Sampling diverse neural networks for exploration in reinforcement learning

Maxime Wabartha
School of Computer Science
McGill University
maxime.wabartha@mail.mcgill.ca

Audrey Durand
School of Computer Science
McGill University
audrey.durand@mcgill.ca

Vincent François-Lavet
School of Computer Science
McGill University
vincent.francois-lavet@mcgill.ca

Joelle Pineau
School of Computer Science
McGill University
jpineau@cs.mcgill.ca

Abstract

Current methods providing neural networks with uncertainty struggle with drawing diverse samples from the network posterior prediction distribution. In this work, we introduce an approach for sampling various functions from the posterior predictive distribution of a neural network by considering in the loss a *repulsive constraint* applied in the function space. We illustrate the usefulness of the method on a 1D regression problem, as well as reinforcement learning problems where exploring unseen regions of the state space is required.

1 Introduction

Being able to quantify the uncertainty of a predictive model in different regions of the input space can benefit to many applications. For example, reinforcement learning approaches can rely on this information to conduct efficient exploration [1, 7, 8]. Gaussian processes are well known for the task as they provide a closed form posterior distribution over the target function, allowing the noise information and the richness of the function distributions to be controlled *a priori* through the choice of a kernel and its parameters. However, Gaussian processes struggle with scaling to large and high-dimensional training datasets [10].

On the contrary, while neural networks are able to circumvent scaling issues, they do not come with an intrinsic definition of uncertainty. This has motivated researchers to provide new tools to estimate uncertainty over the predicted function [1, 3, 5]. In the context of regression, uncertainty can be represented by a numerical value associated with the model's degree of certainty over its output [6]. A network can directly output this value, or an ensemble of networks can be trained to compute an empirical predictive variance [9]. Uncertainty can also appear implicitly instead of being explicitly quantified. In variational inference, the weights of a neural network are sampled from their approximate posterior distribution [11]. One can also use bootstrapping to randomize the inputs [7]. Yet, the ability to control the resulting functions is limited compared to the facility of choosing a Gaussian process kernel, and efficient sampling and ensembling require variety in the functions [9].

In this work, we tackle the challenge of producing functions from a neural network architecture different from one another, given the same training samples. To this end, we introduce a loss that attempts to produce a function that predicts well the targets while simultaneously being different from a reference function in unknown regions of the input space. We present experiments showing that

Third workshop on Bayesian Deep Learning (NeurIPS 2018), Montréal, Canada.

the uncertainty estimated from these diverse functions does not suffer from the caveats of existing approaches, such as the controllability of the network uncertainty [3, 9].

2 Trading-off error and similarity

Methods for sampling pointwise function evaluations from Bayesian neural networks usually focus on finding a posterior distribution for the weights [4]. We propose here to work directly in the *function space* $\mathcal{H}$ by explicitly training toward the behaviour expected for the posterior distribution, *i.e.* generate functions that differ more in regions with lower density of training samples. Similarly to a Gaussian process, we want to enforce outputs of a neural network not to covary when their inputs as far from each other, *w.r.t.* a similarity metric. We justify this approach by stepping aside from the Bayesian formalism and making an analogy with the training of neural networks. We call the output of an untrained neural network a *quasi-prior* [9]. Training a neural network to obtain a function that is likely given the data (*i.e.* , passes near or through the training points) can be seen as applying the Bayes law using the observations in a dataset and sampling from the corresponding posterior.

Imposing an ensemble-based constraint to ensure variety would require to train the functions jointly. To enable a sequential training, we rather characterize the diversity constraint by assuming that function samples should differ from a *reference function* for inputs out of the training distribution, but have consistent values in the training distribution. The reference function is defined as the output of a regular neural network trained with a standard loss, such as the MSE loss.

Let $\mathcal{X}$ and $\mathcal{Y}$ respectively denote the input and output space and let $\mathcal{D} = \{(x_i, y_i)\}_{i=1\dots n}$ denote the set of training samples, with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Let $\mathbb{X}$ denote a set of input points sampled from $\mathcal{X}$, with $|\mathbb{X}| = n_{\mathbb{X}}$. Let $g : \mathcal{X} \mapsto \mathcal{Y}$ be a function from a space $\mathcal{H}$ mapping inputs to outputs, obtained with a neural network trained with a MSE loss on $\mathcal{D}$. Our goal is to find a function f, also from a space $\mathcal{H}$, that predicts well at the training samples ($\mathcal{D}$), while leading to different outputs (w.r.t a similarity measure) than the reference function g elsewhere ($\mathbb{X}$). We translate this characteristic by trading-off prediction error and functions similarity in the loss function:

