Uncertainty and Prediction in Model-based Reinforcement Learning

Jan Bieser[1]

***Abstract*— This work studies different possibilities to capture model uncertainty for model-based reinforcement learning. The advantages and disadvantages of each method are shown. At first, Gaussian processes and Bayesian neuronal networks are described. These models incorporate uncertainty directly but are not really suitable for complex task in terms of scalability and performance. For this reason there exist methods to approximate model uncertainty. As Bayesian approximation methods, model ensembles and dropout are examined in detail.**

I. Introduction

Reinforcement learning holds the promise of automating a wide range of decision making and control tasks in many different application areas. Model-based reinforcement learning approaches make it possible to solve complex tasks given just a few training samples. Therefore reinforcement learning can also be applied in real world scenarios where the collection of training samples is expensive and possibly dangerous. However, many current learning methods ignore the uncertainty of the predictions. Especially in safety critical systems this can have catastrophic consequences. For this reason it is necessary to capture model uncertainty and taking it into account when taking decisions.

II. Model-based reinforcement learning

The model-based reinforcement learning problem is based on a Markov decision process (MDP) [1]. We denote a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$ and a transition function f to calculate the next state by $s_{t+1} = f(s_t, a_t)$. With a bounded reward function $r = (s, a)$ each state transition is assigned a reward to specify the task to be executed. The goal of reinforcement learning is to maximize the cumulated reward over a sequence of actions called trajectory. In model-based reinforcement learning we learn the dynamics model f in order to minimize a cost function. The learned dynamics model's predictions can either be used to learn a policy or to perform online planning to select optimal actions. In contrast to this model-free reinforcement learning algorithms estimate a optimal policy by exploring the environment without knowing the state transition function and what reward it will receive.

A. Learning the model

Nagabandi et. al. [2] propose a very basic model-based reinforcement learning algorithm (Alg. 1). First of all, random actions are executed at each timestep and the trajectories $\tau = (s_0, a_0, ..., s_{t-2}, a_{t-2}, s_{t-1})$ are saved to the training dataset $\mathcal{D}$. Some additional data preprocessing is necessary to slice the trajectories $\{\tau\}$ into training inputs (s_t, a_t) and corresponding output labels $s_{t+1} - s_t$. Moreover zero mean Gaussian noise is added to the training data in order to increase model robustness. Given the training dataset $\mathcal{D}$ the dynamics model $\hat{f}_\theta(s_t, a_t)$ is trained by minimizing the mean squared error

$$\mathcal{E}(\theta) = \frac{1}{\mid \mathcal{D} \mid} \sum_{(s_t, a_t, s_{t+1}) \in \mathcal{D}} \frac{1}{2} \parallel (s_{t+1} - s_t) - \hat{f}_\theta(s_t, a_t) \parallel^2 \tag{1}$$

using stochastic gradient descent. The vector θ represents the weights of the network. The training progress is validated on a validation dataset $\mathcal{D}_{val}$, which is not used for training.

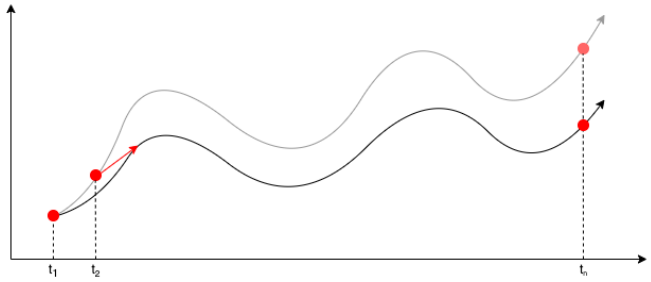

Fig. 1: An optimal trajectory is predicted in t_1. The MPC executes only the first action of the trajectory and then recalculates the trajectory again in t_2. We have followed this procedure until we achieve our goal t_n. Using MPC improves the robustness of dynamics models regarding model inaccuracies.

