Discontinuity-Sensitive Optimal Control Learning by Mixture of Experts

Gao Tang and Kris Hauser, *Member, IEEE*

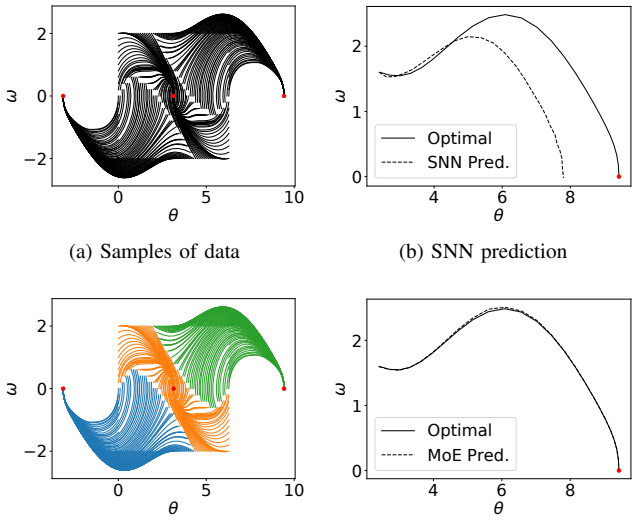
***Abstract*— This paper proposes a machine learning method to predict the solutions of related nonlinear optimal control problems given some parametric input, such as the initial state. The map between problem parameters to optimal solutions is called the problem-optimum map, and is often discontinuous due to nonconvexity, discrete homotopy classes, and control switching. This causes difficulties for traditional function approximators such as neural networks, which assume continuity of the underlying function. This paper proposes a mixture of experts (MoE) model composed of a classifier and several regressors, where each regressor is tuned to a particular continuous region. A novel training approach is proposed that trains classifier and regressors independently. MoE greatly outperforms standard neural networks, and achieves highly reliable trajectory prediction (over 99.5% accuracy) in several dynamic vehicle control problems.**

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 [2]. However, OCPs 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 [3], [4], [5] or reinforcement learning [6], [7].

This paper highlights 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 learning a pendulum swingup task by SNN from optimal trajectories. The optimal trajectories have three possible goal states so the problem-optimum mapping is not globally continuous, so near discontinuities neural networks tend to predict an interpolating result, leading to severe performance degradation (Fig. 1.b).

Our approach addresses this problem by modifying the Mixture of Experts (MoE) [8], [9], [10] 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 for each regressor to work in a region of the parameter space where the problem-optimum mapping is continuous. This is reminiscent of a divide and conquer approach, which has already been widely used for controller design [11]. Fig. 1.c illustrates that the pendulum swingup dataset can be divided into three regions, and by classifying them and approximating them separately, MoE outperforms SNN (Fig. 1.d).



(a) Samples of data (b) SNN prediction

(c) Clustered data (d) MoE prediction

Fig. 1. Learning a pendulum swingup task from (a) samples of optimal trajectories. The red circles are possible target states. (b) Standard neural network (SNN) poorly predicts the solution for a selected state. Solid and dashed lines denote optimal and predicted trajectories, respectively. (c) By clustering optimal trajectories (clusters distinguished by color) and training separate networks for each cluster, (d) MoE predicts the solution trajectory accurately.

Considerable care must be taken during MoE training. Although MoE is generally trained using backpropagation [10] or expectation maximization [9], training can be unstable. We propose a training approach specially designed for learning optimal trajectories for parametric OCPs. We first partition the data into several clusters, and then train the classifier to predict the identity of the partition. A separate expert is trained for each partition. Interestingly, this outperforms joint training as is typically done in MoE, as well as weighted blending of experts [8]. For the clustering method, we found that PCA in trajectory space followed by k-Means performs well given a suitable number of experts.

Experiments on toy underactuated control problems and more challenging agile vehicle control problems, with state space dimension up to 12, demonstrate that suitably trained MoE models can learn near-optimal trajectories suitable for trajectory tracking with remarkably high success rates (99.5+%). In contrast, standard neural networks succeed less than 90% of the time, even given hundreds of thousands of training examples.

*This work is supported by NSF grant #IIS-1816540.

G. Tang is with the Department of Mechanical Engineering and Material Science, Duke University, Durham, NC, 27708 USA e-mail: gao.tang@duke.edu.

