Combining deep gaussian process and rule-based method for decision-making in self-driving simulation with small data

Wenqi Fang[1,*], Huiyun Li[1], Shaobo Dang[1]
[1]*Center for Automotive Electronics*
Shenzhen Institutes of Advanced Technology, CAS
Shenzhen, P.R.China
Email: wq.fang@siat.ac.cn

Hui Huang[1], Lei Peng[1], Li-Ta Hsu[2], Weisong Wen[3]
[2]*Interdisciplinary Division of Aeronautical and Aviation Engineering*
[3]*Department of Mechanical Engineering*
Hong Kong Polytechnic University
Hong Kong, P.R.China

***Abstract*—Self-driving vehicle is a popular and promising field in artificial intelligence. Conventional architecture consists of multiple sensors, which work collaboratively to sense the units on road to yield a precise and safe driving strategy. Besides the precision and safety, the quickness of decision is also a major concern. In order to react quickly, the vehicle need to predict its next possible action, such as acceleration, brake and steering angle, according to its latest few actions and status. In this paper, we treat this decision-making problem as a regression problem and use deep gaussian process to predict its next action. The regression model is trained using simulation data sets and accurately captures the most significant features. Combined with rule-based method, it can be used in Torcs simulation engine to realize successful loop trip on virtual roads. The proposed method outperforms the existing reinforcement learning methods on the performance indicators of time consumption and the size of data volume. It may be useful for real road tests in the future.**

***Keywords*-gaussian process; kernel function; rule-based; decision-making**

I. Introduction

Self-driving vehicle is one of the most promising fields of artificial intelligence. To realize safety driving on the real road, ego-vehicles need to recognize and track the objects with its perceptual equipment, as well as act properly according to the current road conditions with decision-making modules. The decision-making module is not only the most important but also the challenging part to achieve. The core tasks include action prediction, obstacle avoidance, trajectory planning and so on [1]. As a classical and robust tool for dynamic modeling, finite state model is wildly used in the conventional decision-making modeling of self-driving. It summarizes the events, which decide the action of unmanned cars, and divides them into finite states. Each action represents a specific control over the car. Building the decision-making model with rule-based [2] or statistical method [3] are two popular schemes. Rule-based methods can achieve functionality quickly, but they are confined by the incomplete sets of state and the inability of capturing uncertainty. These shortcomings are overcome by combining with statistical methods. In addition, with the advent of simulation engines like Carla [4] and Torcs [5], various methods based on reinforcement learning [6] are proposed in the decision-making research and achieve satisfactory performances. Although reinforcement learning is wildly used in the visual game engine, it is far from real road test and the stability also need to be further strengthened.

In recent years, Gaussian process (GP) and deep gaussian process (deep GP) regression have become prevailing regression techniques [7, 8]. To be precise, a GP is a distribution over functions such that any finite set of function values have a joint Gaussian distribution. The obtained mean function and covariance function are used for regression and uncertainty estimation, respectively. The strength of GP regression lies in avoiding overfitting while still finding functions complex enough to describe any observed behavior, even in noisy or unstructured data. GP is usually applied to the situation when observations are rare or expensive to produce and methods like deep learning perform poorly. And it has been applied among a wide range from robotics [9], biology [10], global optimization [11], astrophysics [12] to engineering [13]. A deep GP consists of a cascade of hidden layers of latent variables where each node acts as output for the layer above and as input for the layer below. GPs govern the mappings between the layers with their own kernels. Damianou and Lawrence [14] showed that the probabilistic nature of deep GP guards against overfitting and it is flexible and robust enough to model complex data.

Based on the analysis above, in this paper, we propose a decision-making framework combining deep GP and rule-based methods. The model is trained with small data in Torcs and tested in Torcs. The rest of the paper is organized as follows. In Sec. II, we give a introduction to the GP method. In Sec. III, we explain our method for decision-making problem thoroughly. Finally, we make some necessary discussions and give our conclusions in Sec. IV and V, respectively.

