Randomized Value Functions via Multiplicative Normalizing Flows

Ahmed Touati[1, 3] **Harsh Satija**[2, 3] **Joshua Romoff**[2, 3] **Joelle Pineau** [2, 3] **Pascal Vincent**[1, 3]
[1] Mila, Université de Montréal [2]Mila, McGill University [3]Facebook AI Research

Abstract

Randomized value functions offer a promising approach towards the challenge of efficient exploration in complex environments with high dimensional state and action spaces. Unlike traditional point estimate methods, randomized value functions maintain a posterior distribution over action-space values. This prevents the agent's behavior policy from prematurely exploiting early estimates and falling into local optima. In this work, we leverage recent advances in variational Bayesian neural networks and combine these with traditional Deep Q-Networks (DQN) and Deep Deterministic Policy Gradient (DDPG) to achieve randomized value functions for high-dimensional domains. In particular, we augment DQN and DDPG with multiplicative normalizing flows in order to track a rich approximate posterior distribution over the parameters of the value function. This allows the agent to perform approximate Thompson sampling in a computationally efficient manner via stochastic gradient methods. We demonstrate the benefits of our approach through an empirical comparison in high dimensional environments.

Efficient exploration is one of the main obstacles in scaling up modern deep reinforcement learning (RL) algorithms (Bellemare et al., 2016; Osband et al., 2017; Fortunato et al., 2017). The main challenge in efficient exploration is the balance between exploiting current estimates, and gaining information about poorly understood states and actions. Despite the wealth of research into provably efficient exploration strategies, most of these focus on tabular representations and are typically intractable in high dimensional environments (Strehl and Littman, 2005; Kearns and Singh, 2002; Brafman and Tennenholtz, 2002). Currently, the most widely used technique, in Deep RL, involves perturbing the greedy action with some local random noise, e.g ϵ-greedy or Bolzmann exploration (Sutton and Barto, 1998). This naive perturbation is not directed; it continuously explores actions that are known to be sub-optimal and may result in sample complexity that grows exponentially with the number of states (Kearns and Singh, 2002; Osband et al., 2017).

Optimism in the face of uncertainty is one of the traditional guiding principles that offers provably efficient learning algorithms (Strehl and Littman, 2005; Kearns and Singh, 2002; Brafman and Tennenholtz, 2002; Jaksch et al., 2010). These algorithms incentivize learning about the environment by rewarding the discoveries of poorly understood states and actions with an exploration bonus. In these approaches, the agent first builds a confidence set over Markov Decision Processes (MDPs) that contains the true MDP with high probability. Then, the agent determines the most optimistic and statistically plausible version of its model and acts optimally with respect to it. Inspired by this principle, several Deep RL works prescribe guided exploration strategies, such as pseudo-counts (Bellemare et al., 2016), variational information maximization (Houthooft et al., 2016) and model prediction errors (Stadie et al., 2015). All of the aforementioned methods add an intrinsic reward to the original reward and then simply train traditional Deep RL algorithms on the augmented MDP.

An entire body of algorithms for efficient exploration is inspired by *Thompson sampling* (Thompson, 1933). *Bayesian dynamic programming* was first introduced in Strens (2000) and has become known more recently as *posterior sampling for reinforcement learning* (PSRL) (Osband et al., 2013). In PSRL, the agent starts with a prior belief over world models and then proceeds to update its full posterior distribution over models with the newly observed samples. A model hypothesis is then sampled from this distribution, and a greedy policy with respect to the sampled model is followed thereafter. Un-

fortunately, due to their high computational cost, these methods are only feasible on small MDPs and are of limited practical use in high dimensional environments.

Osband et al. (2017) developed *randomized value functions* in order to improve the scalability of PSRL. At an abstract level, randomized value functions can be interpreted as a model-free version of PSRL. Instead of maintaining a posterior belief over possible models, the agent's belief is expressed over value functions. Similarly to PSRL, a value function is sampled at the start of each episode and actions are selected greedily thereafter. Subsequently, actions with highly uncertain values are explored due to the variance in the sampled value functions. In order to scale this approach to large MDPs with linear function approximation, Osband et al. (2016b) introduce randomized least-square value iteration (RLSVI) which involves using Bayesian linear regression for learning the value function.

In the present work, we are interested in using randomized value functions with deep neural networks as a function approximator. To address the issues with computational and statistical efficiency, we leverage recent advances in variational Bayesian neural networks. Specifically, we use normalizing multiplicative flows (MNF) (Louizos and Welling, 2017) in order to account for the uncertainty of estimates for efficient exploration. MNF is a recently introduced family of approximate posteriors for Bayesian neural networks that allows for arbitrary dependencies between neural network parameters.

Our main contribution is to introduce MNFs into standard value-based Deep RL algorithms, yielding a well-principled and powerful method for efficient exploration. We validate this approach experimentally by comparing against recent Deep RL baselines on several challenging exploration domains, including the Arcade Learning Environment (ALE) (Bellemare et al., 2013). We thus show that the richness of the approximate posterior in MNF enables more efficient exploration in deep reinforcement learning.

1 Reinforcement Learning Background

In reinforcement learning, an agent interacts with its environment which is modelled as a discounted Markov Decision Process $(\mathcal{S}, \mathcal{A}, \gamma, P, r)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, discount factor $\gamma \in [0, 1)$, transition probabilities $P : \mathcal{S} \times \mathcal{A} \to (\mathcal{S} \to [0, 1])$ mapping state-action pairs to distributions over next states, and reward function $r : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$ (Sutton and Barto, 1998). We denote by $\pi(a \mid s)$ the probability of choosing an action a in the state s under the policy $\pi : \mathcal{S} \to (\mathcal{A} \to [0, 1])$. The action-value function for policy π, denoted $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, represents the expected sum of discounted rewards along the trajectories induced by the MDP and π: $Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid (s_0, a_0) = (s, a), \pi\right]$. The expectation is over the distribution of admissible trajectories $(s_0, a_0, s_1, a_1, \ldots)$ obtained by executing the policy π starting from $s_0 = s$ and $a_0 = a$. The action-value function of the optimal policy is $Q^\star(s, a) = \arg\max_\pi Q^\pi(s, a)$ and it satisfies the Bellman optimality equation:

$$Q^\star(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \max_{a \in \mathcal{A}} Q^\star(s', a) \tag{1}$$

Fitted Q iteration (FQI) (Gordon, 1999; Riedmiller, 2005) assumes that the entire learning dataset of agent interactions is available from the start. If $\mathcal{D}$ represents the dataset consisting of (s, a, r, s'), and w represents the weights of the function approximator, then the problem can be formulated as a supervised learning regression problem by minimizing the following:

$$L(w) = \frac{1}{|\mathcal{D}|} \sum_{(s,a,r,s') \in \mathcal{D}} \left[\left(y - Q(s, a; w)\right)^2 \right], \tag{2}$$

where $y = r + \gamma \max_{a \in \mathcal{A}} Q(s', a; w)$.

