Discontinuity-Sensitive Optimal Trajectory Learning by Mixture of Experts

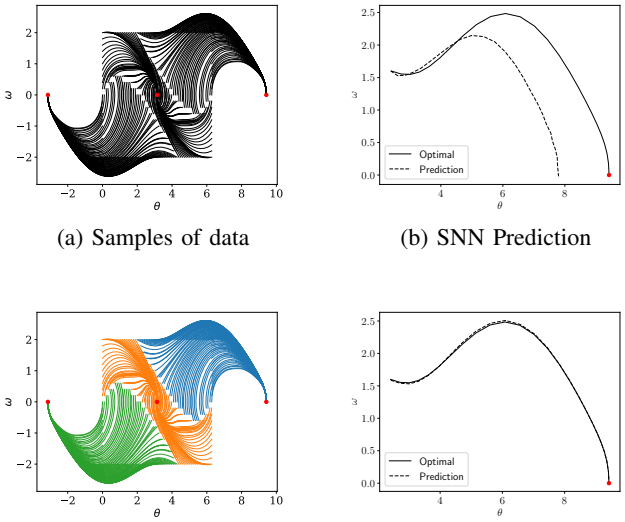
Gao Tang[1] and Kris Hauser[2]

***Abstract*— Many robotic tasks, ranging from model predictive control to reinforcement learning, may be characterized as solving a parametric optimization problem where the optimal solutions are 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 such as neural networks. This paper proposes a discontinuity-sensitive approach to learn optimal trajectories with high accuracy. A Mixture of Experts (MoE) model composed of a classifier and several regressors is proposed. The optimal trajectories to sampled problem 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 coupled training of MoE. Compared with either supervised learning with neural network or deep reinforcement learning, this approach achieves dramatic improvements in the reliability of trajectory tracking.**

I. Introduction

Nonlinear Optimal Control Problems (OCPs) are critical to achieve high performance in robotics applications such as model predictive control [1] and kinodynamic motion planning [2]. There is intense interest in using learning to obtain approximations of optimal policies, either using supervised learning [4], [7] or deep reinforcement learning (DRL) [8]. 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 SNN to learn a pendulum swing up task from precomputed optimal trajectories. The optimal trajectories are calculated by solving optimal control problems from different initial states (angle and angular velocity) to swing up state. The optimal trajectories have three possible goal states due to periodicity of angles so the parameter-solution mapping is discontinuous and has three homotopy classes. Near the region of homotopy class switching, SNN predicts a final state that interpolates between two goal states, as shown in Fig. 1b. As a result, such a prediction cannot be used for designing trajectory tracking controller.

This paper addresses the discontinuity of parameter-solution mapping by modifying the Mixture of Experts (MoE) model [3], [5], [12]. An MoE model has a classifier (gating network) to select one out of many regressors (experts) to make the prediction. We differentiate from [3], [5], [12] by using the prediction from *one* regressor instead of weighted average of all regressors. The training data is clustered into continuous regions such that each regressor learns a continuous function, thus avoiding the problem faced by SNN. As shown in Fig. 1d, the prediction does not average two nearby homotopy classes.



(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 swing up task. (a) samples of optimal swing up trajectories from different initial states. The red circles are swing up states. (b) prediction of a selected state by SNN with large error. (c) samples of clustered optimal trajectories where each color denotes one cluster. Trajectories are clustered according to final angle. (d) prediction by MoE to the same state as (b).

Training of MoE has been done by backpropagation [12] and expectation maximization [5]. In this paper, backpropagation is used to train the classifier and regressors *individually* instead of coupledly. In fact, coupled training does decrease regression error, but it leads to lower task success rate. Experiments on benchmark underactuated control problems demonstrate that suitably trained MoE models can learn near-optimal trajectories suitable for trajectory tracking with remarkably high success rates (99.5+%), while SNN and DRL suffer from the discontinuity of the parameter-solution mapping.