II. Gaussian Process

GP prior over function $\mathbf{f}(\mathbf{x})$ implies that any set of function values $\mathbf{f}$, indexed by the input $\mathbf{x}$, have a multivariate

Gaussian distribution

$$p(\mathbf{f}\,|\mathbf{x},\theta) = \mathrm{N}(\mathbf{f}\,|\mathbf{0},\mathbf{K}_{\mathrm{f,f}}), \tag{1}$$

where $\mathbf{K}_{\mathrm{f,f}}$ is the covariance matrix. And it is always denoted as $\mathbf{f}(\mathbf{x}) \sim \mathcal{GP}(\mathbf{f}\,|\mathbf{0},\mathbf{K}_{\mathrm{f,f}})$. The knowledge about mean value can be easily incorporated into the covariance function, so the mean value for the GP is always set to be zero. The covariance matrix is constructed by a covariance function, $[\mathbf{K}_{\mathrm{f,f}}]_{i,j} = k(\mathbf{x}_i,\mathbf{x}_j|\theta)$, which characterizes the correlation between different points in the process. Covariance function can be chosen freely as long as the covariance matrices produced are symmetric and positive semi-definite. A common stationary covariance function is the squared exponential

$$k(\mathbf{x}_i,\mathbf{x}_j\,|\theta) = \sigma_{se}^2 \exp(-\frac{(\mathbf{x}_i-\mathbf{x}_j)^2}{2l^2}), \tag{2}$$

where $\theta = [\sigma_{se}, l]$ are the hyper-parameters. σ_{se} is the scaling parameter, and l is the length-scale parameter, governing how fast the correlation decreases as the distance increases. If the input is in multiple dimensions, a flexible way to model functions with multidimensional input is to multiply together kernels defined on each individual dimension. For example, a product of squared exponential kernels over d-dimension inputs is, each having a different length-scale parameter,

$$k(\mathbf{x}_i,\mathbf{x}_j\,|\theta) = \sigma_{se}^2 \exp(-\sum_{k=1}^{d} \frac{(x_{i,k}-x_{j,k})^2}{2l_k^2}). \tag{3}$$

This is often called automatic relevance determination squared exponential kernel, so named because the length-scale parameters l_1, l_2, . . ., l_d, implicitly determines the relevance of each dimension. The input dimensions with relatively large length-scales imply relatively little variation along those dimensions in the function being modeled. Other common used kernel functions can be found and discussed in [15].

Assuming that we want to predict the values $\tilde{\mathbf{f}}$ at new input locations $\tilde{\mathbf{x}}$. The joint prior for latent variables $\mathbf{f}$ and $\tilde{\mathbf{f}}$ is

$$p(\begin{bmatrix}\mathbf{f}\\ \tilde{\mathbf{f}}\end{bmatrix}|\mathbf{x},\tilde{\mathbf{x}},\theta) \sim \mathrm{N}\left(\mathbf{0},\begin{bmatrix}\mathbf{K}_{\mathrm{f,f}} & \mathbf{K}_{\mathrm{f},\tilde{\mathrm{f}}}\\ \mathbf{K}_{\tilde{\mathrm{f}},\mathrm{f}} & \mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}}\end{bmatrix}\right), \tag{4}$$

where $\mathbf{K}_{\mathrm{f,f}} = k(\mathbf{x},\mathbf{x}|\theta)$, $\mathbf{K}_{\mathrm{f},\tilde{\mathrm{f}}} = k(\mathbf{x},\tilde{\mathbf{x}}|\theta)$ and $\mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}} = k(\tilde{\mathbf{x}},\tilde{\mathbf{x}}|\theta)$. By definition of GP, the marginal distribution of $\tilde{\mathbf{f}}$ is $p(\tilde{\mathbf{f}}|\tilde{\mathbf{x}},\theta) = \mathrm{N}(\tilde{\mathbf{f}}|\mathbf{0},\mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}})$ similar to (6). The conditional distribution of $\tilde{\mathbf{f}}$ given $\mathbf{f}$ is

$$p(\tilde{\mathbf{f}}|\,\mathbf{f},\mathbf{x},\tilde{\mathbf{x}},\theta) \sim \mathrm{N}(\mathbf{K}_{\tilde{\mathrm{f}},\mathrm{f}}\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathbf{f}, \mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}} - \mathbf{K}_{\tilde{\mathrm{f}},\mathrm{f}}\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathbf{K}_{\mathrm{f},\tilde{\mathrm{f}}}), \tag{5}$$