Deep Q-Networks (DQN) (Mnih et al., 2015) incorporate a deep neural network, parameterized by w, as a function approximator for the action-value function of the optimal policy. The neural network parameters are estimated by minimizing the squared temporal difference residual:

$$L(w) = \mathbb{E}_{\mathcal{D}} \left[\left(y - Q(s, a; w)\right)^2 \right], \tag{3}$$

where $y = r + \gamma \max_{a \in \mathcal{A}} Q(s', a; w^-)$ and $\mathbb{E}_{\mathcal{D}}$ denotes the expectation over transitions $(s, a, r = r(s, a), s' \sim p(s' \mid s, a))$ sampled uniformly from a replay buffer $\mathcal{D}$ of recent observed transitions. Here w^- denotes the parameters of a target network which is updated $(w^- \leftarrow w)$ regularly and held fixed between individual updates of w. The action-value function defines the policy implicitly by $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$.

Double DQN (DDQN) (van Hasselt et al., 2016) improves DQN by removing its over-estimation bias by simply replacing y in equation 3 with the following:

$$y = r + \gamma Q(s', a; w^-), \quad a = \arg\max_{a' \in \mathcal{A}} Q(s', a'; w).$$

Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015) is an off-policy actor-critic (Sutton et al., 2000) algorithm that uses the deterministic policy gradient to operate over continuous action spaces. Similar to

DQN, the critic estimates the value function by minimizing the same loss as in equation 3 but by replacing the target y with the following:

$$y = r + \gamma Q(s', \pi(s'; \psi); w^{-}), \tag{4}$$

where $\pi(.; \psi)$ is the parameterized actor or policy. The actor is trained to maximize the critic's estimated value function by back-propagating through both networks. The exploration policy in DDPG is attained by adding noise, $\pi(s; \psi) + \epsilon$, where $\epsilon \sim OU(0, \sigma^2)$ and OU denotes the Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930).

2 Variational inference for Bayesian neural networks

In order to explore more efficiently, our approach captures the uncertainty of the value estimates by using Bayesian inference. Instead of maintaining a point estimate of the deep Q-network parameters, we infer a posterior distribution. However, due to the nonlinear aspect of neural networks, obtaining the posterior distributions is not tractable and approximations have to be introduced. Thus, in this work, we use the variational inference procedure (Hinton and van Camp, 1993) and the so-called reparametrization trick for neural networks (Kingma and Welling, 2013; Rezende et al., 2014).

Variational Inference. Let $\mathcal{D}$ be a dataset consisting of input output pairs $\{(x_1, y_1), \dots (x_n, y_n)\}$. A neural network parameterized by weights w models the conditional probability $p(y \mid x, w)$ of an output y given an input x. Let $p(w)$ and $q_\phi(w)$ be respectively the prior and approximate posterior over weights w. Variational Inference (VI) consists of maximizing the following Evidence Lower Bound (ELBO) with respect to the variational posterior parameters ϕ:

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(w)} \left[\log p(y \mid x, w)\right] - \mathbb{KL}(q_\phi(w) || p(w)). \tag{5}$$

where $\mathbb{E}_{q_\phi(w)}$ denotes expectation over parameters w sampled from $q_\phi(w)$. We note that the ELBO is a lower bound on the marginal log-likelihood of the dataset $\mathcal{D}$.

Mean Field Approximation. Blundell et al. (2015) assumes a mean field with independent Gaussian distributions for each weight: Let $w \in \mathbb{R}^{D_{in} \times D_{out}}$ be the weight matrix of a fully connected layer, $q_\phi(w) = \prod_{i=1}^{D_{in}} \prod_{j=1}^{D_{out}} q_\phi(w_{i,j})$ and $q_\phi(w_{i,j}) = \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$ where $\phi = (\mu, \sigma)$ are learned parameters. The uni-modal and the fully factorized Gaussian are both limiting assumptions for high dimensional weights. They are not flexible enough to capture the true posterior distribution which is much more complex. Thus, the accuracy of the model's uncertainty estimates are potentially compromised.

Multiplicative Normalizing Flows (MNF). Louizos and Welling (2017) use multiplicative noise to define a more expressive approximate posterior. Multiplicative noise is often used as stochastic regularization in training a deterministic neural network, such as Gaussian Dropout (Srivastava et al., 2014). The technique was later reinterpreted as an algorithm for approximate inference in Bayesian neural networks (Kingma et al., 2015). The approximate posterior is as follows:

$$z \sim q_\phi(z); \quad w \sim q_\phi(w \mid z) = \prod_{i=1}^{D_{in}} \prod_{j=1}^{D_{out}} \mathcal{N}(z_i \mu_{i,j}, \sigma^2_{i,j}). \tag{6}$$

The approximate posterior is considered an infinite mixture $q(w) = \int q(w \mid z) q(z) dz$, where $z \in \mathbb{R}^{D_{in}}$ plays the role of an auxiliary latent variable (Salimans et al. (2015); Ranganath et al. (2016)). The vector z is of much lower dimension (D_{in}) than w ($D_{in} \times D_{out}$). To make the posterior approximation richer and allow arbitrarily complex dependencies between the components of the weight matrix, the mixing density $q(z)$ is modeled via normalizing flows (Rezende and Mohamed, 2015). This comes at an additional computational cost that scales linearly in the number of parameters.

