on benchmark problems show that training the classifier and regressors individually outperforms joint training of MoE. With suitably chosen clusters, this approach not only achieves lower prediction error with less training data and fewer model parameters, but also leads to dramatic improvements in the reliability of trajectory tracking compared to traditional universal function approximation models (e.g., neural networks).**

I. Introduction

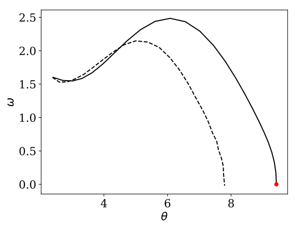
Nonlinear Optimal Control Problems (OCPs) are critical to solve to obtain high performance in many engineering applications. For example, model predictive control (MPC) requires an OCP being solved in every control loop [1], while kinodynamic motion planners rely on solving OCPs between sampled states [3]. However, they are generally difficult to solve to global optimum quickly and with high confidence due to inherent nonconvexity. This has led to an intense interest in using learning to obtain approximations of optimal control policies, either using supervised learning [9, 12] or reinforcement learning [13].

In this paper, we highlight the problem that function approximators such as standard neural networks (SNN) perform poorly near discontinuities that are prevalent in many nonlinear OCPs. Fig. 1 shows the results of using a multilayer SNN to learn a pendulum swingup task from optimal trajectories. The optimal trajectories have three possible goal states so the parameter-solution mapping is discontinuous. Although neural networks are quite useful for approximating nonlinear functions [7], near the region where the optimal goal state switches, their prediction tends to predict a final state that interpolates between two goal states.

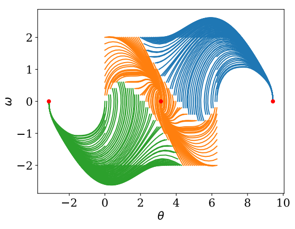
This paper addresses this problem by modifying the Mixture of Experts (MoE) [8, 11, 17] model to learn the solutions to parametric OCPs. The model structure uses a classifier (gating network) to select a regressor (expert) which makes the final prediction (Fig. 2). We intend to train a model such that each regressor works in a region of the parameter space where the parameter-solution mapping is continuous. This is reminiscent of a divide and conquer approach, which has already been widely used in control community for controller design [16]. Fig. 1 illustrates that the pendulum swingup dataset can be divided into three regions, and by classifying them and approximating them separately, MoE makes better prediction than SNN particularly near discontinuity.



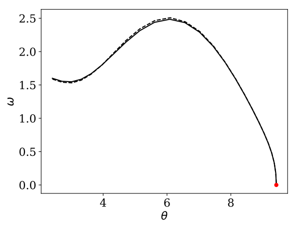
(a) Samples of data



(b) SNN Prediction



(c) Samples of clustered data



(d) MoE Prediction

Fig. 1: Illustration of dataset and prediction of a selected state from SNN and MoE for the pendulum swingup task. (a) samples of optimal pendulum swingup trajectories from different initial states. The red circles are possible target states. (b) prediction of a selected state by SNN that is trained using data in (a). The solid and dashed lines denote the optimal and predicted trajectories, respectively. (c) samples of clustered optimal trajectories where each color denotes one cluster. Trajectories are clustered according to final state. (d) prediction by MoE to the same state as (b).

Considerable care must be taken during MoE training.

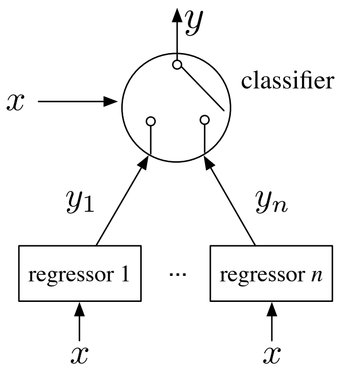


Fig. 2: Illustration of MoE. The classifier selects a model which makes the final prediction.

Although MoE is generally trained using backpropagation [17] or expectation maximization [11], training can be unstable. We propose an approach specially designed for parametric OCPs. The training set consists of solutions to a sampling of parametric OCPs, and we first partition the data into several clusters. Then the classifier is trained to predict the identity of the partition and a separate regressor is trained for each partition. Each component is trained individually using backpropagation. Interestingly, although joint training leads to a model with lower prediction error (loss), it tends to *worsen* trajectory tracking success rate. Moreover, clustering the dataset appropriately is nontrivial and it is fundamental to our approach. Rather than using general methods of input partitioning [18], we propose certain features of optimal trajectories that tend to work well empirically.

Experiments on toy underactuated control problems and agile vehicle control problems demonstrate that suitably trained MoE models can learn near-optimal trajectories suitable for trajectory tracking with remarkably high success rates (99.5+%).

II. Related Work

Nonconvex OCP is generally difficult to solve to global optimum, despite much work to enlarge the convergence domain, e.g., [10]. Moreover, numerical trajectory optimization [2] techniques are, in general, too computationally expensive for highly reactive motions.