$$f: \ \arg\min_f \mathcal{L}(f; g, \mathcal{D}, \mathbb{X}) = \underbrace{\frac{1}{n} \sum_{(x_i,y_i)\in\mathcal{D}} d(f(x_i), y_i))}_{\text{error loss}} + \underbrace{\frac{\lambda}{n_{\mathbb{X}}} \sum_{x\in\mathbb{X}} k(f(x), g(x))}_{\text{similarity loss}} \tag{1}$$

where $d : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ is a distance function and $k : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ is a kernel measuring the similarity between two values in $\mathcal{Y}$. In addition to considering the typical empirical error loss on $\mathcal{D}$, a parameter $\lambda \geq 0$ adds a penalization based on the similarity between f and g on $\mathbb{X}$. The intuition behind the similarity loss is to induce a *repulsion* between f and g outside the training samples. Observe that for $\lambda = 0$ and $d(y, y') = (y - y')^2$, Eq. 1 corresponds to the mean square error (MSE) loss.

Ideally, $\mathbb{X}$ should be sampled from the manifold where the observations are, out of the training distribution. However, the access to this distribution is challenging. Instead, we sample repulsive inputs located at the boundary between $\mathcal{D}$ and the rest of the space by adding noise to the training data [5]. Some of these samples will be out-of-distribution, while the repulsive effect of the samples in-distribution will be countered by the attractive effect of the empirical error loss. We adapt the noise to the nature of $\mathcal{X}$. Adding the similarity loss term requires to sample $\mathbb{X}$, compute $k(f(x), g(x))$ at all locations $x \in \mathbb{X}$, and perform a backpropagation for the repulsive inputs $\mathbb{X}$. Overall, we find that the added training time of f corresponds to the time required to perform an additional backpropagation.

3 Experiments

All experiments are conducted using the euclidean distance function for d and a Gaussian kernel function $k(f(x), g(x)) = \exp\left(-\frac{1}{2\sigma^2}||f(x) - g(x)||_2^2\right)$. The bandwidth σ controls the strength of the *repulsiveness* applied during training. We sample σ from a log-uniform distribution. Finally, the number $n_{\mathbb{X}}$ of inputs in $\mathbb{X}$ influences the locality (if the number is low) or globality (if the number is high) of the repulsiveness.

3.1 1D regression visualization

We first compare the diversity obtained with anchoring [9], input bootstrapping [2], dropout at inference time [3] and the proposed approach to tradeoff error and similarity. We consider a training set of 10 points sampled in $[-0.5, 0]$ from a bimodal ground truth function, such that the second mode lies out of the training distribution. We first train a neural network with a regular MSE loss on the training set until convergence. The resulting function g is used as the reference function for our proposed approach. Figure 1 shows the posterior predictive distribution function empirically estimated by taking, for each approach, the pointwise average and standard deviation over 50 sampled functions. We expect the empirical posterior predictive distribution to cover the ground truth function.

While we succeed to do so using a MSE loss and the proposed approach, we do not manage to obtain diverse functions using solely anchoring neither using dropout; in our experiments, changing the dropout rate did not improve the quality of the obtained uncertainty. Input bootstrapping does produce functions that better span the width of outputs, but it also disregards by nature certain points of the training set, where we expect the uncertainty to be low given our current knowledge. We also provide in the appendix an example of the functions generated by our function approach when fixing $\mathbb{X}$.

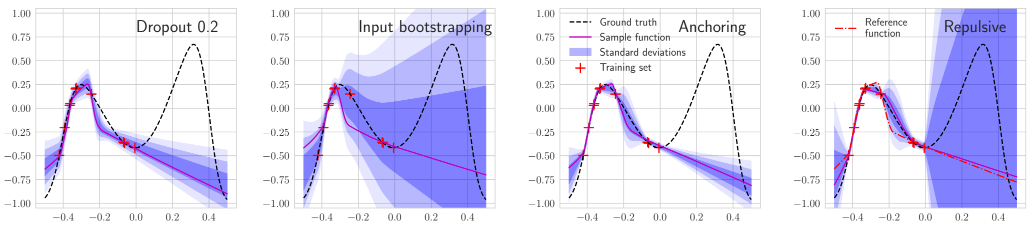

Figure 1: Comparison of the empirical (over 20 sample functions) posterior predictive distribution for dropout, input bootstrapping, anchoring and repulsive constraint.