where the mean and covariance of the conditional distribution are functions of input vector $\tilde{\mathbf{x}}$ and $\mathbf{x}$ serves as a fixed parameter. Typically, it is more realistic that in our training data, only the noisy observations are available rather than the precise function value of $\mathbf{f}(\mathbf{x})$. Further, we assume that each data point $\mathbf{y}$ is generated from the corresponding $\mathbf{f}(\mathbf{x})$ by adding independent Gaussian noise:

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) + \varepsilon, \tag{6}$$

where $\varepsilon \sim \mathrm{N}(\mathbf{0},\sigma\mathbf{I})$. In this case, we should incorporate this noise into our model by adding noise σ to every diagonal term in each covariance matrix which corresponds to a kernel $\mathbf{K}_{\mathrm{f,f}} = k(\mathbf{x},\mathbf{x}|\theta) + \sigma\mathbf{I}$. Once the kernel is updated, the analysis is identical as previously discussed. Now the observation becomes $\mathbf{y}$, and we are trying to predict $\tilde{\mathbf{y}}$ with new kernel $\mathbf{K}_{\mathrm{f,f}}$:

$$p(\tilde{\mathbf{y}}|\,\mathbf{y},\mathbf{x},\tilde{\mathbf{x}},\theta) \sim \mathrm{N}(\mathbf{K}_{\tilde{\mathrm{f}},\mathrm{f}}\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathbf{y}, \mathbf{K}_{\tilde{\mathrm{f}},\tilde{\mathrm{f}}} - \mathbf{K}_{\tilde{\mathrm{f}},\mathrm{f}}\,\mathbf{K}_{\mathrm{f,f}}^{-1}\,\mathbf{K}_{\mathrm{f},\tilde{\mathrm{f}}}). \tag{7}$$

To use GP to solve the problem, we need to infer the parameters in the model during training procedure. A crucial character for GP is that we can calculate its marginal likelihood, and it can help significantly for model selection. To maximize the log marginal likelihood is a popular way to tune kernel parameters. The marginal likelihood is, given the parameters, $p(\mathcal{D}|\theta,\phi) = \int p(\mathbf{y}|\,\mathbf{f},\phi)p(\mathbf{f}\,|\mathbf{x},\theta)d\,\mathbf{f}$, where θ and ϕ are for kernel and noise parameters, respectively. With a Gaussian likelihood, it has an analytic closed-form which gives the log marginal likelihood:

$$\begin{aligned}\log p(\mathcal{D}|\theta,\sigma) = &-\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\,\mathbf{K}_{\mathrm{f,f}} + \sigma^2\mathbf{I}|\\ &-\frac{1}{2}\,\mathbf{y}^{\mathrm{T}}(\mathbf{K}_{\mathrm{f,f}} + \sigma^2\mathbf{I})^{-1}\,\mathbf{y},\end{aligned} \tag{8}$$

where n is the number of the input data. If the likelihood is not Gaussian, the marginal likelihood needs to be approximated. Many approximate methods can be used, like Laplace approximation [16], variational method [17] and sampling method [18].

III. Method for decision-making problem

In this section, the detailed description of applying GP in decision-making process of self-driving is given. The whole framework is presented in Fig. 1. First, the training data is generated by interactions between network trained by deep deterministic policy gradient (DDPG) algorithm [19] and the simulated driving on CG road (the overview map in upright corner in Fig. 1) in TORCS. Then, the data set is fed into a multiple-layer GP, which is often called deep GP. The trained deep GP model is used to predict the actions according to the state feedbacks from TORCS. For validation, all the generated actions are further filtered by a set of man-made rules for feasibility and safety concerns. The refined actions are then sent to TORCS to demonstrate visually the performance on a loop trip. All the details of each procedure are further explained in the following section.