As a result, machine learning approaches have been proposed to solve OCPs approximately but in real-time. Reinforcement learning learns the optimal policy by interacting with the environment, and deep neural network policy approximators have been shown to solve complex control problems [13]. Another approach uses supervised learning to learn from precomputed optimal solutions to solve novel problems, and has seen successful application in trajectory optimization [9, 19, 20] and global nonlinear optimization [6]. In [9] precomputed optimal motions are used in a regression to predict trajectories for novel situations to speed up subsequent optimization. In [19] the nearest-neighbor optimal control (NNOC) method is proposed, with a multiple restart method proposed to handle discontinuities. In both these works, the techniques work faster than optimizing from scratch, but still require some amount of optimization for their predicted trajectories. This paper also learns optimal trajectories instead of optimal policies, which has the advantage that trajectories can be tracked using a stabilizing feedback controller to handle model uncertainties and disturbances. It should be noted that the predicted trajectory might not fully satisfy the system dynamics constraints. However, if learning is sufficiently accurate, then this should not be an issue because a feedback controller can correct for such violations.

The discontinuity of the solutions to parametric OCPs as a function of problem parameters has long been known [4], a fact that has been underappreciated in the control learning community. Under certain assumptions, this function is piecewise continuous, and discontinuity-tolerant methods have been proposed for learning from optimal solutions [6, 19]. However, their approaches do not explicitly try to partition the space into regions. In contrast, the discontinuity-sensitive approach proposed here does indeed segment the dataset according to estimated discontinuities.

The most related work is previous research on MoE [11, 17, 18]. This paper proposes several modifications to MoE make it suitable for learning optimal control. We use hard classification boundaries to avoid predicting an average of both sides, and we also modify the training approach. Traditionally MoE is trained using either backpropagation [17] or expectation maximization [11] so the gating function and experts are both updated. However, we train the classifier and regressors individually, and experiments suggest that this is fundamental to achieving high trajectory tracking accuracy.

III. Problem Formulation

In this section, the problem of learning from optimal control is formulated and the key components are analyzed. The proposed approach first forumlates a parametric OCP and then performs the following procedure:

1) Input: collect dataset of solutions to parametric OCPs on sampled parameters.
2) Cluster: select a clustering approach to cluster the trajectories and partition the parameter space.
3) Train: weights of classifier and regressors are trained individually using backpropagation.
4) Validate: predict optimal trajectories for novel states and validate the learned model by trajectory rollout.

A. Parametric Optimal Control

A system is governed by dynamical equations

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) \tag{1}$$

where t is time; $\boldsymbol{x} \in \mathbb{R}^n$ is the state variable; $\boldsymbol{u} \in \mathbb{R}^m$ is the control variable; $\boldsymbol{p} \in \mathbb{R}^l$ is the problem parameters and captures the variability of studied problems. The vector $\boldsymbol{p}$ may specify the initial state, model parameters, and modifications to costs or constraints. We use subscript 0 and f to denote the variables at initial and final time, respectively. The goal is to control the system from some state $\boldsymbol{x}_0$ to some state $\boldsymbol{x}_f$ while minimizing the cost function

$$J = \varphi(t_0, \boldsymbol{x}_0, t_f, \boldsymbol{x}_f, \boldsymbol{p}) + \int_{t_0}^{t_f} L(t, \boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}) \tag{2}$$

where φ only depends on initial and final states; L depends on state and control variables within $[t_0, t_f]$. Practical OCPs may have state, control, and terminal set constraints that have to be satisfied and we refer to [2] for details.

Parametric OCP is generally difficult to solve analytically [14], but for any given parameter, numerical methods may be used to solve the resulting OCP [2]. In this work we employ a direct transcription method, which transforms the OCP into a nonlinear optimization problem and solves it using SNOPT [5]. The solution trajectory is a sequence of state and control variables along a time grid, denoted as $\boldsymbol{z} \equiv \{t_i; bx_i; \boldsymbol{u}_i\}_{i=0}^{N}$ where N is the grid size for discretization. Stacking the element of $\boldsymbol{z}$ into a vector, our goal is to approximate the map from problem parameters to optimal trajectories $\boldsymbol{z}^\star(\boldsymbol{p})$.

B. Optimal Trajectory Database Generation

To train and test models we generate a database of optimal trajectories $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_M$ to sampled problems $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_M \in \mathbb{R}^l$. Due to non-convexity, even finding a global optimum to a single problem can be difficult. One practical approach is to pick the best local optimum from a multi-start method. However, the local optimum can be also quite difficult to find if an initial guess not close to the optimum is provided. We adopt a nearest-neighbor approach [19] to help generate large databases quickly. We first sample some number of problems (fewer than M but much larger than the number of expected partitions) and use an exhaustive random restart approach to solve them. These solutions are used as the initial database. Then we sample more parameters, and for each new problem we attempt local optimization from each of its k-nearest neighbors to find k local optima. The best solution is kept in the database. We note that this process is done completely offline and parallelizable.

C. Mixture of Experts

The MoE model is composed of a classifier and r regressors, as shown in Fig. 2. In this paper both models are chosen as multilayer perceptrons (MLP). The goal is to learn a function $z : \mathbb{R}^l \to \mathbb{R}^R$ that approximates $\boldsymbol{z}(\boldsymbol{p})$ where R is the length of vector $\boldsymbol{z}$. Each regressor takes input $\boldsymbol{p} \in \mathbb{R}^l$ and makes a prediction $y_i(\boldsymbol{p}, w_i) \in \mathbb{R}^R, i = 1, \ldots, r$ where w_i specifies the weights of each regressor. The classifier, with weights w_c, takes input $\boldsymbol{p}$ and predicts r values $\{c_i\}_{i=1}^r$. The output of the classifier are combined with softmax to assign probabilities for each model, i.e.

$$P_i = \frac{\exp c_i}{\Sigma_{i=1}^n \exp c_i} \tag{3}$$

or argmax to select one model only (in this case, $P_k = 1$ for $k = \arg\max_i c_i$ and $P_k = 0$ otherwise.) The difference between softmax and argmax is softmax tends to give a prediction that is a mixture of predictions from all experts. Argmax, however, selects one model and ignores other models' predictions.