3.2 Diverse functions in high-dimensional input space

We apply the method to function approximation in the case of a reinforcement learning problem requiring exploration. More precisely, we showcase how our method can help sample diverse reward functions in a model-based setting. We create a dataset of 43 13x13 frames with the associated reward. We use as function approximator a small CNN outputing a reward for a given frame (see appendix). To illustrate our method, we sample the repulsive points from possible frames, thus directly from the manifold, in or out of the training distribution (see appendix). Figure 2 (rightmost figure) shows how a sample function trained with a repulsive loss can approximate high rewards in unseen parts of the state space without having visited it while fitting well the seen states. Similarly, we conduct the same study for dropout and input bootstrapping. However, we do not manage in our experiments to get diverse reward functions out of the training distribution for other methods (see appendix, Figure 7).

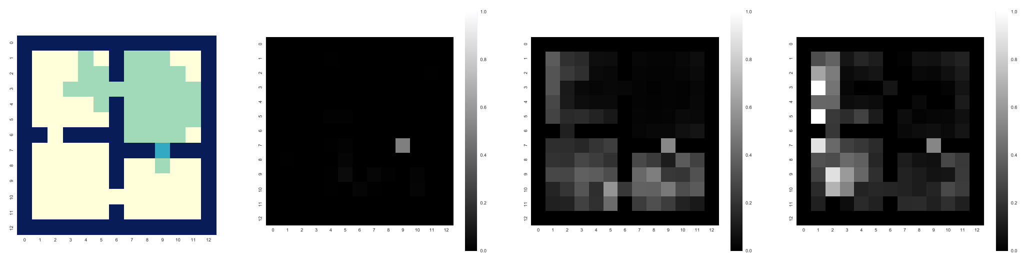

Figure 2: Left: 4-rooms environment, with visited states in teal, and the goal with a reward of 0.5 in light blue. Middle-left: reference function trained with a regular CNN, without the repulsive constraint. Middle-right: mean plus one standard deviation of an empirical distribution of reward functions, computed over 50 functions. Right: Example of one sampled reward function. Higher rewards are displayed in in white.

3.3 Application to exploration in reinforcement learning

Directed exploration in reinforcement learning requires to visit regions of the state-action space where the agent's knowledge is limited. Particularly in the case of model-based reinforcement learning, we expect the transition and reward function to provide information related to the uncertainty on the environment. We apply the repulsive loss to a simple 1D reinforcement learning problem where the state space is $[0, 1]$ and the agent can move by fixed displacements in the set $\{-3m, -2m, -1m, 0, 1m, 2m, 3m\}, m = \frac{1}{9}$. We use the ground truth function from Figure 1 as the reward function. The agent starts from 0 and necessarily encounters the suboptimal region. We adapt the PSRL algorithm, which iteratively solves a RL problem by sampling a Markov Decision Process (denoted MDP, here, a transition and reward function) from a MDP posterior distribution given the current training dataset $\mathcal{D}$. An optimal policy π with respect to the sampled MDP is computed, and new data points from the environment are generated using π and augment $\mathcal{D}$, before updating the posterior distribution over the MDPs. The reward and transition of the previous episode is used as the reference function. Results of the comparison with ϵ-greedy methods are given in Table 1 which shows the difficulties of a non diverse method in sampling various reward and transition functions. See appendix for additional details and figures about the algorithm and the experiment.

	% Success	Average discounted cumulative reward
Our algorithm	**94.7%**	**6.15**
ϵ-greedy (0.95)	31.6%	5.85
ϵ-greedy (0.9)	26.3%	5.94
ϵ-greedy (0.5)	26.3%	5.71

Table 1: Metrics comparison between our algorithm and ϵ-greedy methods, for 38 runs of 50 episodes. Numbers were rounded at 2 decimals. Success means that the optimal state w.r.t. the rewards was attained at least once before the end of the run. Best in **bold**.

4 Conclusion and future work

In this work, we studied how to obtain more diverse functions using a modified loss function. We compared the technique with existing methods providing neural networks with uncertainty, and illustrated the importance of sampling diverse functions on both low and high-dimensional problems.