K. Hauser is with the Departmeent of Electrical and Computer Engineering, Duke University, Durham, NC, 27708 USA e-mail: kris.hauser@duke.edu.

II. Related Work

Nonconvex OCP is generally difficult to solve to global optimum, despite much work to enlarge the convergence domain, e.g., [12]. Moreover, numerical trajectory optimization [13] 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. RL learns the optimal policy by interacting with the environment, and deep neural network policy approximators have been shown to solve complex control problems [6]. Another approach uses supervised learning to learn from precomputed optimal solutions to solve novel problems, and has seen successful application in trajectory optimization [3], [14], [15] and global nonlinear optimization [16]. In [3] precomputed optimal motions are used in a regression to predict trajectories for novel situations to speed up subsequent optimization. In [14] 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. Optimal trajectories for quadrotors are learned in [5] and applied on a real system for aggressive maneuvers, demonstrating the potential of learning based control. 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, as demonstrated in [5].

The discontinuity of the solutions to parametric OCPs as a function of problem parameters has long been known [17], 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 [16], [14]. 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.

Clustering is a fundamental problem in machine learning and many algorithms such as k-Means have been proposed. Existing literature on trajectory clustering [18] studies trajectories of hurricane or animal movement which are different from the trajectories in this work. To the best of the authors' knowledge, little attention is paid to clustering of optimal trajectories of robotic systems.

The most related work is previous research on MoE [9], [10], [19]. This is the first time MoE is applied to trajectory learning to the best of the authors' knowledge. 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 [10] or expectation maximization [9] so the gating function and experts are both updated. However, we train the classifier and regressors individually, and experiments suggest that this significantly increases trajectory tracking accuracy.

Deep RL has shown success in robot control problems [7] by learning a control policy. Our method is different from Deep RL as the optimal trajectory is directly learned using supervised methods. As a result, the prediction is a trajectory and can be directly combined with trajectory tracking controller for real robotics application, with success in prior work [5]. On the contrary, deep RL learns a policy and has to be directly executed, losing guarantee of stability. Another difference is our method uses to supervised learning where optimal trajectories are precomputed by offline optimization. However, deep RL does not need supervision but the sample efficiency is low. One reason is deep RL has to explore a large proportion of state-action joint space which are actually far from the optimal state-action distribution. Other authors have observed that deep RL has exhibited instability in training and difficulty in reproducibility [20].

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 formulates a parametric OCP and then performs the following procedure:

1) Input: sample OCP parameters and collect dataset of problem-optimum pairs.
2) Cluster: select a clustering approach to cluster the trajectories (it also partitions parameter space).
3) Train: weights of classifier and regressors are trained individually using backpropagation.
4) Validate: predict optimal trajectories for states in the validation set and perform trajectory rollout.

A. Parametric Optimal Control

A system is governed by dynamical equations

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

where t is time; $x \in \mathbb{R}^n$ is the state variable; $u \in \mathbb{R}^m$ is the control variable; $p \in \mathbb{R}^l$ is the problem parameters and captures the variability of studied problems. The vector 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 times, respectively. The goal is to control the system from some state x_0 to some state x_f while minimizing the cost function

$$J = \varphi(t_0, x_0, t_f, x_f, p) + \int_{t_0}^{t_f} L(t, x(t), u(t), p) \tag{2}$$

where φ and L are functions mapping to $\mathbb{R}$. Practical OCPs may have state, control, and terminal set constraints that have to be satisfied and we refer to [13] for details.

In this work we employ a direct transcription method [13] with sparse nonlinear optimization solver SNOPT [21] to

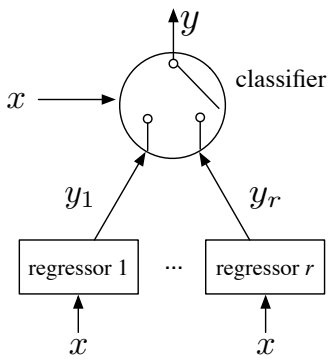


Fig. 2. Illustration of MoE. The prediction is made by *one* out of r regressors selected by the classifier.

solve OCPs. The solution trajectory is a sequence of state and control variables at a time grid. Stacking the trajectory into a vector of length R, our goal is to approximate the mapping from problem parameters p to optimal trajectories, denoted as $z^\star(p) \in \mathbb{R}^R$.