In either case, the ultimate prediction is a mixture of predictions from all regressors, i.e.

$$z(\boldsymbol{p}) = \Sigma_{i=1}^n P_i(\boldsymbol{p}, w_c) y_i(\boldsymbol{p}, w_i) \tag{4}$$

The target is to find w_c and $\{w_i\}_{i=1}^r$ in order to miminize

$$L = \mathbb{E}_{\boldsymbol{p} \sim P_{\text{data}}} \text{loss}(z(\boldsymbol{p}), \boldsymbol{z}^\star(\boldsymbol{p})) \tag{5}$$

where P_{data} is a distribution over problems and $\text{loss}(\cdot,\cdot)$ is any regression loss function.

The most straightforward way train MoE is to treat it as an SNN, randomly initialize weights, and miminize (5) using backpropagation. Although several heuristics have been proposed to train MoE using backpropagation such as [17], training may still be unstable. If softmax is used, all the data is used to train each regressor, with weights equal to the probabilities predicted by the classifier. In the case of argmax, each regressor is only trained using data assigned to it by the classifier. There is no gradient for the classifier to update its weights if argmax is used. Softmax, on the other hand, can still have gradient to update the weights of the classifier.

To perform joint training, since argmax is the limit of softmax if we scale $\{c_i\}_{i=1}^r$ by a large positive scalar, we introduce $\varepsilon \in [0, \infty)$ which is used to divide the output of the classifier before applying softmax, i.e.

$$P_i = \frac{\exp(c_i/\varepsilon)}{\Sigma_{i=1}^n \exp(c_i/\varepsilon)}. \tag{6}$$

As $\varepsilon \to 0$, the softmax weights approach the argmax function. Hence, ε must be gradually lowered to balance between updating weights of classifier and restricting mixture of outputs from multiple regressors. As we shall show later, joint training of MoE may improve the loss function compared to decoupled training, but appears to be detrimental to trajectory tracking performance.

D. Parameter Space Partition

Clustering has been shown to be effective to avoid some instability in MoE training [18] by training the classifier and regressors of MoE individually on subsets of the data. We adopt the same approach here, and study how to partition parameter space such that in each region the parameter-solution mapping is continuous.

The dataset $\{(\boldsymbol{p}_j, \boldsymbol{z}_j)\}_{j=1}^M$ is divided into r groups $C_1, \ldots, C_r$, ideally so that $\boldsymbol{z}^\star(\boldsymbol{p})$ is a continuous function for all $\boldsymbol{p}$ in a given region. This problem can be formulated as a clustering problem and each cluster denotes a region of the partitioned parameter space. The classifier is trained to predict $P_i(\boldsymbol{p}_j, w_c) = 1$ for all $\boldsymbol{p}_j$ in C_i, and the i'th regressor is trained as usual, restricted to the examples in C_i. We call this process (decoupled) *pretraining*.

Parametric OCPs have rich features that can be used to find appropriate clusters. We note that this partition cannot be done simply using problem parameters only since the target is to find the discontinuity in the solutions. Discontinuity comes from switching from one family of local optima to another. Hence, although the objective function value and the problem parameters at these discontinuities is similar, the trajectory may not. For example, a car might reverse first or move forward first, and a quadcopter might avoid an obstacle from above or below.

Hence, we experiment with using distance between optimal trajectories to classify the family of solution. The simplest approach is to apply standard clustering techniques, such as the k-Means algorithm, on the trajectory vector space. In order to do so, we first normalize the state and control variables to zero mean and unit variance. After choosing a number of clusters k, the k-Means algorithm is run from random initial centers.

Our experiments observe that k-Means is for some problems successful at predicting discontinuities, but can also group trajectories poorly when k is small. On the other hand, when k is large, each cluster contains less training data, causing the regressors to overfit, and making the job of the classifier harder.

We also propose custom clustering criteria that are based on a system designer's intuition and inspection of datasets. As an example, the periodicity of angles is a useful feature when angle is in the state space and optimal trajectories have distinct final angles; in other words, trajectories lie in distinct homotopy classes. This is useful for the pendulum swingup problem as well as the ground vehicle control problem we consider later. Another approach that is applicable is to examine the Lagrange multipliers of constraints at optimal solutions, since they provide rich information about how constraints influence the trajectory's shape. For example, in quadcopter obstacle avoidance the shortest path might go on either side of the obstacle. Hence, the gradient of the active constraint will have different sign.

E. Discussion and Preliminary Experimentation

The usual approach to MoE is to first perform pretraining before (coupled) *retraining* by minimizing (5). The rationale is that pretraining provides a good initialization, but if the data is clustered badly, i.e. in one cluster there is discontinuity, the loss function may be large. Moreover, even if clustering is perfect, a pretrained model does not necessarily minimize (5) due to misclassification. In this section we shall experimentally demonstrate and discuss why this may be a poor approach for parametric OCPs.