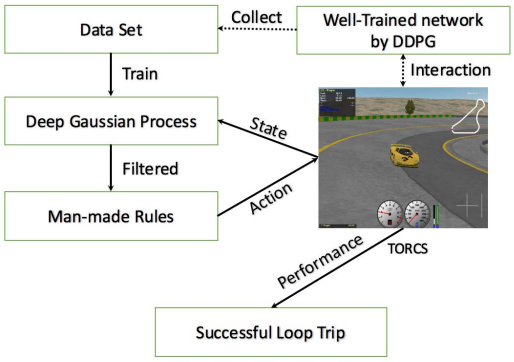

Figure 1: The framework of our method

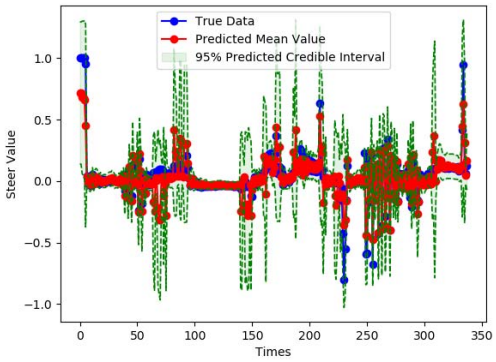

Figure 2: Steer Value vs Times

A. *data structure*

We collect the training data by DDPG well-trained network on the CG road in software TORCS. About 340 records are collected during the loop trip simulation. The training data consists of state set regarding sensor's states and action set from the controller in TORCS. The state set and action set are represented by several variables presented in tables I and table II, respectively. For deep GP network training, the state set is the input and action set is the output. And the raw data are fed into deep GP network with no need for data processing before training. The data and demo will be public online in the near future.

Table I: The information of state set

Name	Range	Detail description
ψ	[-1,1]	Angle between the car direction and track axis direction
d_1	[0, 1]	Distance between the car and the track edge in front of the car
d_2	$[-\infty, \infty]$	Shift from the center line
v	$[-\infty, \infty]$	Car speed in x, y, z direction
ω	$[0, \infty]$	The speed of four wheels
r	$[0, \infty]$	Gear speed

Table II: The information of action set

Commad	Range	Detail description
steering	[-1,1]	-1 and +1 mean full right and full left, respectively
acceleration	[0, 1]	Virtual gas pedal (0 means no gas, 1 means full gas)
brake	[0, 1]	Virtual brake pedal (0 means no brake, 1 means full brake)

B. *deep gaussian process*

As for this multidimensional input and output problem, we use deep GP method to fit the training data in consideration of its advantage over a standard GP [14]. An example of a deep GP is a composition of vector-valued functions, with each function drawn independently from GP priors [15]:

$$\mathbf{f}^{(1:L)}(\mathbf{x}) = \mathbf{f}^{(L)}(\mathbf{f}^{(L-1)}(\dots \mathbf{f}^{(2)}(\mathbf{f}^{(1)}(\mathbf{x}))\dots)) \quad (9)$$
$$\text{with each} \quad f_d^l \sim \mathcal{GP}\ \ 0, k_d^l(\mathbf{x}, \mathbf{x}') \ \ \text{for} \quad f_d^l \in \mathbf{f}^l\,.$$

In addition to the input $\mathbf{x}$ and output $\mathbf{y}$, there are $L-1$ hidden layers. The input dimension, also known as the number of units, of layer l is the output dimension of layer $l-1$, while the output dimension of layer l is the number of functions in $\mathbf{f}^l$. Thus, each function $f_d^l \in \mathbf{f}^l$ takes as input a vector whose dimension is the output dimension of layer $l-1$, and returns a scalar. In this case, we use famous package GPy [20], which is developed by Sheffield machine learning group, to do the simulation. We use two layers deep GP to fit the training data. The kernels we used per layer are as follows:

$$[StdPeriodic * RatQuad + RBF + White, \\ MLP * Matern52 + RBF + White], \quad (10)$$

all the function expressions corresponding to these function names can be found in the GPy document webpage. The number of inducing points we used is 200. After optimization, the output action values are shown in Fig. 2, Fig. 3, and Fig. 4. The true data and predicted mean value legend in the figures mean the true training data action values and the predicted values after finish training deep GP, respectively. The green zone, with its margin depicted by the green dashed line, in the figures represents the 95% credible interval of the predicted value. The x-axis variable $Times$ represents the steps of self-driving simulation. We can see that the deep GP can capture the main features of the training data except for some strong vibration zones. The further experiments suggest that a more dedicated parameter tuning process will improve the performance.