Our approach is fundamentally different from imitation

*This work was not supported by any organization

[1]Gao Tang is with Department of Mechanical Engineering and Material Science, Duke University, Durham, NC 27708, USA gao.tang@duke.edu

[2]Kris Hauser with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA kris.hauser@duke.edu

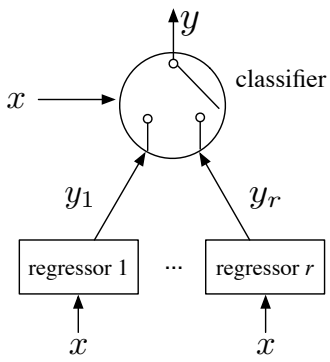


Fig. 2: Illustration of MoE. The prediction is made by *one* out of r regressors selected by the classifier.

learning [10] which learns the policy of the expert. Our approach directly learns the optimal trajectory and relies on tracking algorithm for stability guarantee. Besides, if the test set has the same distribution with training set, there is no need to recollect expert's trajectory. Admittedly, our approach requires we know precisely the system dynamics. Our approach is also different from reinforcement learning [8] since it is supervised learning and has better sample efficiency. Besides, it directly learns the trajectory other than policy.

II. Mixture of Experts

The MoE model is composed of a classifier and r regressors, as shown in Fig. 2. We denote $P(\boldsymbol{p}, w_c) \in \mathbb{R}^r$ as the output from the classifier and $\{y_i(\boldsymbol{p}, w_i)\}_{i=1}^r$ as the outputs from each regressor where $\boldsymbol{p}$ is problem parameter and w_c, w_i are neural network weights. The prediction of MoE is thus

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

The output P from the classifier can be the output from a softmax layer [3] which results in a weighted sum of the prediction from all regressors. Alternatively, we calculate P by an argmax layer which results in vector of zeros except for one entry, essentially selecting one among all regressors. The target is to find w_c and $\{w_i\}_{i=1}^r$ in order to miminize

$$L = \mathbb{E}_{\boldsymbol{p} \sim P_{\text{data}}} \text{loss}(z(\boldsymbol{p}), z^\star(\boldsymbol{p})) \tag{2}$$

where P_{data} is a distribution over problem parameters.

III. How to Train MOE–Study on Simple Task

We study a simple pendulum swing up task from different initial states. The pendulum has two states–angle and angular velocity; and one control–the moment. It has to reach a straight up state where the angle is $(2k+1)\pi, k \in \mathbb{Z}$. For this problem, the goal is to control the pendulum to straight up state from arbitrary state with certain cost function. The function to be learned maps from system state (angle and angular velocity, in $\mathbb{R}^2$) to an optimal trajectory which is a collection of time stamped state and control variables, in $\mathbb{R}^{75}$. The optimal trajectory contains information of optimal time and control to reach the goal, and the system states under this control. The initial angle and angular velocity are sampled on a uniform grid of 1281 points. We compare on two metrics: 1) trajectory regression error and 2) task success rate by tracking the predicted trajectory, denoted as rollout success rate. The following variations are considered:

1) SNN vs MoE,
2) Coupled training (random initialization) against individual training with k-Means clustering, and against clustering by expert knowledge, denoted as custom clustering, and
3) Retraining after training individually vs no retraining.

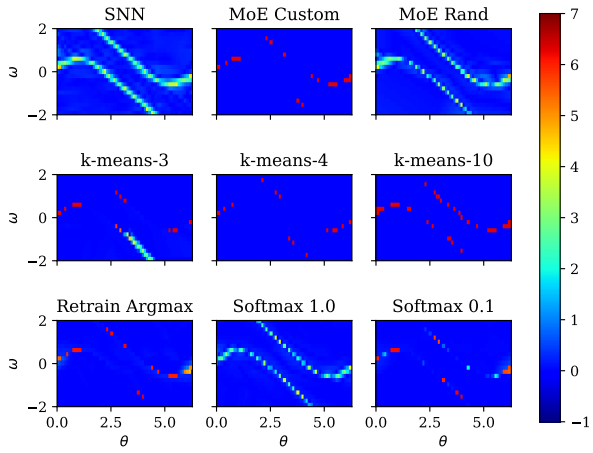
The SNN is chosen as fully-connected multi-layer perceptron (MLP) of size (2, 300, 75) where each number denotes size of each layer (from input layer to output layer). For MoE, the classifier is of size (2, 50, r) and the r regressors are all of size (2, 20, 75). Custom MoE and random-weight MoE use 3 experts. The custom clustering divides the data into 3 clusters based on the three possible final angles. We also use k-Means with 3, 4, and 10 clusters solely on trajectories with the same design of network size to study the effect of cluster numbers. We note that custom clustering differs from k-Means with 3 clusters.