B. Planning with the model

After training, the trained model can be used to select actions that perform a task which is defined with the reward function $r = (s_t, a_t)$. Nagabandi et. al. [2] use like many other recent applications [3], [4], [5] the iterative approach model predictive control (MPC) to solve this optimization. Using the learned dynamics model a optimal sequence of actions $A_t^{(H)} = (a_t, ..., a_{t+H-1})$ for the planning horizon H is estimated:

$$A_t^{(H)} = \underset{A_t^{(H)}}{\operatorname{argmax}} \sum_{t'=t}^{t+H-1} r(\hat{s}_{t'}, a_{t'}) \quad : \quad \hat{s}_t = s_t, \hat{s}_{t'+1} = \hat{s}_{t'} + \hat{f}_\theta(\hat{s}_{t'}, a_{t'}) \tag{2}$$

After a optimal trajectory is calculated, the policy will only execute the first action a_t. With the updated state information s_{t+1} the optimal trajectory is calculated again. Because calculating an optimal trajectory is difficult due to nonlinear dynamics and reward functions, a simple random sampling

[1]The authors are with FZI Research Center for Information Technology, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany uoqox@student.kit.edu

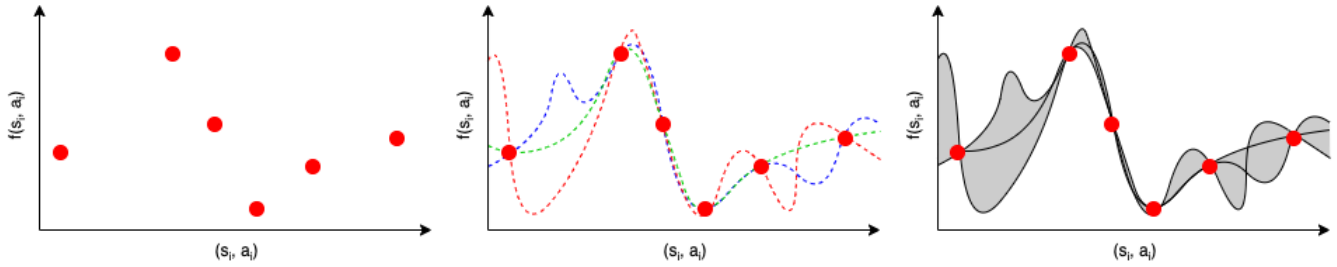

Fig. 2: Given a small set of transitions (left), there exist multiple deterministic transition functions which fit the data (center). In contrast to a single deterministic transition function, probabilistic transition functions (right) incorporate uncertainty in their predictions. This is important in areas where little data is available [6].

shooting method [7] is used. The randomly generated action sequence with the highest expected cumulative reward is chosen. The sampled trajectory is then added to the dataset $\mathcal{D}_{RL}$ to update the dynamics model $\hat{f}_\theta$ with each iteration. Fig. 1 shows the recalculation of the sequence of actions after the first action was executed. [8] demonstrates that using MPC in model-based reinforcement learning improves the robustness of dynamics models regarding model inaccuracies. Nagabandi et. al. [2] uses the dynamics model to generate expert trajectories for training a neural network policy which is then used as an initial policy for a model-free reinforcement learning algorithm.

Algorithm 1 Model-based Reinforcement Learning

```
gather dataset D_RAND of random trajectories
initialize empty dataset D_RL, and randomly initialize f̂_θ
for iter=1 to max_iter do
    train f̂_θ(s,a) by performing gradient descent on Eqn. 1 using D_RAND and D_RL
    for t = 1 to T do
        get agent's current state s_t
        use f̂_θ to estimate optimal action sequence A_t^(H) (Eqn. 2)
        execute first action a_t from selected action sequence A_t^(H)
        add (s_t, a_t) to D_RL
    end for
end for
```

C. Advantages and disadvantages of model-based reinforcement learning

Recent work has shown that model-free reinforcement learning algorithms can archive a very good performance on complex tasks [9]. A popular example is the highly challenging board game Go where a model-free reinforcement learning algorithm in combination with supervised learning from human expert games managed to outperform a human professional player [10]. Furthermore model-free reinforcement learning algorithms are simple by design, easy to implement and can generally be applied to different tasks. However a large amount of training data is necessary to archive a good performance. This sample inefficiency is a major barrier to apply model-free reinforcement learning in complex real-world environments, such as robotics [11]. In contrast to that, model-based reinforcement learning algorithms are very data efficient, because very few training samples are sufficient to train the dynamics model. Moreover flexible tasks, such as moving a pencil to follow arbitrary user specified strokes can be learned [3]. Despite all, model-based methods are not widely used in practice because they suffer from model bias. This means that the learned dynamics model does not resemble the real environment sufficiently. Especially in complex environments where only a few data samples are available, model bias is a serious issue because the prediction of trajectories can be very inaccurate. Nevertheless, the prediction is made with absolute confidence and uncertainties in the dynamics model are not taken into account [6].