Normalizing flows is a class of invertible deterministic transformations for which the determinant of the Jacobian can be computed efficiently. A rich density function $q(z_K)$ can be obtained by applying a invertible transformation f_k on an initial random variable z_0, K times, successively. Consider a simple distribution, factorized Gaussian, $q(z_0) = \prod_{j=1}^{D_{in}} \mathcal{N}(\mu_{z_{0,j}}, \sigma^2_{z_{0,j}})$, the computation is then as follows:

$$z_K = \text{NF}(z_0) = f_K \circ \dots \circ f_1(z_0); \tag{7}$$

$$\log q(z_K) = \log q(z_0) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial z_{k-1}} \right|. \tag{8}$$

In *multiplicative normalizing flows* (MNF) (Louizos and Welling, 2017), z_k acts multiplicatively on the mean to the weights w as shown in equation 6. We denote ϕ as the learnable posterior parameters which are composed of $\mu_{z_0}, \sigma_{z_0}, \mu_w, \sigma_w$ and Normalizing flow (NF) parameters.

Unfortunately, the $\mathbb{KL}$ divergence term in the ELBO defined in equation 20 becomes generally intractable as the posterior $q(w)$ is an infinite mixture. This is addressed by introducing an auxiliary posterior distribution $r_\theta(z \mid w)$ parameterized by θ and using it to further lower bound the $\mathbb{KL}$ divergence term of equation 20. Formally, $r_\theta(z \mid w)$

is parameterized by inverse normalizing flows as follows:

$$z_0 \sim r(z_0 \mid w) = \prod_{i=1}^{D_{in}} \mathcal{N}(\tilde{\mu}_i(w), \tilde{\sigma}_i(w)), \quad (9)$$

where $\tilde{\mu}_i(w)$ and $\tilde{\sigma}_i(w)$ can be functions of w

$$z_0 = \text{NF}^{-1}(z_K) = g_1^{-1} \circ \ldots \circ g_K^{-1}(z_K); \quad (10)$$

$$\log r(z_k \mid w) = \log r(z_0 \mid w) + \sum_{k=1}^{K} \log \left| \det \frac{\partial g_k^{-1}}{\partial z_k} \right|, \quad (11)$$

and where we parameterize $g_1^{-1}, \ldots, g_K^{-1}$ as another normalizing flow. The parameters θ are the learnable auxiliary network parameters which are composed of the parameters of $\tilde{\mu}$ and $\tilde{\sigma}$ and the parameters of the inverse normalizing flows NF^{-1}. Finally, we obtain the lower bound that MNF should optimize by replacing the $\mathbb{KL}$ divergence term with the lower bound in terms of the distribution r_θ:

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{q_\phi(w, z_K)}[\log p(y \mid x, w) - \underbrace{(\mathbb{KL}(q_\phi(w \mid z_K) || p(w)) - \log r_\theta(z_K | w) + \log q_\phi(z_K))}_{\text{regularization\_cost}(w, z_k, \phi, \theta)}]. \quad (12)$$

We can now parameterize the random sampling from $q_\phi(w, z_K)$ in terms of noise variables ϵ_w and ϵ_z, and deterministic function h by $w, z_K = h(\phi, \epsilon_w, \epsilon_z)$ as described by the following sampling procedure:

$$\begin{aligned} \epsilon_z &\sim q(\epsilon_z) = \mathcal{N}(0, \mathbf{I}_{D_{in}}), \\ z_0 &= \mu_{z_0} + \sigma_{z_0} \odot \epsilon_z, \\ z_K &= \text{NF}(z_0), \\ \epsilon_w &\sim q(\epsilon_w) = \mathcal{N}(0, \mathbf{I}_{D_{in} \times D_{out}}), \\ w &= (z_K \mathbf{1}_{D_{out}}^\top) \odot \mu_w + \sigma_w \odot \epsilon_w, \end{aligned} \quad (13)$$

where $\odot$ is elementwise multiplication, $\mathbf{I}_{D_{in}}$ and $\mathbf{I}_{D_{in} \times D_{out}}$ are identity matrices and $\mathbf{1}_{D_{out}}$ is a D_{out} dimensional vector whose entries are all equal to one. The lower bound $\mathcal{L}(\phi, \theta)$ in equation 12 can be written as:

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{\epsilon_z, \epsilon_w}[\log p(y \mid x, h(\phi, \epsilon_w, \epsilon_z)) - \text{regularization\_cost}(h(\phi, \epsilon_w, \epsilon_z), \theta)] \quad (14)$$

Thus, we can have a Monte Carlo sample of the gradient of $\mathcal{L}(\phi, \theta)$ with respect to ϕ and θ. This parameterization allows us to handle the approximate parameter posterior as a straightforward optimization problem.

Choice of normalizing flows. In practice, we use masked RealNVP (Dinh et al., 2016) for the normalizing flows. In particular, we use the numerically stable updates described in Inverse Autoregressive Flow (Kingma et al., 2016):

$$\begin{aligned} m = \text{Bern}(0.5), \quad h = \tanh(a(m \odot z_k)), \\ \mu = b(h), \quad \sigma = \text{sigmoid}(c(h)), \end{aligned}$$

$$\begin{aligned} z_{k+1} &= f_k(z_k) \\ &= m \odot z_k + \\ &\quad (1 - m) \odot (z_k \odot \sigma + (1 - \sigma) \odot \mu), \end{aligned} \quad (15)$$

$$\log \left| \frac{\partial f_k}{\partial z_k} \right| = (1 - m)^\top \log \sigma, \quad (16)$$

where $a(.), b(.)$ and $c(.)$ are linear transformations. For the auxiliary posterior distribution $r(z_0 \mid w)$ defined in equation 9, we parameterize the mean and the standard deviation $\tilde{\mu}$ $\tilde{\sigma}$ as in the original paper (Louizos and Welling, 2017).

3 Multiplicative Normalizing Flows for Randomized Value Functions

We now turn to our novel proposed approach that incorporates the techniques we previously introduced to modify both DQN and DDPG in a similar fashion. In particular, we introduce a new parametrization of the value functions (optimal value function in DQN and the critic in DDPG) and change their corresponding losses.

We model the distribution of target return y (y is equal to $r + \gamma \max_{a \in \mathcal{A}} Q(s', a; w^-)$ for DQN and to $r + \gamma Q(s', \pi(s'; \psi); w^-)$ for DDPG) as a Gaussian distribution with parameterized mean $Q(s, a; w)$ [1] and constant standard deviation τ: $y \sim p(y \mid s, a) = \mathcal{N}(Q(s, a; w), \tau^2)$.