We study a toy pendulum swingup task, where the task is to reach the upright position. Details on the system and neural network models are given in Sec. IV-A. We compare on two metrics: 1) test error (smoothed L1 loss) and 2) rollout success rate after trajectory tracking. In trajectory tracking, we simulate trajectory execution under an LQR controller, which compensates for errors dynamic constraint violations. About each state along the predicted trajectory, we compute an LQR solution for a linear dynamics model and a quadratic cost obtained by Taylor expansion. After trajectory tracking is complete, the simulation switches to a stabilizing controller about the origin. If after 5 seconds the norm of the state error is within a certain threshold (0.1) we denote the rollout as a success. (We note that for the car problem, only the first stage is implemented since the final state is not controllable.)

The following variations are considered:

1) SNN vs MoE,
2) MoE with random weights against k-means clustering on trajectories, and against custom clustering, and
3) Retraining vs no retraining.

The SNN is chosen as MLP of size (2, 300, 75), there the first number denotes the size of the input layer, the last number denotes the size of the output layer, and intermediate numbers indicate the size of hidden layers. We experimented with SNN with more hidden layers or more neurons in the hidden layer, but they result in similar or larger test error. Specifically, MLPs of size (2, 50, 20, 75) and (2, 20, 50, 75), (2, 30, 30, 30, 75) yield test errors of 0.258, 0.170, and 0.232, respectively. The size (2, 300, 75) network, on the other hand, has test error of 0.058.

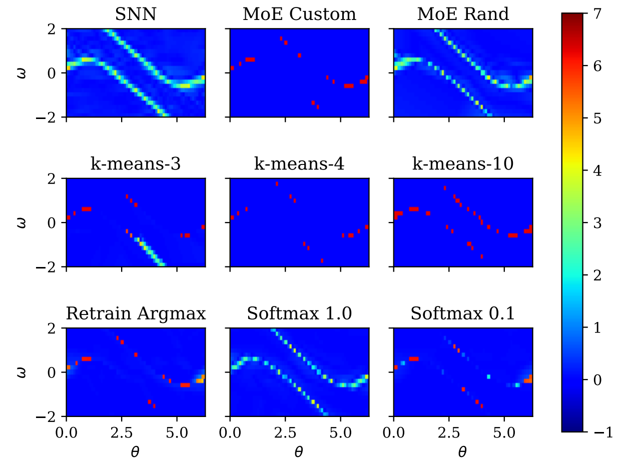
For MoE, the classifier is of size (2, 50, r) and the r regressors are all of size (2, 20, 75). Custom MoE and random weight MoE use 3 experts. The custom clustering divides the data into 3 clusters based on the final angle. We also use k-means with 3, 4, and 10 clusters solely on trajectories with the same design of network size.

Fig. 3.a plots the prediction error on θ_f and Fig. 3.b plots the state error after trajectory tracking. The validation error and rollout success for each model are also listed in Tab. I.

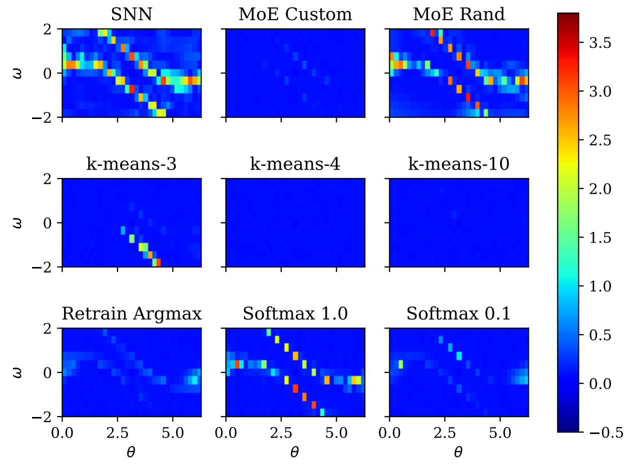
Row 1 shows that SNN has difficulty in making predictions in regions near the discontinuity, averaging between both sides. MoE does also make inaccurate prediction, but these are caused by misclassification and the prediction is a local optimal trajectory belonging to another cluster. Hence, they are suboptimal but still reach the vertical position as desired, since the difference in θ_f is 2π. The suboptimality is not too great, because near the boundaries two families of solutions have similar objective function. MoE trained from random initialization does achive lower prediction error than SNN, but is not very successful. This indicates that training by simply descending (5) is unable to guide the classifier to the appropriate clusters.

Row 2 tests MoE with k-Means and various cluster sizes., which are shown in Fig. 4. $k = 3$ has one cluster that has data from both families of trajectories, so the prediction close to the discontinuity is worse. $k = 4$ and $k = 10$ clusters finds the discontinuity successfully, and the resulting MoE achieves high success rate.

Row 3 of Fig. 3 shows various methods of retraining after pretraining MoE with custom clustering. In all cases this approach decreases regression error but also rollout success rate. In (vii) argmax is used following the output layer of the classifier. The classifier has no gradient to update it self so only the regressors are updated. Due to classification error, the regressors will be trained with trajectories from other clusters. As a result, the prediction near the boundaries will tends towards the average of two clutsers. In (vii) and (ix) we



(a) Prediction error of θ_f



(b) State error after trajectory tracking

Fig. 3: Comparing several models for learning the pendulum swing-up task.