B. Optimal Trajectory Database Generation

To train and test models we generate a database of optimal trajectories $z_1, \ldots, z_M$ to sampled problems $p_1, \ldots, p_M$ where M is the data size. We adopt a nearest-neighbor approach [14] 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. As reported in [14], this approach is computationally efficient and has high probability of finding global optima. We note that this process is done completely offline and is 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 fully-connected neural network. The goal is to learn a function $z : \mathbb{R}^l \to \mathbb{R}^R$ that approximates $z(p)$. Each regressor takes input $p \in \mathbb{R}^l$ and makes a prediction $y_i(p) \in \mathbb{R}^R, i = 1, \ldots, r$. The classifier takes input 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.

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

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 them is softmax predicts a mixture of all experts' predictions. Argmax, however, selects one model and ignores other models' predictions. While softmax has been widely used [8], [9], [22] since argmax has no gradient to update the classifier, our proposed model uses argmax since softmax assign weights to regressors and has the averaging issue.

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

$$z(p) = \Sigma_{i=1}^r P_i(p) y_i(p) \tag{4}$$

The target is to find weights of the classifier and regressors in order to minimize

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

where P_{data} is a distribution over problems and $\text{loss}(\cdot,\cdot)$ is any regression loss function. Another possible cost function is used to encourage specialization of experts instead of cooperation [22]:

$$L = \mathbb{E}_{p \sim P_{\text{data}}} \sum_{i=1}^r P_i(p) \text{loss}(z_i(p), z^\star(p)), \tag{6}$$

which is a weighted average of loss functions over each regressors' outputs. Jacobs et.al. [8] introduced another loss function based on the negative log probability of generating desired outputs

$$L_{\text{regressors}} = \mathbb{E}_{p \sim P_{\text{data}}} - \log \sum_{i=1}^r P_i \exp\left(-\frac{1}{2}(z^\star - z_i)^T (z^\star - z_i)\right) \tag{7}$$

where Gaussian distribution with identity covariance matrix is assumed. These are only used when training regressors, while the cost function for classifier training remains Eq. (5). For each mini-batch of data, regressors and classifier are updated independently. We refer to [22] for details.

Although those two cost functions improve upon the straightforward one, our experiments show they are still unlikely to find correct trajectory clusters purely through backpropagation.

D. Trajectory Clustering

Our training method divides the dataset $\{(p_j, z_j)\}_{j=1}^M$ into r groups $C_1, \ldots, C_r$, ideally so that $z^\star(p)$ is a continuous function for all 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 then trained directly to predict the one-hot encoding of cluster labels.

Although expert knowledge can be used to cluster the trajectories, such as the pendulum problem shown in Fig. 1, a more general approach with minimum human involvement is generally preferred. The simplest approach is to perform standard clustering techniques after dimensionality reduction. In this paper, PCA and k-Means are used for dimensionality reduction and clustering, respectively. These approaches are straightforward to implement and work well empirically. We note that a mini-batch version of k-Means has to be used [23] to scale to larger datasets.

This approach introduces the hyperparameter r of how many clusters to use. The optimal value is high enough to place cluster boundaries along discontinuities in $z(p)$, but not so high as to over-fragment the data. We empirically study how the parameter choice affects the performance of MoE.

IV. Numerical Experiments

Numerical examples demonstrate the effectiveness of our MoE approach several dynamically-constrained optimal control problems. For each problem a dataset and validation set is generated according to the same distribution of problem instances. 80% of the dataset is used for training and the remaining 20% is for testing.

In all experiments, neural networks are used as classifier and regressors. The network topology is described by an array which denotes the size of each layer, from input to output. Hidden layers use the LeakyReLU activation function with $\alpha = 0.2$ and a linear output layer is used. The smooth L1 and cross entropy loss are used for regression and classification, respectively. We use the Adam optimizer with learning rate 0.001 for training.

A. Benchmark Problems Description

We study several benchmark optimal control problems with increasing state space dimensionality. In each problem, the problem parameter is the initial state, while the goal state is fixed. These problems are summarized in Tab. I. All problems are solved using a direct transcription approach [13] with a fixed discretization grid, shown in Tab. I. We note that these problems have free final time so final time is also an decision variable. For a discretization grid of size N, the trajectory dimension is thus $Nn+(N-1)m+1$ where n and m are the state and control dimensions, respectively.