In this setting, minimizing the loss of DQN or of the DDPG critic corresponds to a maximum log-likelihood estimation. Indeed $L(w)$ from equation 2 is such that:

$$\begin{aligned} |\mathcal{D}| L(w) &= \sum_{(s,a,r,s') \in \mathcal{D}} \left[(y - Q(s, a; w))^2 \right] \\ &= -2\tau^2 \sum_{(s,a,r,s') \in \mathcal{D}} \log p(y \mid s, a), \end{aligned} \quad (17)$$

where we ignore constant terms. Instead of computing a maximum likelihood estimate of the parameters w of the value function network, we use a randomized value function to track an *approximate posterior distribution* over network parameters w, using the MNF family to

[1] We overload our notation w for both the weight matrix of a single layer and the full set of network parameters.

parameterize this posterior. The weights w are considered random variables and are obtained by the sampling procedure described in equation 13. The value function $Q(s, a, \epsilon_z, \epsilon_w; \phi, \theta)$ is now parameterized in terms of the approximate posterior (ϕ, θ) defined in Section 2. Our approach optimizes the ELBO in equation 12 with respect to the approximate posterior parameters (ϕ, θ), which amounts to minimizing the following loss:

$$\mathbb{E}_{\epsilon_z,\epsilon_w}[\sum_{(s,a,r,s')\in\mathcal{D}} (y - Q(s, a, \epsilon_z, \epsilon_w; \phi, \theta))^2 + 2\tau^2 \text{regularization\_cost}(\epsilon_z, \epsilon_w; \phi, \theta)], \quad (18)$$

where $y = r + \gamma \max_a Q(s, a, 0, 0; \phi^-, \theta^-)$ for DQN, $y = \gamma Q(s, \pi(s'; \psi), 0, 0; \phi^-, \theta^-)$ for DDPG and where the noise is disabled for the target network. This loss is amenable to mini-batch optimization. In a supervised learning setting, for each mini-batch $\mathcal{M} \subset \mathcal{D}$, we would take a gradient step to lower the following loss:

$$\mathbb{E}_{\epsilon_z,\epsilon_w}[\frac{1}{|\mathcal{M}|}\sum_{(s,a,r,s')\in\mathcal{M}} (y - Q(s, a, \epsilon_z, \epsilon_w; \phi, \theta))^2 + \lambda \text{ regularization\_cost}(\epsilon_z, \epsilon_w; \phi, \theta)], \quad (19)$$

where $\lambda = \frac{2\tau^2}{|\mathcal{D}|}$. This makes the regularization cost uniformly distributed among mini-batches at each epoch. In the RL setting, however, we only keep a moving window of experiences in the replay buffer. Thus, the size of replay buffer $|\mathcal{D}|$ is not directly analogous to the size of the dataset. So we leave λ as a hyper-parameter to tune.

The expectation in equation 19 is with respect to the distribution of noise variables ϵ_z and ϵ_w of the online value function. An unbiased estimate of the loss is thus obtained by simply sampling the two noise variables. The current noise samples are held fixed across the mini-batch. The learnable parameters (ϕ, θ) are then updated by performing a gradient descent on the mini-batch loss. In the case of DDPG, in addition to the critic parameters, we update the actor policy parameters ψ using the following sampled policy gradient:

$$\frac{1}{|\mathcal{M}|}\sum_{s\in\mathcal{M}} \nabla_a Q(s, a, \epsilon_z, \epsilon_w; \phi, \theta)|_{a=\pi(s;\psi)} \nabla_\psi \pi(s; \psi)$$

After the update, we generate new noise samples and we select actions greedily with respect to the corresponding value function.

We call the new adaptations of DQN and DDPG, MNF-DQN and MNF-DDPG, respectively. Detailed algorithms are provided as Algorithm 1 and Algorithm 2 (MNF-DDPG could be find in appendix).

Algorithm 1 MNF-DQN

Input: m mini-batch size; D empty replay buffer; K update frequency, U target update frequency
for episode $e \in 1, ..., M$ **do**
  Set $s \leftarrow s_0$
  Sample noise variables ϵ_w and ϵ_z from standard normal distribution.
  for $t \in 1, \ldots$ **do**
    Select an action $a_t \leftarrow \arg\max_{a\in A} Q(s, a, \epsilon_z, \epsilon_w; \phi, \theta)$
    Observe next state s_{t+1} and reward r_t after taking action a_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer D
    if $t \mod K == 0$ **then**
      Sample a mini-batch of m transitions $((s_j, a_j, r_j, s'_j) \sim D)_{j=1}^m$
      for $j = 1, \ldots, m$ **do**
        if s'_j is terminal state **then** $y_j = r_j$
        else
          $y_j = r_j + \gamma Q(s, a, 0, 0; \phi^-, \theta^-)$
        end if
      end for
      Re-sample noise variables ϵ_w and ϵ_z and perform gradient step with respect to (ϕ, θ):
      $\frac{1}{m}\sum_{j=1,...m} (y_j - Q(s_j, a_j, \epsilon_z, \epsilon_w; \phi, \theta))^2 + \lambda \text{regularization\_cost}(\epsilon_z, \epsilon_w; \phi, \theta)$
      Re-sample noise variables ϵ_w and ϵ_z
    end if
    if $t \mod U == 0$ **then**
      Set $\theta^-, \phi^- \leftarrow \theta, \phi$
    end if
    Set $s \leftarrow s_{t+1}$
  end for
end for

4 Related work

There have been several recent works that incorporate Bayesian parameter updates with deep reinforcement learning for efficient exploration.

Osband et al. (2016a) propose BootstrappedDQN which consists of a simple non-parametric bootstrap with random initialization to approximate a distribution over value functions. BootstrappedDQN consists of a network with multiple Q-heads. At the start of each episode, the agent samples a head, which it follows for the duration of the episode. BootstrappedDQN is a non-parametric approach to uncertainty estimation. In contrast, MNF-DQN uses a parametric approach, based on variational inference, to quantify the uncertainty estimates.

Azizzadenesheli et al. (2018) extend randomized least-square value iteration by Osband et al. (2016b) (which was restricted to linear approximator) to deep neural networks. In particular, they consider only the last layer as stochastic and keep the remaining layers deterministic. As the last layer is linear, they propose a Bayesian linear regression to update the posterior distribution of its weights in closed form. In contrast, our method is capable of performing an approximate Bayesian update on the full network parameters. Variational inference could be applied for all layers using stochastic gradient descent on the approximate posterior parameters.