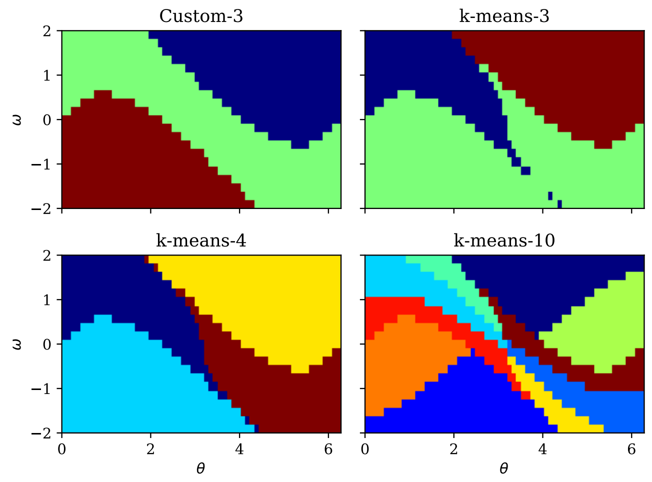


Fig. 4: Choices of clusters for the pendulum problem. Different colors means different clusters. Figures include: 1) custom 3 clusters 2) k-Means with 3 clusters 3) k-Means with 4 clusters 4) k-Means with 10 clusters

use softmax with different ε. In these cases, the classifier is updated but the regressors will predict towards the average. As shown in Tab. I, retraining does decrease the prediction error at the cost of lower rollout success rate.

These experiments suggest that proper clustering is important for MoE training. Moreover, rollout success is a better metric to use in practice, while testing error can be misleading. Due to misclassifications, a lower testing error can be achieved by averaging at discontinuities, but this leads to severe failures. We also observe that coupled retraining is detrimental to performance. This is because the imperfect classification causes the individual regressors to be provided with discontinuous training data, again leading to averaging artifacts.

IV. Numerical Examples

We run experiments on the pendulum task and three dynamic vehicle problems, and the details are given below. Results are summarized in Tab. II. In each case, training sets contained 80% of examples specified in Dataset size, and the testing sets had the remaining 20%. Validation sets (of size Validation size) are generated separately.

SNN test error indicates the testing error when training is terminated. SNN hyperparameters (SNN size) were tuned to achieve low test error. Validation error (SNN/MoE validation) indicates loss on the validation set, while rollout error (SNN/MoE rollout) indicates success rate during trajectory tracking. Except for the car problem, this involves the stabilizing LQR approach described in Sec. III-E. Details on the car rollout success criteria are specified below.

Details on the MoE network design are listed in the rows listing the number of clusters, the resulting cluster sizes, and the network hyperparameters (Classifier/Regressor size). The Regressor Test Error row indicates how well the MoE regressors are fitting on clustered data, showing that each regressor has quite small error when fit on a continuous region.

In all of these experiments, hidden layers use LeakyReLU with $\alpha = 0.2$. The output layer of regressors is a linear layer without nonlinear activation function. The loss function is the smooth L1 loss and cross entropy loss for regressors and classifier, respectively.

A. *Pendulum Swing-up*

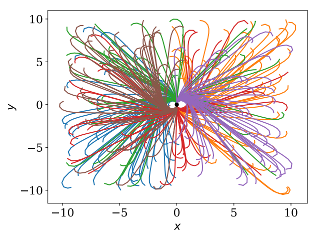
1) Problem Setup: The system dynamic equations are

$$\dot{\theta} = \omega, \dot{\omega} = u - \sin\theta \tag{7}$$

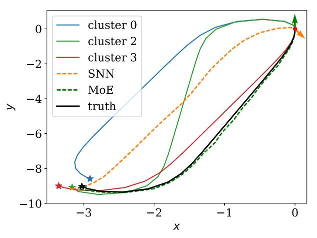
where θ, ω are the angle and angular velocity of the pendulum; $u \in [-1, 1]$ is the control torque. The problem parameters are the initial states. The target state is the straight up state, i.e. $\omega_f = 0, \mod(\theta_f, 2\pi) = \pi$. The cost function is a weighted

TABLE I: Comparison of prediction error and rollout success rate on the pendulum problem

Model	SNN	MoE							
Clustering	—	Custom	Rand.	k-means-3	k-means-4	k-means-10	Custom	Custom	Custom
Retrain	—	—	—	—	—	—	argmax	softmax 1.0	softmax 0.1
Validation error	0.046	0.030	0.035	0.039	0.029	0.051	0.027	0.028	0.026
Success (out of 1000)	717	998	829	970	1000	1000	941	896	969



(a) Samples of optimal trajectories for the car problem



(b) Illustration of poor predictions from SNN

Fig. 5: Left: samples of optimal trajectories. Each color corresponds to one cluster of trajectories. Black circle is the target. Right: A selected state that SNN makes worse prediction than MoE. It also shows states near this state might belong to to three different trajectory clusters. SNN predicts a trajectory with incorrect final angle.

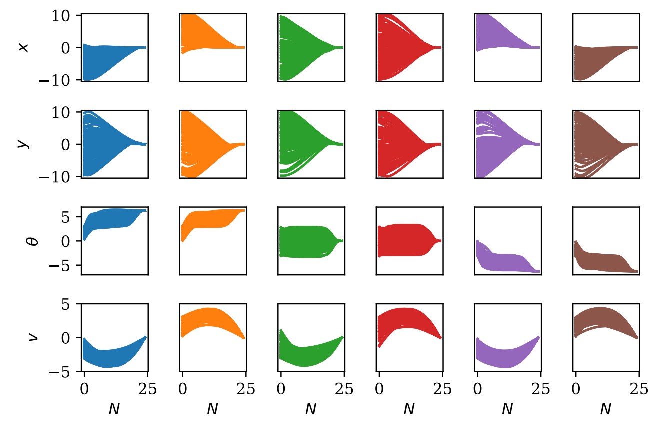


Fig. 6: Samples of trajectories in each cluster for the car problem. Column: different state variables for each cluster. Row: state variable for different clusters.

sum of time and control energy, i.e. $J = w(t_f - t_0) + r\int_{t_0}^{t_f} u^2\,\mathrm{d}t$ with $w = 1, r = 1$.