In this paper both MoE and SNN are trained to predict the optimal trajectories. As a result, trajectory tracking controller can be designed which compensates for uncertainties in system dynamics and prediction errors from learned models. For all the benchmark problems, we use LQR to design the tracking controllers. Since all problems terminate at a equilibrium state, we further use an LQR stabilizing controller to stabilize the system. It has benefits of compensating for trajectory tracking error. We define the process of tracking a trajectory and stabilizing at the equilibrium as trajectory rollout.

For a trajectory regression task, we minimize the loss function between model prediction and optimal trajectories. However, for a control task, we care more about if the predicted trajectory allows the system to achieve the goal. As a result, trajectory rollout success rate should be preferred as the metric for model evaluation. In fact, the averaging effect is beneficial to loss function but detrimental to trajectory rollout.

1) Pendulum Swing-Up : The system dynamic equations are

$$\dot{\theta} = \omega, \quad \dot{\omega} = u - \sin\theta \tag{8}$$

where θ is the pendulum angle, ω is the angular velocity, and $u \in [-1,1]$ is the control torque. 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 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 $w = 1, r = 1$.

The parameter space is a subset of $\mathbb{R}^2$ and we directly sample parameters on a uniform grid of 61×21. The validation set is sampled at random. Samples of optimal trajectories are shown in Fig. 1. The LQR stabilizing controller after trajectory tracking is simulated for 5 s and rollout success is determined by the norm of system state being within 0.1.

2) Vehicle: We model a ground vehicle with second-order dynamic equations

$$\dot{x} = v\sin\theta, \dot{y} = v\cos\theta, \dot{\theta} = u_\theta v, \dot{v} = u_v \tag{9}$$

where the state $\boldsymbol{x} = [x, y, \theta, v] \in \mathbb{R}^4$ 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 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 $w = 10, r_1 = r_2 = 1$. It is noted that stabilizer is not used since the linearized system is not controllable at the target state. As a result, rollout success is determined by the norm of system state being within 0.5.

3) Drone with One Spherical Obstacle: The system has state $x = (x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, p, q, r) \in \mathbb{R}^{12}$ and control $u \in \mathbb{R}^4$, and uses the dynamic equations specified in [24]. One spherical obstacle is considered which imposes path constraints on the state variables. The obstacle is characterized by its center (o_x, o_y, o_z) and radius o_r. The goal is to control the drone from any equilibrium state with position within $[-10\,m, 10\,m]^3$ and all other states zero to the goal state $\mathbf{0}$. The cost function is $J = w(t_f - t_0) + \int_{t_0}^{t_f} x^\mathrm{T} Q x + u^\mathrm{T} R u \,\mathrm{d}t$ which is a weighted sum of time, control energy, and penalty on states with $w = 10$, $Q = \mathrm{diag}(0,0,0,1,1,1,0.1,0.1,0.1,1,1,1)$, $R = \mathrm{diag}(1,1,1,1)$.

We uniformly sample initial positions of the drone. We randomly sample obstacles with radius within $[1\,m, 5\,m]$ around the straight line connecting initial and final positions with at least 0.5 m penetration. Additionally, the obstacle is required to be at least 0.5 units from initial and final positions. By trajectory rollout the drone can always reach the target position, but with different amount of constraint violation, i.e. the drone is too close to the obstacle. The rollout metric for each trajectory is chosen as the maximum constraint violation (in meter, lower value is desired.).

4) Drone with Two Spherical Obstacles: This problem is similar to the previous one and the only difference is two obstacles are considered. It increases the problem parameter dimensionality to 11.

B. Comparison of Training Approach

This section compares our proposed MoE approach with other training methods, using the pendulum swing-up benchmark task for better visualization. Results are comparable on other benchmarks. We compare 1) SNN; 2) MoE trained using (6) (MoE I); 3) MoE trained using (7) (MoE II); 4) our proposed decoupled MoE training with r clusters found by k-Means, denoted as "k-Means-r".