We illustrated the method with a model-based approach to study its effects on exploration in reinforcement learning. It could also be applied to model-free methods by enforcing heterogeneity in the successive policies. However, several gaps remain to be bridged: sampling repulsive points is a hard task, and so is the computation of a meaningful similarity loss in high dimension. Finding an adapted representation of the input space could help reduce the dimension of the problem. Still, we believe that being able to sample diverse functions is an interesting prospect; future research includes continuing experiments on high dimensional tasks and working on more principled ways of sampling repulsive points.

References

[1] 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*, ICML'15, pages 1613–1622. JMLR.org, 2015.

[2] B. Efron. *The Jackknife, the bootstrap and other resampling plans*. Number 38 in Regional Conference Series in applied mathematics. Society for Industrial and applied mathematics, Philadelphia, Pa., 1982.

[3] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1050–1059, 2016.

[4] A. Graves. Practical variational inference for neural networks. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2348–2356. Curran Associates, Inc., 2011.

[5] D. Hafner, D. Tran, A. Irpan, T. Lillicrap, and J. Davidson. Reliable uncertainty estimates in deep neural networks using noise contrastive priors. In *Submitted to International Conference on Learning Representations*, 2019. under review.

[6] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6402–6413. Curran Associates, Inc., 2017.

[7] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4026–4034, 2016.

[8] I. Osband, D. Russo, and B. V. Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3003–3011, 2013.

[9] T. Pearce, N. Anastassacos, M. Zaki, and A. Neely. Bayesian inference with anchored ensembles of neural networks, and application to reinforcement learning. In *Accepted to 35th International Conference on Machine Learning Workshop on Exploration in Reinforcement Learning*, 2018.

[10] J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[11] A. Touati, H. Satija, J. Romoff, J. Pineau, and P. Vincent. Randomized value functions via multiplicative normalizing flows. In *Accepted to 35th International Conference on Machine Learning Workshop on Exploration in Reinforcement Learning*, 2018.

The experiments were coded in Python and used the PyTorch deep learning library.

A Method for the 1D experiment

We use a 1D bimodal function as a ground truth function. It was designed as part of a one dimensional experiment aimed at evaluating exploration in a RL setting.

```
def reward_1d(state, action, next_state):
    y_left = st.skewnorm(a=4, loc=.3, scale=.7).pdf(3 * next_state) / 1.6
    y_right = st.skewnorm(a=4, loc=.3, scale=.6).pdf(3 * (1 - next_state)) / 1.4
    return y_left + y_right
```

We scale this function so that the input is defined on $[-0.5, 0.5]$ and the output on $[-1, 1]$. We sample once 10 points uniformly between -0.5 and 0, which serve as the training set $\mathcal{D}$ for all the approximators.

For each implicit uncertainty method, we display a sample function that was chosen completely arbitrarily as the first sample network trained.

Dropout. We train 20 neural networks with 2 hidden layers of 64 neurons per layer using a dropout rate of 0.2, the Adam optimizer, and a learning rate of 0.001. We do not use any weight decay. The weights and biases of the neural networks are drawn uniformly following the default Pytorch weight initialization for linear layers[1]. In order to get a stochastic prediction from the trained neural network using the dropout strategy at test time and output a smooth function, we sample a mask from Bernoulli matrices B^i, one for each layer W^i. We then use Hadamard product between the weight matrix and the Bernoulli matrix $B^i \odot W^i$ to obtain a consistent dropout mask for all inputs. We denote this modified layer a FixedDropout layer. The parameter common to all Bernoulli variables is the same dropout rate of 0.2. We train the 20 neural networks for 10000 epochs. Finally, we compute their empirical mean and standard deviation to display the empirical posterior predictive distribution. We also make the same experiment for a dropout rate of 0.5, which slightly changes the shape of the posterior predictive distribution but not our conclusions (see Figure 3).

Input bootstrapping. For each neural network, we bootstrap a new dataset $\mathcal{D}_j$ from the initial dataset $\mathcal{D}$. We then train a neural network with the same structure and training parameters as for dropout, but with a dropout rate of 0.1 at training time (and no dropout at test time).

Anchoring. We follow the instructions given in [9]. The neural network structure is the same as the previous experiments.