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

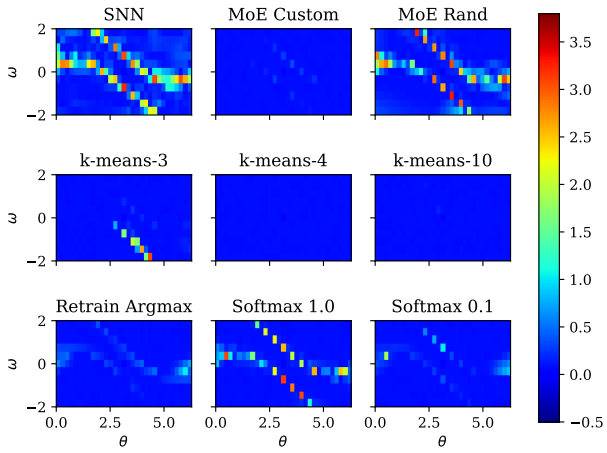
Row 1 shows that SNN has difficulty in making predictions in regions near the discontinuity, averaging between both sides. MoE does also make inaccurate prediction, but these are caused by misclassification and the prediction is a local optimal trajectory belonging to another homotopy class. Hence, prediction from misclassification still reaches the vertical position as desired, since the difference in θ_f is 2π. MoE trained from random initialization makes better prediction than SNN, but is not very successful at trajectory tracking. This indicates that training MoE by simply minimizing regression error is unable to guide the classifier to the appropriate clusters.

Row 2 tests MoE clustered by k-Means with various cluster sizes. $k = 3$ shows the results when appropriate clusters are not found by k-Means. $k = 4$ and $k = 10$ clusters finds the discontinuity successfully, and the resulting MoE achieves high success rate. Although experts knowledge indicates existence of at least three clusters, the results with higher number of clusters shows using more than sufficient clusters *does not* degrade the performance.

Row 3 shows various choices 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 to give binary outputs. The classifier has no gradient to update itself so only the regressors are updated. Due to classification error, the regressors will be trained with trajectories from other clusters so its prediction tends toward the average. In (vii) and (ix) we use softmax with different ε. (softmax function is performed to the output layer after dividing ε.) In these cases, the classifier is updated but the regressors



(a) Prediction error of θ_f



(b) State error after trajectory tracking

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

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 regression error can be misleading. Due to misclassifications, lower regression 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. In order to achieve high rollout success, the MoE prediction cannot be some combination of predictions from several regressors and each regressor cannot be trained using data from different clusters.

IV. Comparison with DRL–Study on Vehicle Task

We apply MoE on a ground vehicle task where a vehicle has 4 states–$[x, y, \theta, v]$ of planar coordinates, orientation and velocity and 2 controls–$[u_\theta, u_v]$ of steering and accelerating.

$$\dot{x} = v\sin\theta, \dot{y} = v\cos\theta, \dot{\theta} = u_\theta v, \dot{v} = u_v \tag{3}$$

The task is to drive the vehicle from any initial states within some range to the origin with zero velocity and orientation. The cost function is a weighted sum of time and control energy.

For MoE, we collect 120,009 optimal trajectories by randomly sampling initial states and solving the corresponding trajectory optimization problem. In order to apply MoE, 6 clusters of trajectories are used corresponding to 3 possible final angles and two velocity profile (going forward or backward). Existence of 6 clusters can be found by either manifold embedding like UMAP [9] or principle component analysis. Defining success as controlling the vehicle with final state error within 0.5, the success rate is 99.8% (9,975/10,000) on a validation set, as shown in Fig. 5.

We apply open-source PyTorch implementation [6] of Proximal Policy Approximation (PPO) [11], a state-of-the-art DRL method on this task. In order to have fair comparison, we let PPO be trained with 3,600,000 steps which surpass the total states seen by MoE (each trajectory has 20 states) by 50%. We use a reward function that penalizes state, control, and time with additional rewards encouraging the state close to the goal. The learning curve is shown in Fig. 4 and we note that successful policies should have positive rewards. It shows PPO is unable to find a universal policy that applies to all the states, at least within the same number of states used in training. Policy rollout on the 10000 initial states from validation set gets *four* success out of 10000 random initial states.