The closest work to ours is BBQ-Networks by Lipton et al. (2016). Their algorithm, called Bayes-by-Backprop Q-Network (BBQN), uses variational inference to quantify uncertainty. It uses independent factorized Gaussians as an approximate posterior (Blundell et al., 2015). In our work, we argue that to achieve efficient exploration, we need to capture the true uncertainty of the value function. The latter depends importantly on the flexibility of the approximate posterior distribution. We also note that BBQN was proposed for Task-Oriented Dialogue Systems and was not evaluated on standard RL benchmarks. Furthermore, BBQN can be seen simply as a sub-case of MNF-DQN. In fact, the two algorithms are equivalent when we set the auxiliary variable z in MNF-DQN to be equal to one.

Our work is also related to methods that inject noise in the parameter space for exploration. For such methods, at the beginning of each episode, the parameters of the current policy are perturbed with some noise. This results in a policy that is perturbed but still consistent for individual states within an episode. This is sometimes called state-dependent exploration (Sehnke et al., 2010) as the same action will be taken every time the same state is sampled in the episode. Recently, Fortunato et al. (2017) proposed to add parametric noise to the parameters of a neural network and show that its aids exploration. The parameters of the noise are learned with gradient descent along with the remaining network weights. Concurrently, Plappert et al. (2017) proposed a similar approach but they rely on heuristics to adapt the noise scale instead of learning it as in Fortunato et al. (2017).

Osband et al. (2018) recently discuss limitations of some popular approaches for exploration in Deep RL. In particular, approaches such as NoisyDQN, BBQ-Networks and even our proposed method MNF-DQN treat each Bellman error as an independent draw from the posterior. This fact, according to Osband et al. (2018), prevents these methods from propagating uncertainty in the correct way. On the other hand, BootstrapedDQN would be able to propagate a temporally-consistent sample of Q-values, since each head is trained only on its own target value. However, we argue that MNF-DQN is still a principled approach: We can view DQN from the value iteration perspective. DQN considers a target network $\tilde{Q}$ and tries to minimize the mean-squared Bellman error (MSBE) $||Q - \mathcal{T}\tilde{Q}||^2$ where $\mathcal{T}$ is the optimality Bellman operator. For a given target network, DQN tries to solve the latter regression problem by performing multiple stochastic gradient steps. This can be viewed as one-step of value iteration that tries to fit Q to $\mathcal{T}\tilde{Q}$. Maintaining only a point estimate of Q does not capture the prediction uncertainty and leads to decisions that do not reflect our understanding of the environment. Therefore, instead of having a point estimate of Q, we turn the mean-squared Bellman error minimization into bayesian regression and maintain a posterior distribution over Q. This aims at capturing the parametric uncertainty due to limited data and prevents overfitting. When (s, a) are poorly understood, the distribution of $Q(s, a)$ might have large variance. When we draw a sample from this distribution, there is a chance that we attribute a high value to (s, a) due to the variance which in turn enables exploration. As long as we collect more data, the variance should decrease, which enable exploitation.

5 Experiments

We evaluate the performance of MNF-DQN on two toy domains (N-chain and Mountain Car), as well as on several Atari games (Bellemare et al., 2013). Moreover, we compare MNF-DDPG on continuous control tasks from OpenAI Gym (Brockman et al., 2016). We compare the performance of our agents to several recent state-of-the-art deep exploration methods.

5.1 Discrete Action Environments

5.1.1 Toy Tasks

n-Chain As a sanity check, we evaluate MNF-DQN on the well-known n-chain environment introduced in Osband et al. (2016a). The environment consists of N states. The agent always starts at the second state s_2 and has two possible actions: move right and move left. A small reward $r = 0.001$ is received in the first state s_1, a large reward $r = 1$ in the final state s_N, otherwise the reward is zero.

We compare the exploration behavior of MNF-DQN, NoisyDQN (Fortunato et al., 2017), BBQN (Lipton et al., 2016) and ϵ-greedy DQN on varying chain lengths. We train each agent with ten different random seeds for each chain length. After each episode, agents are evaluated on a single roll-out with all of their randomness disabled (ϵ is set to zero for DQN, noise variables are set to zero for

MNF-DQN, BBQN, NoisyDQN). The problem is considered solved when the agent completes the task optimally for one hundred consecutive episodes. While the task is admittedly a simple one, it still requires adequate exploration in order to be solved. This is especially true with large chain lengths, as it is easy to discover the small reward and fall into premature exploitation. Figure 1 shows that MNF-DQN has very consistent performance across different chain lengths. MNF-DQN clearly outperforms ϵ-greedy DQN which most of the time fails to solve the problem when $n \geq 10$. BBQN performs well but slightly worse than MNF-DQN for very large chain length. MNF-DQN also outperforms NoisyDQN, which on average needs a larger number of episodes to solve the task.

Mountain Car (Moore, 1990) is a classic RL continuous state task where the agent (car) is initially stuck in a valley and the goal is to drive up the mountain on the right. The only way to succeed is to drive back and forth to build up momentum. We use the implementation provided by OpenAI gym (Brockman et al., 2016), where the agent gets -1.0 reward at every time-step and get $+1.0$ reward when it reaches up the mountain, at which point the episode ends. The maximum length of an episode is set to 1000 time-steps.

We evaluate the performance of the agent every 10 episodes by using no noise over 5 runs with different random seeds. As can be seen in Figure 2, MNF-DQN learns much faster and performs better when compared to the other exploration strategies.

5.1.2 Arcade Learning Environment

Next, we consider a set of Atari games (Bellemare et al., 2013) as a benchmark for high dimensional state-spaces. We compare MNF-DQN to standard DQN with ϵ-greedy exploration, BBQN and NoisyDQN. We use the same network architecture for all agents, i.e three convolutional layers and two fully connected layers. For DQN, we linearly anneal ϵ from 1.0 to 0.1 over the first 1 million time-steps. For NoisyDQN, the fully connected layers are parameterized as noisy layers and use factorized Gaussian noise as explained in Fortunato et al. (2017). For MNF-DQN and BBQ, in order to reduce computational overhead, we choose to consider only the parameters of the fully connected layers as stochastic variables and perform variational inference on them. We consider the parameters of the convolutional layers as deterministic variables and optimize them using maximum log-likelihood. For MNF-DQN, the normalizing flows are of length two for q_ϕ and r_θ, with 50 hidden units for each step of the flow. To have a fair comparison across all algorithms, we fill the replay buffer with actions selected at random for the first 50 thousand time-steps.