The SNN architecture is chosen to have size (2, 300, 75), which was tuned during testing to outperform architectures with more hidden layers or a larger hidden layer. The

TABLE I

SUMMARY OF BENCHMARK PROBLEMS

	Pendulum	Vehicle	Drone-One-Obstacle	Drone-Two-Obstacle
State dims	2	4	12	12
Control dims	1	2	4	4
Problem param.	$\mathbb{R}^2$	$\mathbb{R}^4$	$\mathbb{R}^7$	$\mathbb{R}^{11}$
Param range	$[-\pi,\pi]\times[-2,2]$	$[-10,10]^2\times[-\pi,\pi]\times[-3.1,3.1]$	$[-10,10]^6\times[1,5]$	$[-10,10]^9\times[1,5]^2$
Discretization	25	25	20	20
Trajectory dims	75	149	317	317
Dataset size	1,281	120,009	189,990	454,635
Mini-batch k-Means	No	Yes	Yes	Yes
SNN size	(2, 300, 75)	(4, 200, 200, 149)	(7, 500, 500, 317)	(11, 2000, 2000, 317)
Validation size	1,000	10,000	4,728	9,106

TABLE II

VALIDATION LOSS AND SUCCESS RATE ON PENDULUM PROBLEM

Model	SNN	MoE I	MoE II	k-3	k-5	k-10
Valid. loss	0.046	0.057	0.055	0.085	0.089	0.093
Succ. rate	86.3%	85.3%	83.9%	96.9%	99.9%	100.0%

MoE architectures are chosen such that the total number of parameters in all regressors equals SNN. For the classifier, we use a (2, 100, r) size where r is the number of regressors. For this comparison, we use $r = 5$ for MoE I and MoE II because results from k-Means-5 give high success rate and indicate 5 is a reasonable choice. We also test k-Means-r with 3, 5, and 10 clusters, adjusting regressor sizes to keep the total number of network weights the same.

Tab. II lists the validation loss and rollout success for each model, Fig. 3 plots the prediction error in θ_f, i.e. the final angle, and Fig. 4 plots the regions that each regressor is assigned to. Tab. II shows that with r at least 5, our method achieves extremely high rollout success rates. Moreover, loss is not directly related to success rate. In fact, the k-Means models have higher success rate despite higher loss.

Fig. 3 shows SNN, MoE I and MoE II have problems in trajectory prediction for initial states close to the discontinuities shown in Fig. 1.c. Due to instability of backpropagation, neither MoE I or MoE II are able to train the classifier well, as can be seen in Fig. 4. MoE I actually is dominated by 2 regressors and has only one input space partition boundary. MoE II learns more boundaries, but they are in fact quite different from the true discontinuities.

With an insufficient number of clusters, k-Means-r makes large prediction errors at the smoothed-out discontinuity. $r \geq 5$, our method learns discontinuities quite accurately. With larger numbers of clusters, the input space may be oversegmented, but with a sufficient amount of data the success rate is not affected much.

C. Trajectory Rollout Results

These experiments now compare SNN and our MoE method using clustered training, but with varying numbers of clusters. The SNN sizes shown in Tab. I are fine-tuned for each problem. MoE network architectures are set to match the total number of parameters in SNN.

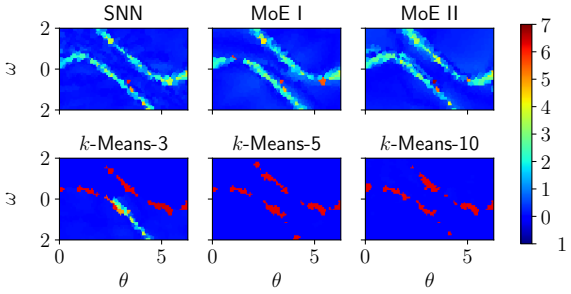


Fig. 3. Prediction error in θ_f for several models. Blue denotes small prediction error. Red means an error of 2π, which indicates a misclassification, but it does not affect trajectory tracking because an upright state is reached. Other colors indicate problems with averaging. (Best viewed in color)

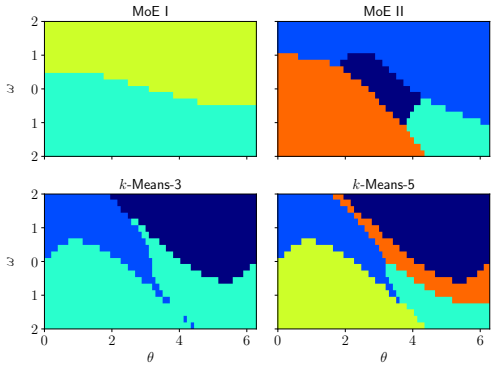


Fig. 4. Input space partition by the classifier. In MoE I 2 regressors dominate the other 3 and the boundary is quite far from discontinuity boundary. MoE II is unable to find boundaries either. k-Means with 3 clusters partially finds the boundary. k-Means with 5 clusters identifies discontinuity boundary with over segmentation.