C. *man-made rules*

After training the deep GP model, all the parameters are determined immediately. We try this model in TORCS, and

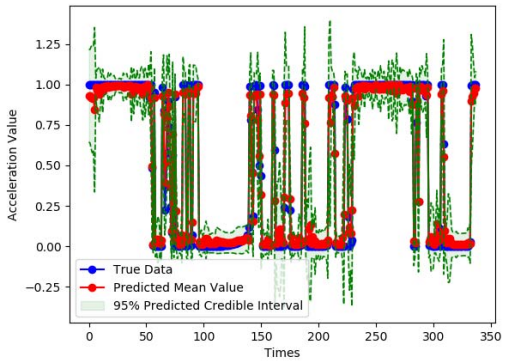

Figure 3: Acceleration Value vs Times

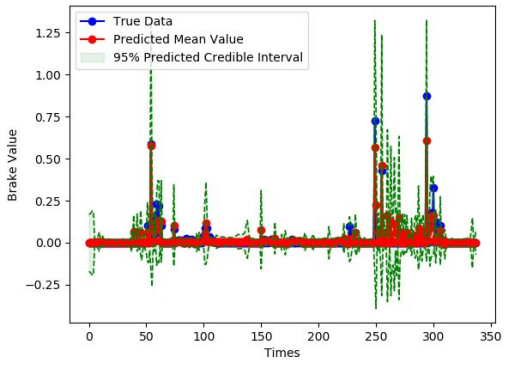

Figure 4: Brake Value vs Times

we find that it can only finish about half loop trip on the CG road. After analyzing the failed predictions and the input data, we find the main cause is that the data from DDPG well-trained network only contain the cases with small ψ, and d_2 which are close to the center. With the highly deviated input, the trained deep GP generate unreasonable action value. We assume that if the values of ψ and d_2 stay in a reasonable range, the successful loop trip can be achieved regardless of the values of other state variables. With that being thought, we design extra logic rules for these two variables. Specifically, a few critical values are set for these two changing variable in the training data and the steering values are also slightly tweaked by adding a specific offset on the original values. And the effectiveness of the adjustment has been approved by the simulation results of the successful completion of loop trip on the CG road.

After all these things done, we can combine the deep GP with rule-based method to do decision-making through interaction with the surrounding environment. Deep GP model generates a reasonable action according to the status of vehicle at each step. After that, each action will be further filtered by the rule-based model to validate the feasibility of the value, resulting in either refining the action or ignoring the suggested action since it fails the condition check in the rule set. It turn out that our method can present a relatively nice loop trip in simulation environment.

IV. Discussion

In the section above, we achieve successful loop trip with deep GP and rule-based method. Here, The performance is compared with DDPG method. The reward function we used is as follows:

$$v_x * \cos(\psi) * (1 - \sin(|\psi|)) * (1 - |d_2|). \tag{11}$$

The reward function can be constructed more reasonably by including other related variabels [21]. After calculating the rewards of these two methods, we find that our method needs 70 steps more to finish the loop trip and the total reward is about 5000 less than the DDPG well-trained network. However, the DDPG well-trained network, which is used to get training data, costs about several dozens of hours to train, and we only takes about 1.5 hours to train deep GP model. It also needs much less time than well-trained network with AMDDPG method according to the result in the paper [21]. In DDPG and AMDDPG methods, they need to interact with environment in each episode to update the new training data, and this procedure will be repeated for multiple times for exploration and exploitation. Thus, these data-hungry approaches need tens of thousands of data, but the training data we used only contains about 340 items, which is far more less than what is required. With these benefits, we believe that this method is a promising manner to be utilized in decision-making development in both simulation environment and in real road condition. However, there are many technical problems to tackle to achieve the real road test. Admittedly, the shortcomings of the proposed methods should also be acknowledged. In this paper, we only test the method in a relative simple road. In a complex road condition, it is obvious that more data should be fed into deep GP network and the man-made rules also need to be dedicatedly designed and validated.