2) Data Generation and Training: The parameter space is a subset of $\mathbb{R}^2$ and we directly sample parameters on a uniform grid. Specifically, we use a grid size of 61×21. The validation set is sampled at random. Samples of optimal trajectories are shown in Fig. 1. The custom clustering partitions the trajectories by θ_f.

B. Ground vehicle

1) Problem Setup: We use a planar car with dynamic equations

$$\dot{x} = v\sin\theta,\ \dot{y} = v\cos\theta,\ \dot{\theta} = u_\theta v,\ \dot{v} = u_v \tag{8}$$

where the state $\boldsymbol{x} = [x, y, \theta, v]$ includes the planar coordinates, orientation, and velocity of the vehicle; the control $\boldsymbol{u} = [u_\theta, u_v]$ includes the control variables which change the steering angle and velocity, respectively. The problem parameters are the initial states, as listed in Tab. II and the goal is to control the system to the origin with zero velocity and $\mod(\theta_f, 2\pi) = 0$. The cost function is a weighted sum of time and control energy, i.e. $J = w(t_f - t_0) + \int_{t_0}^{t_f} r_1 u_\theta^2 + r_2 u_v^2\,\mathrm{d}t$ with $w = 10, r_1 = r_2 = 1$.

2) Data Generation and Training: The data is generated by uniformly sampling the parameter space. Fig. 5 shows a few samples of the optimal trajectories. Similar to the pendulum swingup problem, the constraint on θ_f makes it possible to reach the goal with different θ_f. The custom clustering is developed by inspection, whereby we first divide the dataset into three groups based on the final angle. Then we find that for trajectories with the same θ_f, the car can either go forward or backward to reach the origin, i.e. with positive or negative velocities. This is illustrated in Fig. 6. Hence, we divide the dataset into 6 clusters. We note that the cluster sizes are bimodal and we use larger regression network for cluster with larger size.

3) Trajectory tracking: Because this problem is not controllable at the origin, a stabilizing LQR controller may not be used at the trajectory endpoint. Instead, we simply perform LQR rollout on the predicted trajectory, and stop when the end time is reached. To determine success, we check if norm of final state error is within 0.5.

4) Results and Discussion: The data in Tab. II show similar trends to the pendulum problem, in particular, MoE yields lower validation error and higher rollout success rate than SNN. Moreover, the custom clustering outperforms k-Means which further outperforms SNN. In Fig. 5 we show the predictions from SNN and MoE on a selected parameter as well as the optimal trajectories of its neighbors. It is clearly shown that SNN may fail to predict θ_f correctly.

The histogram in Fig. 7.a shows the norm of the error in predicted final state, indicating that SNN has higher prediction error. Fig. 7.b also show that paths predicted by SNN violates system dynamics more than MoE. The reason why tracking error is actually much larger than predicted is that the predicted trajectory violates system dynamics, so path tracking diverges.

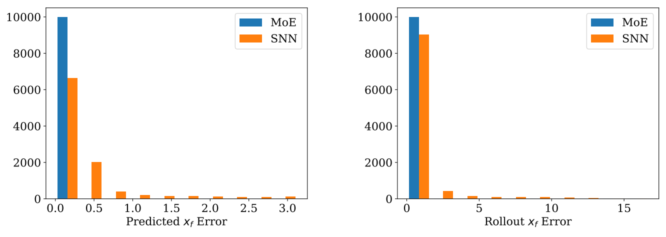
C. Quadcopter with Collision Avoidance

1) Problem Setup: The system has state $\boldsymbol{x} = (x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, p, q, r) \in \mathbb{R}^{12}$ and control $\boldsymbol{u} \in \mathbb{R}^4$.

TABLE II: Summary of experimental results for SNN and MoE

	pendulum	ground vehicle		quadcopter	quadcopter-obstacle	
State dims	2	4		12	12	
Control dims	1	2		4	4	
Problem param.	$\boldsymbol{x}_0 \in \mathbb{R}^2$	$\boldsymbol{x}_0 \in \mathbb{R}^4$		initial position, $\mathbb{R}^3$	initial position and obstacle, $\mathbb{R}^7$	
Param range	$[-\pi,\pi]\times[-2,2]$	$[-10,10]^2\times[-\pi,\pi]\times[-3.1,3.1]$		$[-10,10]^3$	$[-10,10]^6\times[1,5]$[a]	
Dataset size[†]	1281	120009		9000	616758	
Validation size	1000	10000		1000	10000	
SNN size	(2, 300, 75)	(4, 200, 200, 149)		(3, 200, 317)	(7, 1000, 1000, 317)	
SNN test error	0.058	0.045		8.6×10^{-5}	0.014	
SNN validation	0.046	0.046		4.7×10^{-5}	0.024	
SNN rollout	717/1000	6729/10000		1000/1000	-0.315 [b]	
# clusters	3	6		4	8	
cluster approach	custom	custom	k-means	k-means	custom	kmeans
Cluster size range	[388,505]	[7266,45626]	[7228, 28913]	[2072,2356]	[70474,84280]	[64682, 101669]
Classifier size	(2, 50, 3)	(4, 200, 6)	(4, 200, 6)	(3, 50, 4)	(7, 200, 200, 8)	(7, 200, 200, 8)
Test accuracy	97.2%	98.7%	97.7%	99.6%	88.9%	96.4%
Regressor size	(2, 20, 75)	(4, 200, 149) for small clusters (4, 500, 149) for large	(4, 200, 149) for small clusters (4, 300, 149) for large	(3, 50, 317)	(7, 200, 200, 317)	(7, 200, 200, 317)
Regressor test error	0.0032 ± 0.0031	0.0018 ± 0.0014	0.0088 ± 0.0082	$4.6\times10^{-5} \pm 9\times10^{-6}$	0.0022 ± 0.0003	0.0052 ± 0.0027
MoE validation	0.030	0.019	0.031	4.6×10^{-5}	0.015	0.016
MoE rollout	998/1000	9975/10000	9413/10000	1000/1000	-0.043[b]	-0.167[b]