First, let us examine a few examples for the vehicle and drone-one-obstacle problems. Fig. 5 shows the predictions from SNN and MoE on a selected initial state on Vehicle, as well as the optimal trajectories of its neighbors. SNN fails to predict the final orientation correctly, as shown by the orange arrow. The reason is that near this parameter, its neighbors belong in different clusters, leading to an averaging effect. MoE with 10 clusters does not have this effect, and its result is nearly equal to ground truth.

Fig. 6 shows examples of optimal trajectories and predictions on the Drone-One-Obs problem. As the initial state

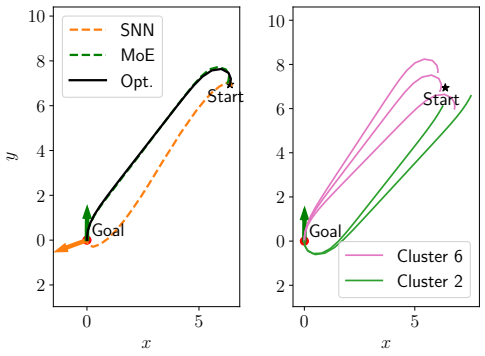


Fig. 5. Left: The optimal trajectory and predictions (no tracking) from SNN and MoE for a chosen start state for the Vehicle problem. MoE makes accurate prediction that SNN which predicts an average trajectory of two clusters. The arrow denotes the final angle (straight up is desired) and SNN makes large prediction on it. Right: the neighbors of the chosen state. The pink and green lines are the neighbors of the start so they start from different locations. They demonstrate the existing discontinuity within trajectories.

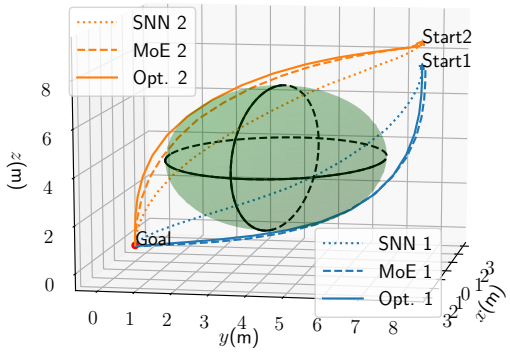


Fig. 6. Optimal trajectories and prediction (no tracking) from SNN and MoE for two selected close states for the Drone-One-Obstacle problem. SNN predicts a trajectory that violates obstacles avoidance constraints. Green sphere: obstacle with center (0, 4, 4) and radius 3. Solid, dashed and dotted lines: optimal trajectories, prediction of MoE, and prediction of SNN, respectively.

moves along z direction, the optimal trajectories turns from going above to going below the obstacle. SNN is unable to handle the discontinuity and predicts a trajectory that severely violates the constraints. However, MoE with 20 clusters is able to detect the discontinuity, and each regressor near this boundary predicts either trajectories above or below the obstacle, but never through it.

Fig. 7 shows the average rollout error over the validation set, with varying numbers of clusters. For the pendulum and vehicle problem, MoE reaches 99.5+% success rate in reaching the target with at least 10 clusters. For the two drone problems, MoE improves the constraint violation error by over 50% improvement, thus has higher reliability than SNN. The number of clusters r is an important parameter for performance. If it is too low, some cluster may learn from a proportion of parameter space with function discontinuity and has the same averaging issue with SNN. As it increases, the performance tends to improve significantly until a certain threshold is reached. If it further increases, the performance tends to degrade slightly, which is likely due to over-fitting since each cluster has less data. Another explanation may