We argue the failure of PPO is not caused by the choice of reward function but the inability of SNN to learn the discontinuous policy as MoE does. To demonstrate so, we modify the sampling region of initial states and validate the policy on samples of 200 initial states. The region of initial states that PPO fails to solve while MoE achieves high success rate is $[-10,-10,-\pi,-3.1]$ to $[10,10,\pi,3.1]$. In the first experiment, we sample initial states within $[5,5,0,0]$ and $[10,10,0,0]$. The policy converges in about 200,000 steps and achieves 200/200 success. Next we sample initial states within $[0,5,0,0]$ and $[10,10,0,0]$ which doubles the sampling region of the first experiment. PPO learns a policy that achieves 161/200 success, which has lower success rate. Finally we sample initial states within $[-10,5,0,0]$ to $[10,10,0,0]$ which further doubles the sampling region. The policy learned by PPO only achieves 19/200 success. The success of experiment one shows the reward function is appropriate for PPO. However, next two experiments shows PPO is unable to learn a policy expressed by SNN as problem complexity increases.

A summary of rollout results are shown in Fig. 5. This task shows deep RL has difficulty in problems where optimal policy is indeed discontinuous. However, MoE is capable of

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 1,000)	717	998	829	970	1,000	1,000	941	896	969

handling such discontinuity. Besides, we also test deep RL on a simplified obstacle avoidance problem, a feasible policy is also difficult to be obtained even with extensive trial of reward shaping.

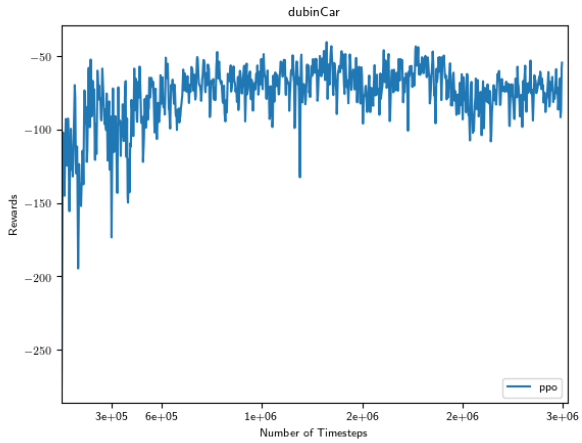


Fig. 4: Learning curve from PPO on the vehicle problem

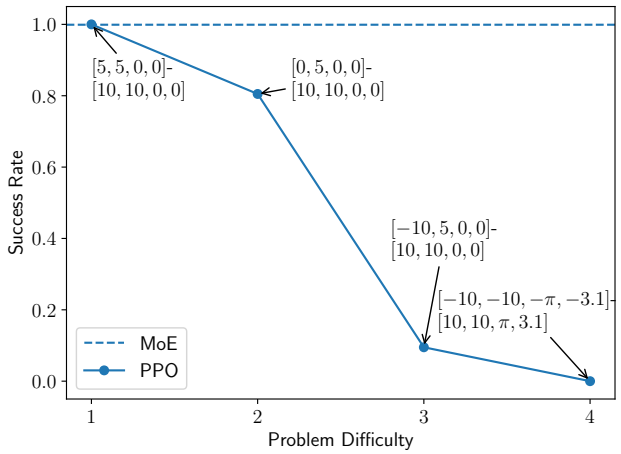


Fig. 5: Rollout success rate on vehicle problems with different difficulty. As initial states are sampled in increasingly larger region, PPO achieves lower success rate to complete failure. MoE manages to achieve high success rate even for the largest sampling region.

V. Conclusion

In this paper we demonstrate that MoE is capable of handling discontinuity in parameter-solution mapping while SNN and DRL both have difficulty in problems with discontinuity. It is important to train MoE individually 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. Applying MoE to problems in higher dimensionality is another direction to explore.

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] 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.

[4] 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.

[5] 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.

[6] I. Kostrikov, "Pytorch implementations of reinforcement learning algorithms," https://github.com/ikostrikov/pytorch-a2c-ppo-acktr, 2018.

[7] 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.

[8] 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.

[9] L. McInnes and J. Healy, "UMAP: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[10] 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*, 2011, pp. 627–635.

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

[12] 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.