We use the standard hyper-parameters of DQN for all agents. MNF-DQN and BBQN have an extra hyper-parameter λ, the trade-off parameter between the log likelihood cost and the regularization cost. To tune λ, we run MNF-DQN and BBQN for $\lambda = \frac{\alpha}{|\mathcal{D}|}$ where $\alpha \in \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$. As explained earlier, $|\mathcal{D}|$ is not good proxy for dataset size but it gives a good value range. We still tune the hyperparameter α. We train each agent for 40 millions frames. We evaluate each agent on the return collected by the exploratory policy during training steps. Each agent is trained for 5 different random seeds. We plot in Figure 3 the median return as well as the interquartile range.

From Figure 3 we see that across all games our approach provides competitive and consistent results. Moreover, the naive epsilon-greedy approach (DDQN) performs significantly worse than the exploration-based methods in most cases. MNF-DQN provides a boost in performance over the baselines in Gravitar, which is considered a *hard exploration* and sparse reward game (Bellemare et al., 2016). BBQN fails completely for this game. In all the other *hard exploration* games (Amidar, Alien, Bank Heist, and Qbert), the difference between MNF-DQN and the best performing baseline is minimal (within the margin of error).

Note that MNF-DQN, as well as the baseline algorithms, fail to solve extremely challenging games such as Montezuma's Revenge. Note also that methods from the literature that achieved some degree of success on Montezuma's Revenge either include prior information about the game (Le et al., 2018; Aytar et al., 2018) or are combined with heuristics on the intrinsic reward (Bellemare et al., 2016). The first category of methods use human demonstrations to drive the exploration, and the latter require maintaining a separate density model over the state-action space.

So far, we considered only parameterized noise baselines. Now, we compare MNF-DQN to Bootstrapped DQN (Osband et al., 2016a). Results are given in Figure 4. MNF-DQN often outperforms Bootstrapped DQN (Out of 13 games, MNF-DQN clearly outperforms Bootstrapped DQN on 6 of them, yields similar performance on 4, and under-performs it only on 3). Moreover, we investigate the randomized prior function introduced by Osband et al. (2018) which consists in adding a randomized un-trainable prior network to each Q-head. We find that the Bootstrapped DQN with randomized prior does not improve performance over Bootstrapped DQN.

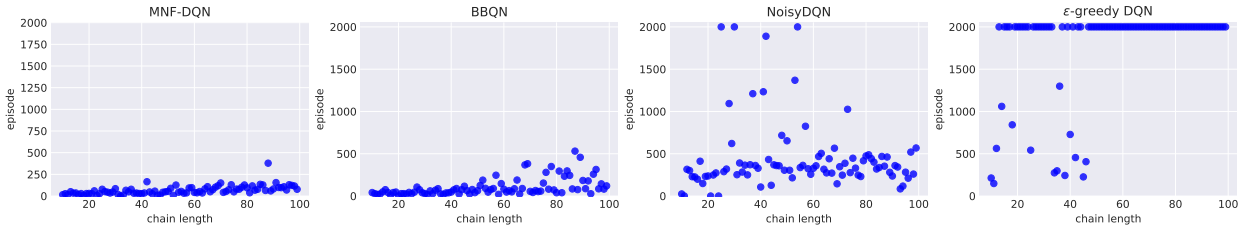

Figure 1: Median number of episodes (max 2000) required to solve the n-chain problem for (figure from left to right) MNF-DQN, BBQN, NoisyDQN and ϵ-greedy DQN. The median is obtained over 10 runs with different seeds. We see that MNF-DQN consistently performs best across different chain lengths.

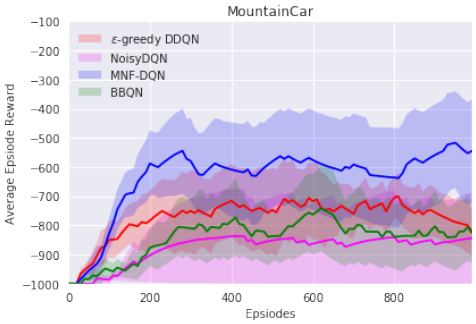

Figure 2: Average return per episode over 5 runs in MountainCar.

5.2 Continuous Action Environments

In this section, we benchmark MNF-DDPG on the continuous control tasks from OpenAI Gym (Brockman et al., 2016). We compare our algorithm to the standard DDPG with OU-noise with $\sigma = 0.2$ (OU-noise DDPG), DDPG with BBQN based critic (BBQN-DDPG) and with Noisy network based critic (Noisy-DDPG). The actor is deterministic in all the agents and only the critic has stochasticity. We use the same setup as Plappert et al. (2017), where both actor and critic use 2 hidden layers with 64 units and a ReLU non-linearity. More details about the architecture can be found in Appendix C.

For MNF-DDPG, the normalizing flows are of length one for q_ϕ and r_θ, with 16 hidden units for each step of the flow. We tune the hyperparameter λ in manner similar to what we described in Sec 5.1.2, trying out $\lambda = \frac{\alpha}{|\mathcal{D}|}$ where $\alpha \in \{1e^{-6}, 4e^{-6}, 8e^{-6}, 1e^{-5}, \ldots, 8e^{-2}, 1e^{-1}, 4e^{-1}, 8e^{-1}\}$. We evaluate the algorithms on 3 different control tasks, where each agent is trained for 1 million time-steps. Each epoch consists of 10,000 time-steps and after every epoch the agent is evaluated with all their randomness disabled for 20 episodes. Each agent is trained for 5 different random seeds and the results are averaged.

Results are shown Figure 5. The MNF-DDPG agent performs better than all the other baselines in the HalfCheetah environment. In the other environments there is not much difference between the different exploration methods. This finding is consistent with the results from Plappert et al. (2017), where they also conclude that the other environments do not require a lot of exploration due to their well-structured reward function.

6 Conclusion

Through the combination of multiplicative normalizing flows and modern value-based deep reinforcement learning methods, we show that a powerful approximate posterior can be efficiently utilized for better exploration. Moreover, the improved sample efficiency comes only at a computational cost that is linear in the number of model parameters. Finally, we find that on several common Deep RL benchmarks, the MNF approximation outperforms state-of-the-art exploration baselines.