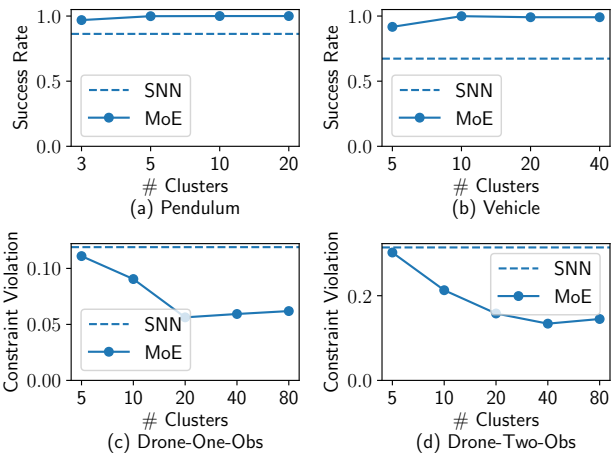


Fig. 7. Comparison between MoE and SNN on benchmark problems. MoE outperforms SNN in all problems. For (a) and (b), MoE achieves higher than 99% success rate. For (c) and (d) MoE has 50% improvement in terms of constraint violation. It also shows as cluster number increases, MoE performance increases and eventually decreases.

be with more clusters, the number of artificial boundaries increases and the neural network has to perform more extrapolation instead of interpolation.

V. Conclusion

This paper demonstrates that optimal trajectories can be learned with extremely high accuracy if the special structure of optimal control problems is taken into account. Without taking the structure into account, learning performs poorly even with huge datasets. The MoE model is designed such that each expert approximates a smooth region in the problem-optimum map, and the classifier handles discontinuities without averaging. Experiments demonstrate that randomly initialized MoE training with cost functions found in existing literature tend to perform worse than our proposed technique, which pre-trains the classifier with clusters of states that produce similar trajectories. As the number of clusters increases, the MoE performance tends to increase first due to better conformance to discontinuity boundaries, and eventually decrease slightly due to overfitting.

Limitations of this approach include the need for a parameterization of the problem domain, which can be challenging for some tasks. For example, to capture different numbers of spherical obstacles, the data collection and training procedure have to be repeated for each obstacle count. Moreover, for obstacles with irregular shape, it is not easy to find a low-dimensional representation.

Future work includes developing more sophisticated clustering algorithms that automatically find the best partitioning strategy. Also, for certain OCPs, differential flatness can be used such that the predicted trajectory satisfies dynamical constraints. We are also interested in proving the stability of the predicted trajectories under perturbations, 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 E. Pistikopoulos, "The explicit solution of model predictive control via multiparametric quadratic programming," in *Proc. American Control Conf.*, vol. 1–6, 2000, pp. 872 – 876.

[2] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[3] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," *Autonomous Robots*, vol. 34, no. 1-2, pp. 111–127, Jan. 2013.

[4] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, "Trajectory planning for optimal robot catching in real-time," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3719–3726.

[5] G. Tang, W. Sun, and K. Hauser, "Learning trajectories for real-time optimal control of quadrotors," in *Intelligent Robots and Systems (IROS), 2018 IEEE/RSJ International Conference on*. IEEE, 2018, pp. –.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[9] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[10] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

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

[12] F. Jiang, H. Baoyin, and J. Li, "Practical techniques for low-thrust trajectory optimization with homotopic approach," *J. Guid. Control Dynam.*, vol. 35, no. 1, pp. 245–258, 2012.

[13] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[14] G. Tang and K. Hauser, "A data-driven indirect method for nonlinear optimal control," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. –.

[15] T. Tomić, M. Maier, and S. Haddadin, "Learning quadrotor maneuvers from optimal control and generalizing in real-time," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1747–1754.

[16] K. Hauser, "Learning the problem-optimum map: Analysis and application to global optimization in robotics," *IEEE Trans. Robotics*, vol. 33, no. 1, pp. 141–152, Feb. 2017.

[17] A. V. Fiacco, "Introduction to sensitivity and stability analysis in nonlinear programming." 1983.

[18] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 593–604.

[19] B. Tang, M. I. Heywood, and M. Shepherd, "Input partitioning to mixture of experts," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 1. IEEE, 2002, pp. 227–232.

[20] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," *arXiv preprint arXiv:1709.06560*, 2017.

[21] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[22] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," *Artificial Intelligence Review*, vol. 42, no. 2, pp. 275–293, 2014.

[23] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 1177–1178.

[24] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, Apr. 2012.