D. Uncertainty-aware model-based reinforcement learning

Prior work has shown that it is necessary to capture model uncertainty to avoid the problems mentioned above, especially if high capacity models are used [12], [5], [13], [9], [4]. There are two types of uncertainty. The first type, aleatoric uncertainty, is caused by noise in the observations. An example for aleatoric uncertainty could be sensor noise which is caused by contamination of the sensor. In that case, even if more observations are available for training, it is not possible to reduce the uncertainty [14]. On the other hand, epistemic uncertainty, refers to the uncertainty about the parameters of the dynamics model. Due to a lack of data it is not possible to determine the underlying systems dynamics exactly, but collecting more training samples would reduce the epistemic uncertainty [12]. In the following sections different possibilities of uncertainty-aware models are described.

III. Gaussian Processes

The effect of model bias is shown in Fig. 2. When learning on a small dataset of transitions (left), there exist multiple deterministic transition functions which fit the data (center). In areas where not enough data is available, the prediction of a single deterministic transition function may be inaccurate but is claimed with full confidence. Especially when making long term predictions or sampling trajectories

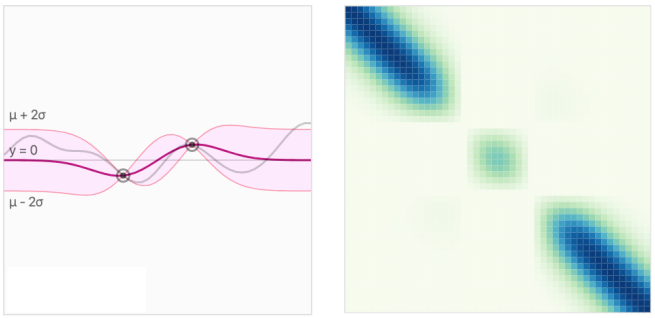

Fig. 3: The distribution of a Gaussian process (left) is described by the a covariance matrix (right). The uncertainty of the prediction is low in areas where training data is available. As we move further away from training points, the uncertainty grows [15].

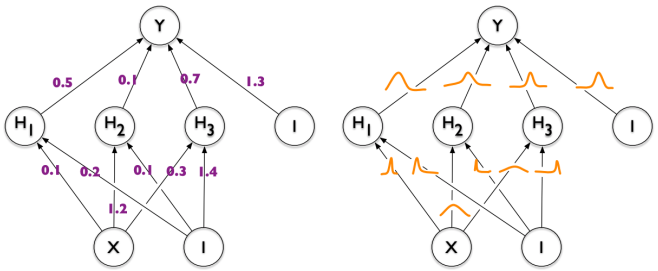

Fig. 4: Conventional neural networks have fixed weight values (left). The weights of a Bayesian neural network (right) weights are represented by a probability distribution over possible values. This introduces uncertainty in the weights of neural networks [17].

from the model this becomes critical. Probabilistic transition functions on the other hand incorporate uncertainty in their predictions [6]. Gaussian processes are probability distribution over possible functions. Given prior knowledge, they let us make predictions about our data. In particular, there exist infinitely many functions that fit a given set of training data. By assigning a probability to each possible function, the mean of this probability distribution would be the most probable representation of the data. In addition to this, Gaussian processes let us directly assess the uncertainty of our prediction. If all functions from the distribution agree in their prediction, the confidence of the model is high. On the other hand, if there is disagreement about the predictions, the uncertainty of the model is high [15]. The covariance matrix Σ describes the shape of the distribution and the characteristics of the predicted function. Σ is generated by evaluating the kernel k pairwise on all points:

$$k : \mathbb{R}^n \times \mathbb{R}^n \rightarrow, \quad \Sigma = Cov(X, X') = k \tag{3}$$

The entry $\Sigma_{i,j}$ defines the correlation between, the i-th and j-th point. As the kernel function k describes the similarity between the function values, it controls the shape of the fitted function [15]. Figure 3 shows the prior distribution of a Gaussian process and the corresponding covariance matrix. We observe that the uncertainty of the prediction is low in areas where training data is available. If the predicted point lies close to training data, there exists a low correlation with other points. By contrast, as we move further away from training points, the uncertainty grows. The best explanation of the training data is given by the mean function of the function distribution.