Our method. We create repulsive points before training the neural network by choosing random training examples and adding a white Gaussian noise to them: $x = x_i + \epsilon, x_i \in \mathcal{D}, \epsilon \sim \mathcal{N}(0, 0.3)$. Note that the original training points are still part of the training set. We then train a neural network with the same structure as presented before using the repulsive loss. We use the following values for the hyperparameters: $\lambda = 3 \times 10^{-3}$, $n_{\mathbb{X}} = 20$. We sample σ from a log-uniform distribution between 10^{-3} and $10^{-0.5}$, to avoid a cycle between the function and the reference function. The hyperparameters were tuned at a minimum to avoid incorporating bias.

[1]We tried other weight initialization strategies but it didn't affect the nature of the resulting uncertainty.

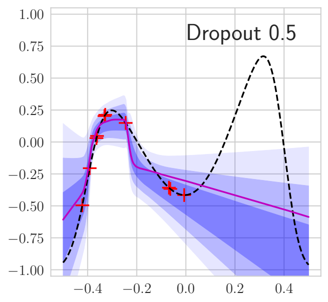

Figure 3: Empirical posterior predictive distribution for a dropout rate of 0.5.

Additional training time for our proposed approach. We give an example of the additional training time of our proposed approach for a simple task. We randomly generate a dataset of 100 points in the range $[-0.5, 0.0]$ and their image through the reward function `reward_1d` defined above. We first train a neural network with a regular MSE loss and the same setting as above. We then train a neural network using our proposed approach, using $n_{\mathbb{X}} = 100$ repulsive points. We compare the training time required to train the networks for 500 epochs. On a 2015 Macbook Pro with a 2,7 GHz Intel Core i5 processor, training a regular neural network takes $\sim$ 85s compared to $\sim$ 100s for a neural network trained with a repulsive loss (results averaged over 50 repetitions).

B Additional insights on the repulsive hyperparameters

Trade-off between losses. The parameter λ represents a trade-off between the error loss and the similarity loss. When sampling points $x \in \mathbb{X}$, it balances between being close to the observations and far from the reference function.

Kernel bandwidth and its sampling distribution. We provide additional insight on the role of the kernel bandwidth σ. This parameter let us control the *authorized* similarity between the function being trained f and the reference function g. To obtain more diverse f, we do not use a fixed σ but rather sample if from a distribution. We train 500 functions with different random seeds on a dataset comprised of 10 points chosen randomly in $[-0.5, 0.0]$. We compare sampling σ from a log-uniform distribution (Figure 4, left) and from an uniform distribution (Figure 4, right). For each choice of distribution of sigma, we draw an histogram of the trained functions predictions at location $x = 1.0$ over the range $[-1.0, 2.0]$. We do not include the outputs outside this range for readability reasons.

The bounds of both distributions are $[10^{-3}, 10^{-0.5}]$. To observe solely the influence of the distribution of σ, we only use one fixed repulsive point $x = 0.55$.

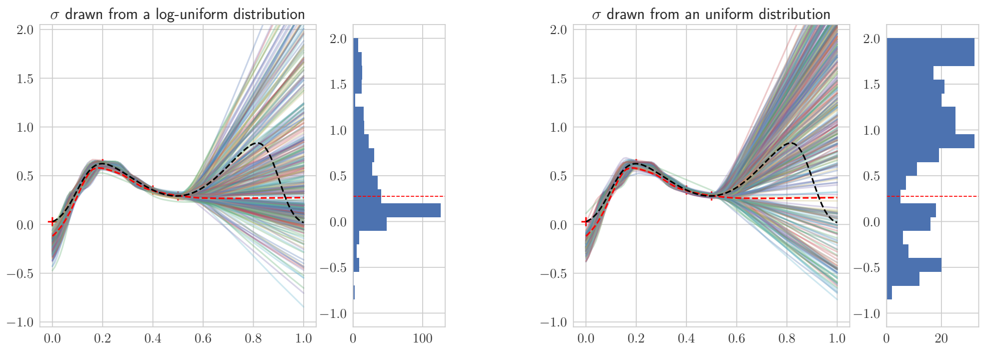

Figure 4: Trained functions and histogram of their predictions at location $x = 1.0$ for σ sampled from log-uniform distribution (left) and uniform distribution (right). The reference function g is plotted in dashed red.

We observe that sampling σ from a uniform distribution tends to apply a higher repulsion, as expected. As a consequence, most of the predictions are outside the $[-1.0, 2.0]$ range. On the contrary, the log-uniform distribution offers a more gradual distribution of the predictions. Optimizing the bounds of the log-uniform distribution should enable even better distributed, diverse functions.