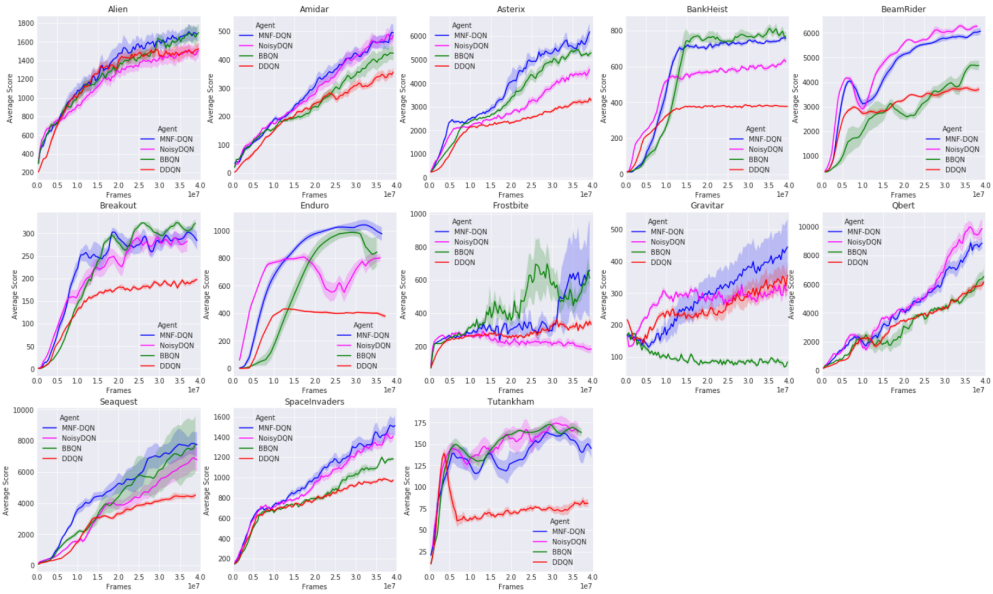

Figure 3: Comparison of the training curves of MNF-DQN agent versus the NoisyDQN, BBQN and DQN + ϵ-greedy baselines for various Atari games.

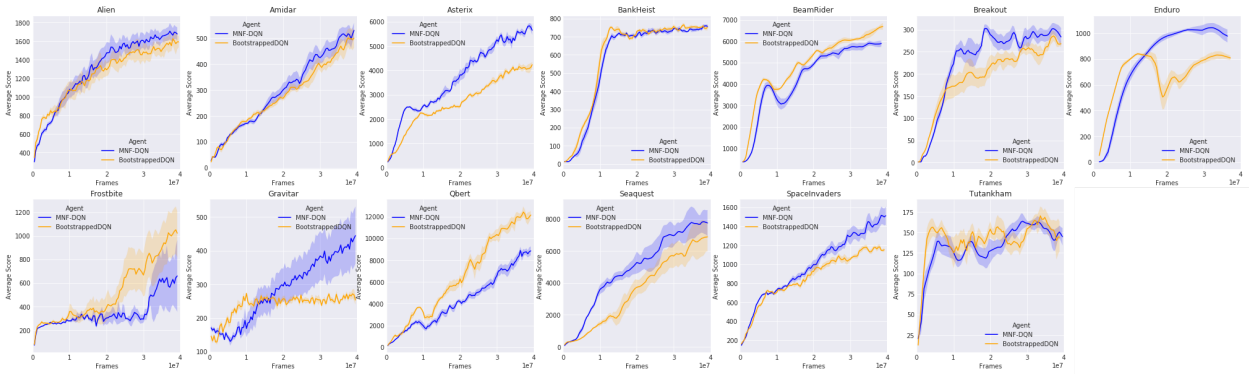

Figure 4: Comparison of the training curves of MNF-DQN agent versus the Bootstrapped DQN for various Atari games.

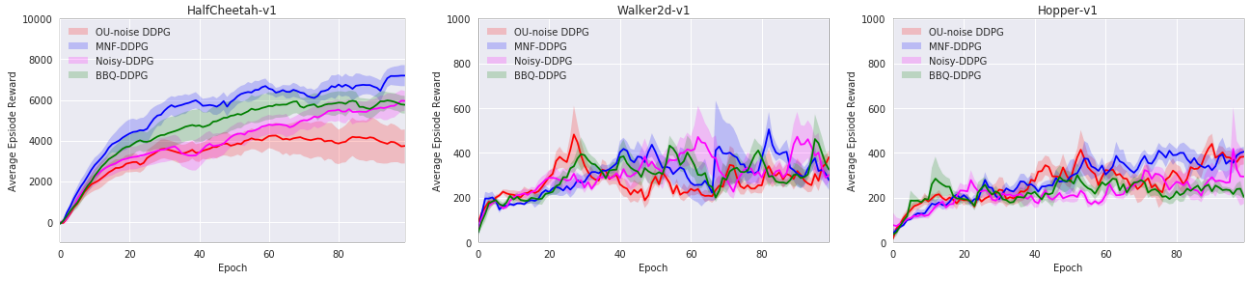

Figure 5: Average return per episode for continuous control environments plotted over epochs over 5 runs.

References

Aytar, Y., Pfaff, T., Budden, D., Paine, T. L., Wang, Z., and de Freitas, N. (2018). Playing hard exploration games by watching youtube. *arXiv preprint arXiv:1805.11592*.

Azizzadenesheli, K., Brunskill, E., and Anandkumar, A. (2018). Efficient exploration through bayesian deep q-networks. *arXiv preprint arXiv:1802.04412*.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622.

Brafman, R. I. and Tennenholtz, M. (2002). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. (2017). Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*.

Gordon, G. J. (1999). Approximate solutions to markov decision processes. *Robotics Institute*, page 228.

Hinton, G. and van Camp, D. (1993). Keeping neural networks simple by minimising the description length of weights. In *Proceedings of COLT-93*.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*.

Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine learning*.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Le, H. M., Jiang, N., Agarwal, A., Dudík, M., Yue, Y., and Daumé III, H. (2018). Hierarchical imitation and reinforcement learning. *arXiv preprint arXiv:1803.00590*.

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

Lipton, Z. C., Gao, J., Li, L., Li, X., Ahmed, F., and Deng, L. (2016). Efficient exploration for dialogue policy learning with bbq networks & replay buffer spiking. *arXiv preprint arXiv:1608.05081*.