A. Applications of Gaussian processes in RL

With PILCO, a practical, data-efficient model-based policy search method, Deisenroth and Rasmussen [6] make use of Gaussian processes in order to reduce model bias and incorporate model uncertainty into long-term planning. PILCO's policy search is a 3-step process. Initially, a dynamics model is fitted to the observed transitions data. Then the policy is evaluated by using dynamics model predictions of future states and costs. Finally, the policy is improved.

B. Advantages and disadvantages of Gaussian processes

Gaussian processes are able to generalize well from small datasets. Then the predictions and the uncertainty measure are accurate. When it comes to larger datasets, performance is a major issue in applying GP methods [16]. Furthermore, finding a kernel which is well suited for discontinuous dynamics is challenging [12].

IV. Bayesian neural networks

In contrast to Gaussian processes, deep neural networks scale well in high dimensional observation spaces. Bayesian learning introduces uncertainty in the weights of neural networks [17]. Figure 4 shows the difference between conventional neural networks and Bayesian neural networks. Instead of having weights with fixed values, Bayesian neural networks weights are represented by a probability distribution over possible values. Given a Bayesian neural network with L layers and a set of weight matrices $\mathcal{W} = \{W_l\}_{l=1}^{L}$ between the fully connected layers, the likelihood for the network weights $\mathcal{W}$ and the noise precision γ, with data $\mathcal{D} = (X, y)$ is then

$$p(y|\mathcal{W}, X, \gamma) = \prod_{n=1}^{N} \mathcal{N}(y_n|f(x_n; \mathcal{W}), \gamma^{-1}) \tag{4}$$

Moreover a Gaussian prior distribution for each weight in each weight matrix is specified by

$$p(\mathcal{W}, \lambda) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l}|0, \lambda^{-1}) \tag{5}$$

where $w_{ij,l}$ is the weight in the e i-th row and j-th column of the weight matrix W_l. λ is a precision parameter. After that the posterior distribution for the weight matrix $\mathcal{W}$, the noise precision γ and λ can be obtained by applying Bayes' rule:

$$p(\mathcal{W}, \gamma, \lambda|\mathcal{D}) = \frac{p(y|\mathcal{W}, X, \gamma)p(\mathcal{W}|\lambda)p(\lambda)p(\gamma)}{p(y|X)} \tag{6}$$

where $p(y|X)$ is a normalization constant. To make predictions y_* the predictive distribution can be used with a new input vector x_*:

$$\int p(y_*|x_*, \mathcal{W}, \gamma)p(\mathcal{W}, \gamma, \lambda|\mathcal{D})d\gamma d\lambda d\mathcal{W} \tag{7}$$

Hernández-Lobato and Adams [18] propose an approach to learn Bayesian neural networks by probabilistic backpropagation which is significantly faster than other techniques.

A. Applications of Bayesian neural networks in RL

Deep PILCO [19] extends PILCO's framework [6] to use Bayesian deep dynamics models with approximate variational inference in order to improve the scalability in terms of number of trials and observation space dimensionality. Because simple neural networks cannot express output model uncertainty, a Bayesian neural network is used. To handle input uncertainty, a input distribution is fed into the dynamics model.

B. Advantages and disadvantages of Bayesian neural networks

The advantages of Bayesian neural networks are very convincing. The uncertainty is naturally incorporated in the network and propagated into predictions. By averaging over parameter values, model overfitting can be avoided [18]. Thus Bayesian neural networks are applicable in scenarios where not much training data is available. Moreover a large class of stochastic functions can be modelled [20]. Nevertheless, exact Bayesian inference is not possible as the number of parameters is very large. For this reason current research refers to methods for approximating Bayesian inference [17].

V. Model ensembles

Ensemble learning is the procedure of combining the outputs of multiple models to a single prediction in order to increase the accuracy of the prediction. Empirical studies have shown that ensemble models archive a better accuracy than single models. In real-world scenarios, every model tends to make false predictions. The principle of ensemble learning is that combining the strengths and weaknesses of the models will lead to the best overall prediction [21]. In model-based RL, model ensembles can provide plausible uncertainty estimates [12]. We consider that the model ensemble $\tilde{f}_{\theta_b}$ is composed out of a set of bootstrap models with the parameters θ_b. Each bootstrap model has its own data set $\mathbb{D}_b$ which is sampled from the recorded dynamics data set. [12]