[a] The obstacle is sampled such that it always collides with optimal obstacle-free trajectory
[b] Average of the largest constraint violations based on trajectory rollout. All states can be controlled to the target. See histogram in Fig. 11 for distribution.



(a) Prediction error of $\boldsymbol{x}_f$ (b) Tracking error of $\boldsymbol{x}_f$

Fig. 7: Histograms of prediction and tracking results for MoE and SNN on the car problem.

We refer [15] to the details. The goal is to control the quadcopter from any equilibrium state with position within $[-10,10]^3$ and all other states zero to the goal state $\mathbf{0}$. The cost function is a weighted sum of time, control energy, and penalty on states, i.e. $J = w(t_f - t_0) + \int_{t_0}^{t_f} \boldsymbol{x}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{u}\,\mathrm{d}t$ with $w = 10$, $\boldsymbol{Q} = \mathrm{diag}(0,0,0,1,1,1,0.1,0.1,0.1,1,1,1)$, $\boldsymbol{R} = \mathrm{diag}(1,1,1,1)$.

The quadcopter-obstacle case imposes additional path constraints on the state variables. The obstacle is a sphere with different position and radius, and obstacles are randomly placed in space with radius within $[1,5]$. We are interested in how the obstacles influence the trajectory.

2) Data Generation and Training: In the obstacle-free case, initial positions are sampled at random, and k-Means is used for clustering.

The obstacle problem is more challenging because it has higher dimensionality in parameter space (7). The OCP is also more challenging to solve due to the non-convex of obstacle

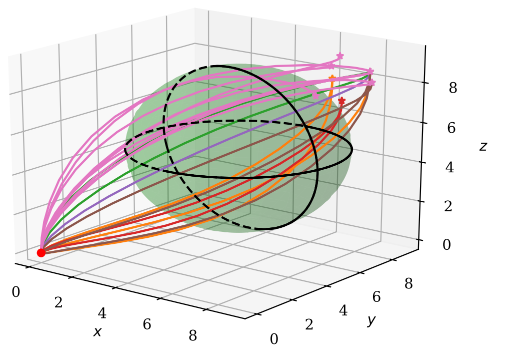


Fig. 8: Trajectories of problems with parmameter close to the problem with sphere at (4, 4, 4), radius 3 and initial position (8, 8, 8). Each color corresponds to trajectories from one cluster. It shows the trajectories can be quite different even for close problem parameters.

avoidance constraint. We want to focus on problem instances with significant obstacles, so our dataset only includes examples where the optimal collision-free trajectory would collide with an obstacle. To generate this dataset, we collect obstacle free trajectories and then sample obstacles that collide with the trajectory. We then re-optimize for the sampled obstacles. Samples of trajectories are shown in Fig. 8 and Fig. 9.

The discontinuity of the parameter-solution mapping in this problem is avoiding the obstacle from different directions

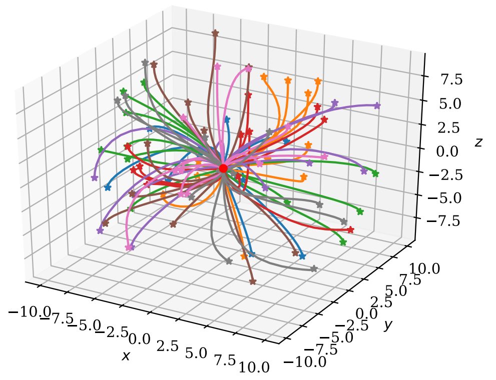


Fig. 9: Samples of optimal trajectories for the quadcopter problem. Each color corresponds to each trajectory cluster.

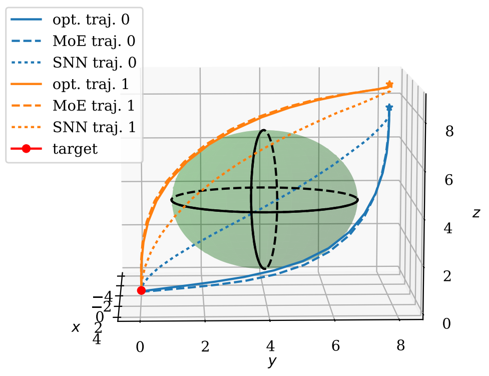


Fig. 10: Optimal trajectories and prediction from SNN and MoE for two selected close states. The green sphere is an obstacle centered at (0, 4, 4) with radius of 3. The solid, dashed and dotted lines are the optimal trajectories, prediction of MoE, and prediction of SNN, respectively. It shows SNN predicts a trajectory that violates obstacles avoidance constraints.

outperforms others and vice versa. One feature that describes how the obstacle-free trajectory is affected by the obstacles is the gradient of the active constraints with respect to state variables. Since the obstacles are spheres, the gradient is essentially the vector from the center of the sphere to the point on surface where constraints are active. Its direction clearly shows which direction the trajectory has to change for collision avoidance. For trajectories that has more than one active constraints, we use the multipliers as weights and take the average. In this way, a 3D vector is calculated for each trajectory and used as features to divide the problem space. We divide the dataset into 8 groups based on the sign of each element of the 3D vector.