To conclude, auxiliary rule-based approaches in our method, though being widely adopted at the current stage, are still not flexible and reliable solutions to the practical self-driving technology. We hope to find a more flexible method to solve the problem in a more mathematical and general way. In our future research, we may use dynamical bayesian network [22], bayesian structure time series model [23] or other not data-hungry methods to solve the decision-making problem in self-driving field with more complex road environments.

V. Conclusion

In conclusion, we present a method which combines deep gaussian process with rule-based method to solve self-

driving decision-making problem in a simulation environment. Our method costs much less time and need much less data than the DDPG well-trained network and can realize successful loop trip in a relative simple road. We think this method can be extended to do much more complex road tests. In the future, we will be committed to developing better methods for decision-making problem.

ACKNOWLEDGMENT

I would like to thank Junta Wu's, who is the author of AMDDPG, help for training the DDPG network, supplying the data and providing many helpful clarifications and suggestions. This work was supported by Shenzhen Engineering laboratory on Autonomous Vehicles, NSFC 61672512, and the Shenzhen S&T Funding with Grant No. JCYJ20160510154531467 and No. JCYJ20180507182628567.

REFERENCES

[1] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[2] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller *et al.*, "Making bertha drive-an autonomous journey on a historic route," *IEEE Intelligent transportation systems magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[3] A. G. Cunningham, E. Galceran, R. M. Eustice, and E. Olson, "Mpdm: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1670–1677.

[4] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[5] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship: Competition software manual," *arXiv preprint arXiv:1304.1672*, 2013.

[6] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[7] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[8] A. Damianou, "Deep gaussian processes and variational propagation of uncertainty," Ph.D. dissertation, University of Sheffield, 2015.

[9] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 2, pp. 408–423, 2013.

[10] J. Futoma, "Gaussian process-based models for clinical time series in healthcare," Ph.D. dissertation, Duke University, 2018.

[11] M. A. Osborne, R. Garnett, and S. J. Roberts, "Gaussian processes for global optimization," in *3rd international conference on learning and intelligent optimization (LION3)*, vol. 2009, 2009.

[12] R. Garnett, S. Ho, and J. Schneider, "Finding galaxies in the shadows of quasars with gaussian processes," in *International Conference on Machine Learning*, 2015, pp. 1025–1033.

[13] S. Thewes, M. Lange-Hegermann, C. Reuber, and R. Beck, "Advanced gaussian process modeling techniques," *Design of Experiments (DoE) in Engine Development. Expert Verlag*, 2015.

[14] A. Damianou and N. Lawrence, "Deep gaussian processes," in *Artificial Intelligence and Statistics*, 2013, pp. 207–215.

[15] D. Duvenaud, "Automatic model construction with gaussian processes," Ph.D. dissertation, University of Cambridge, 2014.

[16] H. Rue, S. Martino, and N. Chopin, "Approximate bayesian inference for latent gaussian models by using integrated nested laplace approximations," *Journal of the royal statistical society: Series b (statistical methodology)*, vol. 71, no. 2, pp. 319–392, 2009.

[17] C. Zhang, J. Butepage, H. Kjellstrom, and S. Mandt, "Advances in variational inference," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[18] M. D. Hoffman and A. Gelman, "The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014.

[19] 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.

[20] GPy, "GPy: A gaussian process framework in python," http://github.com/SheffieldML/GPy, since 2012.

[21] J. Wu and H. Li, "Aggregated multi-deep deterministic policy gradient for self-driving policy," in *International Conference on Internet of Vehicles*. Springer, 2018, pp. 179–192.

[22] E. Benhamou, J. Atif, R. Laraki, and A. Auger, "A new approach to learning in dynamic bayesian networks (dbns)," *Available at SSRN 3304849*, 2018.

[23] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, S. L. Scott *et al.*, "Inferring causal impact using bayesian structural time-series models," *The Annals of Applied Statistics*, vol. 9, no. 1, pp. 247–274, 2015.