A. Applications of model ensembles in RL

1) Stochastic Ensemble Value Expansion: Buckman et. al. [22] introduce model ensembles to extend the approach model-based value expansion (MVE) an algorithm for incorporating predictive models of system dynamics into model-free value function estimation proposed by Feinberg et. al. [9]. The learned dynamics model provides a source of additional data in order to reduce sample complexity. In contrast to MVE, interpolations between many different horizon lengths are made for rollout generation. The horizon lengths with less uncertainty and thus less error are favoured. The uncertainty is estimated by computing the variance under samples from the ensemble. Given a H step trajectory from each of the M models, each trajectory consists out of H states, the empirical mean $\mathcal{T}_i^{\mu}$ and variance $\mathcal{T}_i^{\sigma^2}$ for each partial rollout can be computed. A single target can be generated by weighting the inverse variance of the means:

$$\mathcal{T}_H(r, s') = \sum_{i=0}^{H} \frac{\tilde{w}_i}{\sum_j \tilde{w}_j} \mathcal{T}_i^{\mu}, \quad \tilde{w}_i^{-1} = \mathcal{T}_i^{\sigma^2} \tag{8}$$

2) Model-Ensemble Trust-Region Policy Optimization: In order to tackle the problem that policy optimization tends to exploit regions where not enough data is available (model bias), Kurutach et. al. [23] propose to regularize the policy updates using an ensemble of models which takes model uncertainty into account. First, a set of dynamics models $\hat{f}_\theta = \{f_{\theta 1}, ..., f_{\theta K}\}$ is trained with supervised learning using the same real world data. The models only differ in weight initialization and sampling order. Second, the gradient for policy optimization is estimated. For trajectory simulation a random model is chosen to predict the next state given the current state and action. This leads to more stable learning because the policy does not overfit to any single model. Third, the model ensemble is used to monitor the policy's performance by computing the ratio of K models in which the policy improves:

$$\frac{1}{K} \sum_{k=1}^{K} 1[\hat{\eta}(\theta_{new}; \phi_k) > \hat{\eta}(\theta_{old}; \phi_k)] \tag{9}$$

If the ratio falls below the threshold, only a small number of updates is tolerated. These three steps are repeated until the desired performance in the real environment is reached.

3) Model-Based Meta Policy-Optimization: Another approach which makes use of model ensembles but in a very different way is proposed by Clavera et. al. [11]. Gradient based meta learning is used to learn a policy which can adapt quickly to any model of an ensemble of learned dynamic models. The goal of meta learning is to learn optimal policies from a distribution of tasks (MDPs). While tasks may differ in their transition function and reward function, they share the same action space and state space. In this case a set of dynamics models $\hat{f}_\theta = \{f_{\theta_1}, ..., f_{\theta_K}\}$, each initialized with random weights and trained with a randomly selected subset $\mathcal{D}_k$ of the data set $\mathcal{D}$, approximate the environment. The meta-optimization problem is described as follows:

$$\max_{\theta} \quad \frac{1}{K} \sum_{k=0}^{K} J_k(\theta'_k) \quad s.t.: \quad \theta'_k = \theta + \alpha \nabla_\theta J_k(\theta) \tag{10}$$

$J_k(\theta)$ is the expected return under the policy π_θ and the estimated dynamics model $\hat{f}_{\theta k}$.

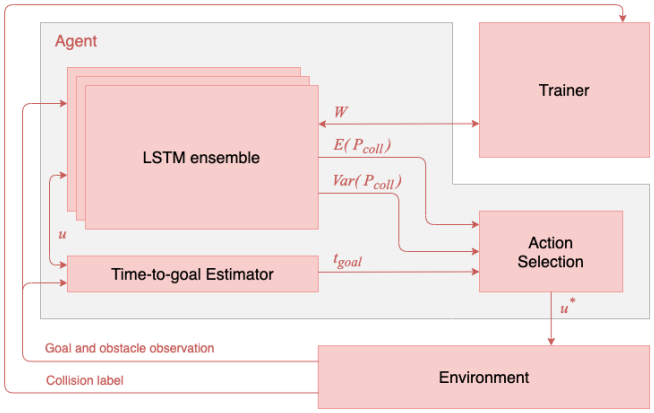

Fig. 5: System architecture. A LSTM ensemble acquires a sample mean and variance collision probability. Based on these measures the system selects the safest motion primitive with the minimal joint cost [5].