3) Results and Discussion: Results show that both SNN and MoE control the quadcopter to a stabilizable state in highly reliable fashion without obstacles. Hence, for validation we focus more on the amount of collision avoidance violation, i.e. $\min\{\|\boldsymbol{x}_i - \boldsymbol{c}_o\| - r_o\}_{i=0}^{N}$ where r_o and $\boldsymbol{c}_o$ are respectively the radius and center of the obstacle.

With obstacles, MoE with custom cluster also significantly outperforms others. A histogram of the constraint violation is shown in Fig. 11, indicating that MoE yields much lower violation of constraints than SNN. Fig. 10 shows examples of optimal trajectories and prediction from SNN and MoE. As the initial state moves along z direction, the optimal trajectories turns from going above to going below the obstacle. SNN is unable to handle such discontinuity and predicts a trajectory that violates the constraints. However, MoE is able to detect such discontinuity and predicts the corresponding trajectories. It is important to note however that MoE still creates grazing collisions, so to successfully avoid an obstacle in practice, either a margin of error should be added to the modeled obstacle, or local collision avoidance should be added to the trajectory tracker.

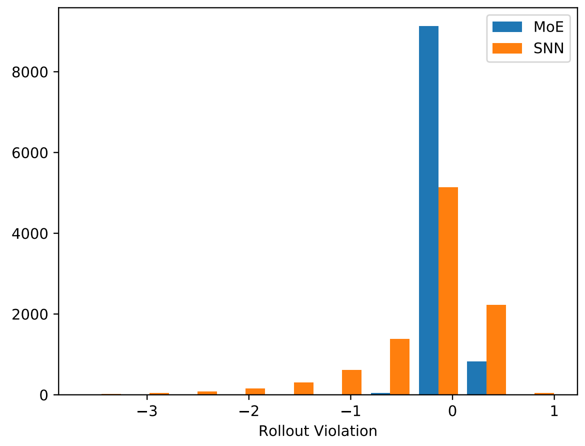


Fig. 11: Rollout constraint violation for quadcopter-obstacle.

V. Conclusion

In this paper we demonstrate that optimal trajectories can be learned with high accuracy if we take into account the special structure of optimal control problems. The mixture of experts model is designed such that each expert approximates a smooth region in the problem optimum map, and the classifier handles discontinuities without averaging. It is important to train MoE with the correct clusters, and curiously, coupled training of the regressors and classifier tends to be detrimental to tracking performance. We also argue that test error is not a good metric to judge learning models, but rather rollout success rate under trajectory tracking control is preferable.

Future work includes developing more sophisticated clustering algorithms that automatically find the best partition strategy. For certain OCPs, differential flatness can be used such that the predicted trajectory satisfies dynamical constraints. Further work also includes how to prove the stability of the predicted trajectories, and to scale up to handle larger problems, e.g., from sensor data or model uncertainties.

References

[1] A Bemporad, M Morari, V Dua, and EN Pistikopoulos. The explicit solution of model predictive control via multiparametric quadratic programming. In *Proc. American Control Conf.*, volume 1–6, pages 872 – 876, 2000.

[2] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[3] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993.

[4] Anthony V Fiacco. Introduction to sensitivity and stability analysis in nonlinear programming. 1983.

[5] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[6] K. Hauser. Learning the problem-optimum map: Analysis and application to global optimization in robotics. *IEEE Trans. Robotics*, 33(1):141–152, February 2017.

[7] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[8] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[9] Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, 34(1-2): 111–127, January 2013.

[10] Fanghua Jiang, Hexi Baoyin, and Junfeng Li. Practical techniques for low-thrust trajectory optimization with homotopic approach. *J. Guid. Control Dynam.*, 35(1): 245–258, 2012.

[11] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[12] Roberto Lampariello, Duy Nguyen-Tuong, Claudio Castellini, Gerd Hirzinger, and Jan Peters. Trajectory planning for optimal robot catching in real-time. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3719–3726. IEEE.

[13] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[14] Helmut Maurer and Dirk Augustin. Sensitivity Analysis and Real-Time Control of Parametric Optimal Control Problems Using Boundary Value Methods. In *Online Optimization of Large Scale Systems*, pages 17–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[15] Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 31(5):664–674, April 2012.

[16] Roderick Murray-Smith and T Johansen. *Multiple model approaches to nonlinear modelling and control*. CRC press, 1997.

[17] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[18] Bin Tang, Malcolm I Heywood, and Michael Shepherd. Input partitioning to mixture of experts. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, volume 1, pages 227–232. IEEE, 2002.

[19] Gao Tang and Kris Hauser. A data-driven indirect method for nonlinear optimal control. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages –. IEEE, 2017.

[20] Teodor Tomić, Moritz Maier, and Sami Haddadin. Learning quadrotor maneuvers from optimal control and generalizing in real-time. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1747–1754. IEEE, 2014.