C 4 rooms experiment

C.1 CNN architecture

Our small convolutional neural network structure is the following:

1. Convolutional layer: 3x3 kernel, stride 1, 3 channels,
2. (for dropout only) FixedDropout, with a rate of 0.2,
3. ReLU activation,
4. Maxpooling layer, stride 2,
5. Convolutional layer: 3x3 kernel, stride 1, 6 channels,
6. (for dropout only) FixedDropout, with a rate of 0.2,
7. ReLU activation,
8. Maxpooling layer, stride 2,
9. Fully connected layer, mapping the $3 \times 3 \times 6$ output of the previous layer to a single output (the reward).

The CNN is trained with an Adam optimizer with a default learning rate of $1e^{-3}$ for 200 epochs, as this number was shown to be sufficient to observe convergence of the loss. We do not use weight decay during training, and dropout only for the dropout method. We adapt the FixedDropout layer from the fully connected case to smoothen the output function.

We center and standardize the frames before feeding them to the network.

The empirical posterior predictive distributions are computed over 20 sample functions.

C.2 Example of repulsive points

We also experiment with sampling the repulsive points from the set of frames that have the same color distribution as the frames from the training set, without success. Example of such repulsive points are displayed in Figure 6.

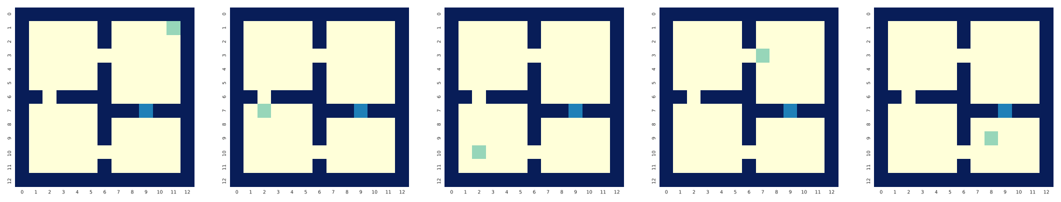

Figure 5: Repulsive points used for training the reward function in Figure 2.

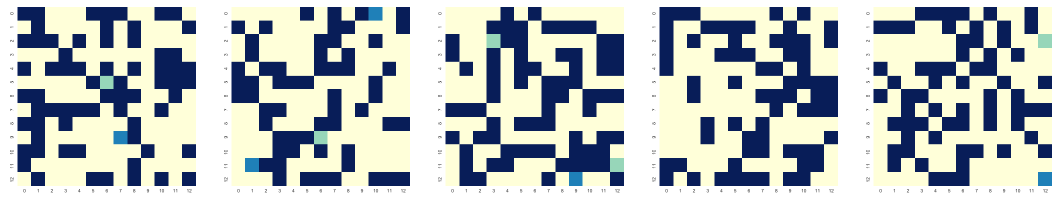

Figure 6: Repulsive points where each pixel color is sampled from the distribution of colors in the training set.

C.3 Empirical posterior predictive distribution and illustration for simple training, input bootstrapping and dropout

We give here some details on how we obtained the frames displaying the evaluation of the sample reward function and the mean plus standard deviation image.

To generate the sample reward, we feed the trained CNN one frame for each legal position of the agent in the state space (*i.e.* for each pixel) and get the corresponding reward per position. The sample reward displayed on Figure 2 is then obtained by coloring each pixel using the value of the predicted reward associated to this position. Repeating this process 50 times allows us compute the mean reward and the standard deviation per position.

We clamp the values of the rewards displayed in Figures 2 between 0 and 1 to facilitate comparison. Attaining a value of 1 already illustrates the potential for exploration, as the unique reward observed by the agent is the reward of 0.5 displayed in Figure 2.

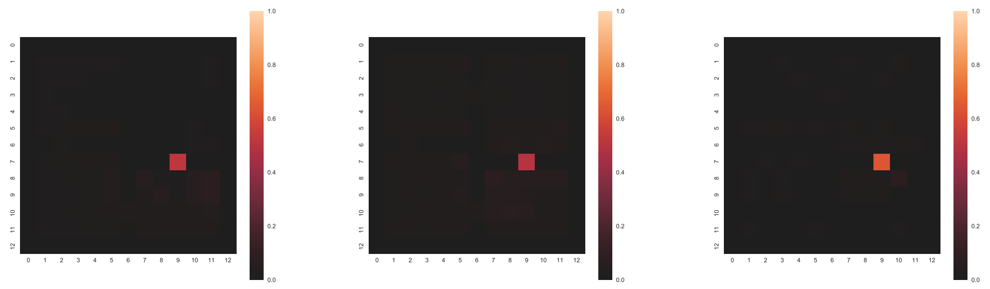

Figure 7: Mean plus one standard deviation of the empirical distribution computer over 50 sample reward functions, by simply training neural networks (left), input bootstrapping (middle) and dropout with a 0.2 rate (right).