4) Safe reinforcement learning: Figure 5 shows the system architecture of an application of model ensembles in a real world scenario for uncertainty-aware navigation around pedestrians [5]. An ensemble of LSTM models, which are the state-of-the-art model in this use case, predict the probability of two moving objects collide. The models are randomly initialized and trained on datasets from different but overlapping sections of the observation space. As a result, each models predictions are similar for common observation but differ for uncommon ones. A distribution over the ensembles prediction is acquired to calculate the sample mean and variance. These measures give an indication of the uncertainty of the predictions. After retrieving the model uncertainty, it is possible for a Model Predictive Controller (MPC) to select the safest motion primitive with the minimal joint cost:

$$u^*_{t:t+h} = \underset{u \epsilon U}{\operatorname{argmin}} (\lambda_v Var_{\mathcal{N}}(P^i_{coll}) + \lambda_c E_{\mathcal{N}}(P^i_{coll}) + \lambda_g t_{goal}) \tag{11}$$

The time goal of the target motion sequence t_{goal} is estimated by measuring the straight line distance between starting point and target. The cost terms are weighted by λ. As seen in Fig. 6, the prediction of a collision with a known obstacle is accurate and the uncertainty of the prediction is low. In scenarios with novel obstacles, the network may fail to generalize. Nevertheless, the network indicates a high uncertainty in its predictions, meaning that the these are false-positives and should not to be trusted (Fig. 7).

B. Advantages and disadvantages of ensemble learning

Even though there are methods providing better uncertainty estimates, ensemble learning is a common choice in other literature to capture model uncertainty [22]. The reason is that it training an ensemble of models is easy and despite the simple approach reasonable uncertainty estimates can be made [12]. Moreover, ensemble learning is applicable to a wide range of different models [22].

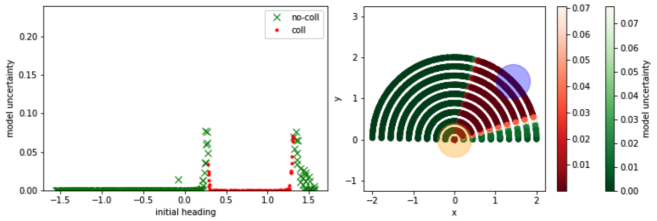

Fig. 6: The prediction of a collision with a known obstacle is accurate and the uncertainty of the prediction is low [5].

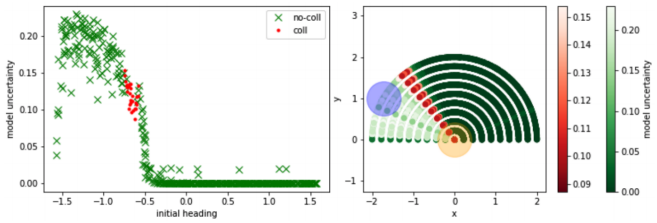

Fig. 7: The prediction of a collision with a unknown obstacle may be inaccurate and the uncertainty of the prediction is high. That means that these predictions are false-positives and should not to be trusted [5].

VI. Dropout as a Bayesian Approximation

When training neural networks, dropout is commonly used to reduce overfitting [4]. Dropout is realized by temporarily removing units along with its incoming and outgoing connections from the network. Figure 8 shows how the network looks like before and after applying dropout. To formally describe a dropout network, consider a neural network with L hidden layers and the layer index $l \in \{1, ..., L\}$. $z^{(l)}$ and $y^{(l)}$ are the input and output vector of layer l. Additionally, $w^{(l)}$ and $b^{(l)}$ denote the weights and biases at layer l. The feed-forward operation of the neuronal network with dropout can be described as:

$$\begin{aligned} r^{(l)}_j &\sim Bernoulli(p) \\ \widetilde{y}^{(l)} &= r^{(l)} * y^{(l)} \\ z^{(l+1)}_i &= w^{(l+1)}_i \widetilde{y}^{l} + b^{(l+1)}_i \\ y^{(l+1)}_i &= f(z^{(l+1)}_i) \end{aligned} \tag{12}$$

In the equation above, the thinned outputs $\widetilde{y}^{(l)}$ for layer are generated by multiplying the vector of independent Bernoulli

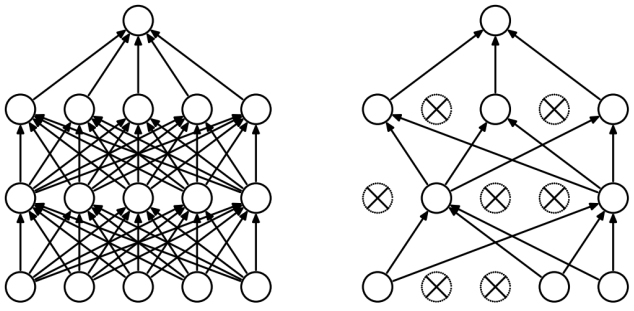

Fig. 8: Network before applying dropout (left). Network after applying dropout (right). Dropout temporarily removes units along with its incoming and outgoing connections from the network [24].

random variables, each of which has the probability p of being 1, with the outputs $y^{(l)}$ of any layer l [24]. Usually, dropout is only applied during training in order to prevent overfitting but Gal and Ghahramani [25] showed that dropout can be used to obtain uncertainty estimates at test time. T feed-forward operations are performed in the network with L layers and the mean and average of the prediction can be calculated. Therefore we estimate:

$$\mathbb{E}_{q(y^*|x^*)}(y^*) \approx \frac{1}{T}\sum_{t=1}^{T}\widehat{y}^*(X^*, W_1^t, ..., W_L^t) \tag{13}$$

with T sets of weight matrices $\{W_1^t, ..., W_L^t\}_{t=1}^T$ sampled from the Bernoulli distribution $\{z_1^t, ..., z_L^t\}_{t=1}^T$. The models predictive variance (model uncertainty) can be calculated as follows:

$$\begin{aligned} &Var_{q(y^*|x^*)}(y^*) \approx \tau^{-1} I_D \\ &+ \frac{1}{T}\sum_{t=1}^{T}\widehat{y}^*(X^*, W_1^t, ..., W_L^t)^T \widehat{y}^*(X^*, W_1^t, ..., W_L^t) \\ &- \mathbb{E}_{q(y^*|x^*)}(y^*)^T \mathbb{E}_{q(y^*|x^*)}(y^*) \end{aligned} \tag{14}$$

The models predictive variance equals the sample variance of T stochastic forward passes through the network plus the inverse model precision.

A. Applications of dropout as Bayesian approximation

Kahn et. al. [4] consider a mobile robot navigating in an unknown environment. In order to avoid collision with blocking objects, the probability of collision is provided together with an estimate of the predictions uncertainty. Therefore bootstrapping and dropout is applied to estimate $\mathbb{E}[f_\theta(x_t, u_{t:t+H}, ot)]$ and $Var[f_\theta(x_t, u_{t:t+H}, ot)]$ for the collision prediction model:

$$\begin{aligned} &P_\theta(COLL|x_t, u_{t:t+H}, ot)) = \\ &L(\mathbb{E}[f_\theta(x_t, u_{t:t+H}, ot)] \\ &+ \lambda_{STD}\sqrt{Var[f_\theta(x_t, u_{t:t+H}, ot)]}) \end{aligned} \tag{15}$$

where the input parameters are the current state x_t, the observation o_t and a sequence of H controls $u_{t:t+H}$. L is the logistic function $L(y) = 1/(1 + exp(-y))$. The collision prediction influences the cost function which is used by MPC to sample trajectories according to

$$\begin{aligned} &\mathcal{C}(x_{t+H}, u_{t+H}) \approx \mathcal{C}_{TASK}(x_{t+H}, u_{t+H}) \\ &+ P_\theta(COLL|x_t, u_{t:t+H}, ot)\mathcal{C}_{COLL}(x_{t+H}) \end{aligned} \tag{16}$$

The collision cost C_{COLL} depends on the velocity of the mobile robot VEL_t and a weighting constant θ_{coll}, that weights the relative importance of C_{COLL} to C_{TASK}, according to

$$C_{COLL}(x_t) = \lambda_{coll}\|VEL_t\|^2 \tag{17}$$

As a result, the mobile robot is encouraged to move slowly if the collision probability prediction P_θ is large. In the other case, if P_θ is small, fast speeds are possible.

B. Advantages and disadvantages of using dropout as Bayesian approximation

Dropout is a computationally cheap method estimate the uncertainty of model predictions and thus represents an alternative to ensemble learning. However, dropout underestimates the uncertainty [4]. Moreover Bernoulli dropout is only one example dropout technique to estimate model uncertainty. Using different techniques would differ in their results [25].