For our proposed approach, we used the following values for the hyperparameters: $\lambda = 2$, $\sigma = 0.5$, $n_{\mathbb{X}} = 5$.

C.4 Application to exploration in reinforcement learning

We adapt the Posterior Sampling for Reinforcement Learning algorithm [8] to our framework, by replacing the MDP sampling phase of the posterior by an approximation of the reward and transition function with neural networks trained with a repulsive loss. Algorithm 1 summarizes the different steps. Similarly to Thompson sampling, actions are taken greedily with respect to the sampled function.

Algorithm 1 Diverse Model Sampling for Exploration

```
Input: Repulsive constraint parameters $\lambda, \sigma, n_{\mathbb{X}}$
Exploration policy $\pi_0^X \sim$ Uniform
$\mathcal{D} = \{\}$
for n ∈ N episodes do
    $\mathcal{D} = \mathcal{D} \cup \left\{(s_{n,i}, a_{n,i}, r_{n,i}, {s'}_{n,i})_{i \in [1;K]}\right\} \sim \pi_n^X$    ▷ K tuples $(s, a, r, s')$ from true model $\mathcal{M}$
    Sample a reward function $R$ using pairs $(s', r)$ ▷ ≈ sampling from the post. pred. distribution
    Sample a transition function $T$ using pairs $((s, a), s')$
    Define approximate model $\hat{\mathcal{M}} = (R, T)$
    Find optimal policy $\hat{\pi}$ for approximate model $\hat{\mathcal{M}}$
    Define $\pi_n^X = \hat{\pi}$    ▷ Update the current policy $\pi_n^X$
end for
Output: Policy $\pi_N^X$
```

We use the following values for the hyperparameters: $\lambda = 3 \times 10^{-3}$, $n_{\mathbb{X}} = 2$. For this experiment, the repulsive points in $\mathbb{X}$ are sampled uniformly in $[0, 1]$ and σ is sampled from a log-uniform distribution in $[10^{-3}, 10^{-0.5}]$. We use a discount factor $\gamma = 0.99$.

Figure 8 shows the reward function for three consecutive episodes. The policy that will be used for generating the data points in the next episode is indicated by colors in the background of each plot: strong red indicates a move of $3m$ while strong blue refers to a move of $-3m$. Lighter colors represent smaller moves. Yellow indicates that the policy is to stay at the same location. We observe that the optimistic draw of a reward function (middle) leads the agent to visit previously unobserved states. This illustrates the behaviour of algorithm 1 and the advantages of sampling diverse functions.

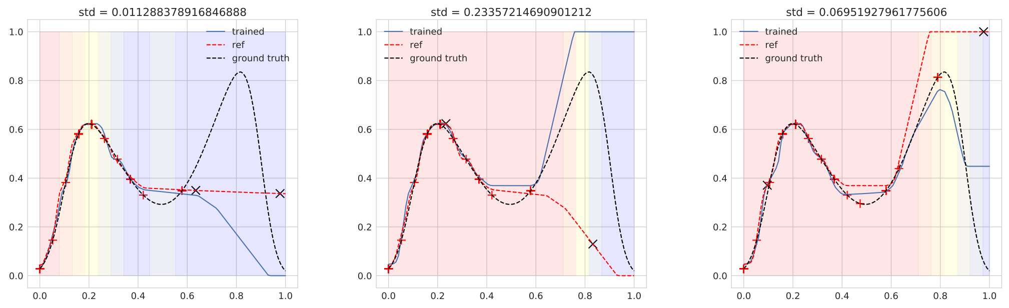

Figure 8: Left: the agent has only explored the suboptimal region. Middle: an optimistic reward function leads to a policy favoring exploration. Right: the agent has explored the optimal region of the state space. Data points from the training set (thus visited by the agent) are represented by red crosses and repulsive points by black crosses. The value of σ sampled from the log-uniform distribution at each episode is indicated on the top of each figure.