VII. Conclusion

This work has studied different possibilities to capture model uncertainty to avoid the problem of model bias. Especially in complex environments where only a few data samples are available, model bias is a serious issue because predicting trajectories can be very inaccurate. Gaussian processes let us directly assess the uncertainty of our prediction. When learning on a small datasets of transitions the predictions and the uncertainty measure are accurate. Due to performance issues it is not possible to apply Gaussian processes on larger datasets. In contrast to Gaussian processes, Bayesian neural networks scale well in high dimensional observation spaces. Because the number of parameters is very large, exact Bayesian inference is not possible. For this reason there exist methods to approximate Bayesian inference. Ensembles of models can provide plausible uncertainty estimates. Due to the simple implementation this method is widely used in current research. Another approach to estimate model uncertainty by applying dropout at test time. Dropout as Bayesian approximation is computationally cheap and can also be combined with ensemble learning.

REFERENCES

[1] R. Bellman, "A Markovian Decision Process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, jan 1957. [Online]. Available: http://www.jstor.org/stable/24900506

[2] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning," *CoRR*, vol. abs/1708.0, 2017. [Online]. Available: http://arxiv.org/abs/1708.02596

[3] A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar, "Deep Dynamics Models for Learning Dexterous Manipulation," in *Conference on Robot Learning (CoRL)*, 2019.

[4] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-Aware Reinforcement Learning for Collision Avoidance," *CoRR*, vol. abs/1702.0, 2017. [Online]. Available: http://arxiv.org/abs/1702.01182

[5] B. Lütjens, M. Everett, and J. P. How, "Safe Reinforcement Learning with Model Uncertainty Estimates," *CoRR*, vol. abs/1810.0, 2018. [Online]. Available: http://arxiv.org/abs/1810.08700

[6] M. Deisenroth and C. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search." 2011, pp. 465–472.

[7] A. Rao, "A Survey of Numerical Methods for Optimal Control," *Advances in the Astronautical Sciences*, vol. 135, 2010.

[8] S. Kamthe and M. P. Deisenroth, "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control," *CoRR*, vol. abs/1706.0, 2017. [Online]. Available: http://arxiv.org/abs/1706.06491

[9] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning," *CoRR*, vol. abs/1803.0, 2018. [Online]. Available: http://arxiv.org/abs/1803.00101

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, 2016.

[11] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-Based Reinforcement Learning via Meta-Policy Optimization," *CoRR*, vol. abs/1809.0, 2018. [Online]. Available: http://arxiv.org/abs/1809.05214

[12] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models," *CoRR*, vol. abs/1805.1, 2018. [Online]. Available: http://arxiv.org/abs/1805.12114

[13] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran, "Epopt: Learning robust neural network policies using model ensembles," *CoRR*, vol. abs/1610.01283, 2016. [Online]. Available: http://arxiv.org/abs/1610.01283

[14] A. Kendall and Y. Gal, "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" *CoRR*, vol. abs/1703.0, 2017. [Online]. Available: http://arxiv.org/abs/1703.04977

[15] J. Görtler, R. Kehlbeck, and O. Deussen, "A Visual Exploration of Gaussian Processes," *Distill*, vol. 4, no. 4, p. e17, apr 2019. [Online]. Available: https://distill.pub/2019/visual-exploration-gaussian-processes

[16] J. Wang, A. Hertzmann, and D. J. Fleet, "Gaussian Process Dynamical Models," in *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, and J. C. Platt, Eds. MIT Press, 2006, pp. 1441–1448. [Online]. Available: http://papers.nips.cc/paper/2783-gaussian-process-dynamical-models.pdf

[17] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 1613–1622.

[18] J. M. Hernández-Lobato and R. P. Adams, "Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 1861–1869.

[19] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving PILCO with Bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, International Conference on Machine Learning*, 2016.

[20] S. Ghosh, J. Yao, and F. Doshi-Velez, "Structured Variational Learning of {B}ayesian Neural Networks with Horseshoe Priors," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1744–1753. [Online]. Available: http://proceedings.mlr.press/v80/ghosh18a.html

[21] G. Brown, "Ensemble Learning Motivation and Background," Tech. Rep., 2010.

[22] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion," *CoRR*, vol. abs/1807.0, 2018. [Online]. Available: http://arxiv.org/abs/1807.01675

[23] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-Ensemble Trust-Region Policy Optimization," *CoRR*, vol. abs/1802.1, 2018. [Online]. Available: http://arxiv.org/abs/1802.10592

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[25] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 2016, pp. 1050–1059. [Online]. Available: http://proceedings.mlr.press/v48/gal16.html