Louizos, C. and Welling, M. (2017). Multiplicative normalizing flows for variational bayesian neural networks. In *International Conference on Machine Learning*, pages 2218–2227.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Moore, A. W. (1990). Efficient memory-based learning for robot control.

Osband, I., Aslanides, J., and Cassirer, A. (2018). Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8626–8638.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016a). Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*.

Osband, I., Russo, D., and Van Roy, B. (2013). (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*.

Osband, I., Russo, D., Wen, Z., and Van Roy, B. (2017). Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*.

Osband, I., Van Roy, B., and Wen, Z. (2016b). Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2017). Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

Ranganath, R., Tran, D., and Blei, D. (2016). Hierarchical variational models. In *International Conference on Machine Learning*, pages 324–333.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International Conference on Machine Learning*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286.

Riedmiller, M. (2005). Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer.

Salimans, T., Kingma, D., and Welling, M. (2015). Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226.

Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., and Schmidhuber, J. (2010). Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

Strehl, A. L. and Littman, M. L. (2005). A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd international conference on Machine learning*. ACM.

Strens, M. (2000). A bayesian framework for reinforcement learning. In *ICML*.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*.

Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the theory of the brownian motion. *Physical review*, 36(5):823.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. *AAAI Conference on Artificial Intelligence*.

A Derivation of the lower bound

We have the following ELBO and we would like to lower bound the KL term using the auxiliary posterior distribution $r_\theta(z \mid \omega)$

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi(w)}\left[\log p(y \mid x, w)\right] - \mathbb{KL}(q_\phi(w)||p(w)). \tag{20}$$

As the KL divergence is always non-negative, we have:

$$\begin{aligned}
\mathbb{KL}(q_\phi(w)||p(w)) &\leq \mathbb{KL}(q_\phi(w)||p(w)) + \mathbb{E}_{q_\phi(\omega)}\left[\mathbb{KL}(q_\phi(z|w)||r_\theta(z|w))\right] && (21)\\
&= \mathbb{E}_{q_\phi(\omega)}\left[\log q_\phi(\omega) - \log p(\omega)\right] + \mathbb{E}_{q_\phi(\omega,z)}\left[\log q_\phi(z \mid \omega) - \log r_\theta(z \mid \omega)\right] && (22)\\
&= \mathbb{E}_{q_\phi(\omega,z)}\left[\log(q_\phi(\omega)q_\phi(z \mid \omega)) - \log p(\omega) - \log r_\theta(z \mid \omega)\right] && (23)\\
&= \mathbb{E}_{q_\phi(\omega,z)}\left[\log(q_\phi(\omega \mid z)q_\phi(\omega)) - \log p(\omega) - \log r_\theta(z \mid \omega)\right] && (24)\\
&= \mathbb{E}_{q_\phi(\omega,z)}\left[\mathbb{KL}(q_\phi(\omega \mid z)||p(\omega)) + \log q_\phi(z) - \log r_\theta(z \mid \omega\right] && (25)
\end{aligned}$$

which proves the lower bound in equation 12

B Algorithms

Algorithm 2 MNF-DDPG algorithm

Input: m mini-batch size; D empty replay buffer; K update frequency, U target update frequency
Initialize critic network $Q(s, a, \epsilon_z, \epsilon_w; \phi, \theta)$ and actor $\pi(s; \psi)$ with weights θ, ϕ and ψ.
Initialize target critic network $Q(s, a, \epsilon_z, \epsilon_w; \phi^{-1}, \theta^-)$ and actor $\pi(s; \psi^-)$ with weights $\theta^- \leftarrow \theta, \phi^{-1} \leftarrow \phi, \psi^- \leftarrow \psi$
for episode $e \in 1, ..., M$ **do**
  Set $s \leftarrow s_0$
  Sample noise variables ϵ_w and ϵ_z from standard normal distribution.
  for $t \in 1, \ldots$ **do**
    Select an action $a_t \leftarrow \pi(s; \psi)$
    Observe next state s_{t+1} and reward r_t after taking action a_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer D
    if $t \mod K == 0$ **then**
      Sample a mini-batch of m transitions $((s_j, a_j, r_j, s'_j) \sim D)_{j=1}^m$
      for $j = 1, \ldots, m$ **do**
        if s'_j is terminal state **then** $y_j = r_j$
        else
          $y_j = r_j + \gamma Q(s, \pi(s; \psi^-), 0, 0; \phi^{-1}\theta^-)$
        end if
      end for
      Re-sample noise variables ϵ_w and ϵ_z and perform gradient step with respect to (ϕ, θ):
      $\frac{1}{m}\sum_{j=1,\ldots m}\left(y_j - Q(s_j, a_j, \epsilon_z, \epsilon_w; \phi, \theta)\right)^2 + \lambda$regularization_cost$(\epsilon_z, \epsilon_w; \phi, \theta)$
      Update the actor using sampled policy gradient: $\frac{1}{m}\sum_{j=1,\ldots m}\nabla_a Q(s_j, a, \epsilon_z, \epsilon_w; \phi, \theta)|_{a=\pi(s_j;\psi)}\nabla_\psi \pi(s_j; \psi)$
      Re-sample noise variables ϵ_w and ϵ_z
    end if
    if $t \mod U == 0$ **then**
      Set $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$
      Set $\phi^- \leftarrow \tau\phi + (1-\tau)\phi^-$
      Set $\psi^- \leftarrow \tau\psi + (1-\tau)\psi^-$
    end if
    Set $s \leftarrow s_{t+1}$
  end for
end for

C Continuous Control Architecture Details

The actor and critic use 2 hidden layers each with 64 ReLU units. The action is added to the second hidden layer of the critic. Layer-normalization is applied between the hidden layers before the non-linearity and target-networks are soft updated with $\tau = 0.001$. Critic is trained with learning rate = 10^{-3} and the actor with 10^{-4}. Adam optimizer is used with batch size of 128 to update the weights. Replay buffer has size of $100K$ transitions with $\gamma = 0.99$. Both update frequency (K) and target update frequency (U) are set to 1 (in Alg.2).

D Comparison with Bayesian DQN

We did not implement Bayesian DQN (Azizzadenesheli et al., 2018) but we used the results of 14 Atari games provided by authors in the article's github repository `https://github.com/kazizzad/BDQN-MxNet-Gluon`. Out of these 14 games, seven are in common with our set of games. Among these 7 games, Bayesian DQN outperforms our method only in 3 games. See Figure .

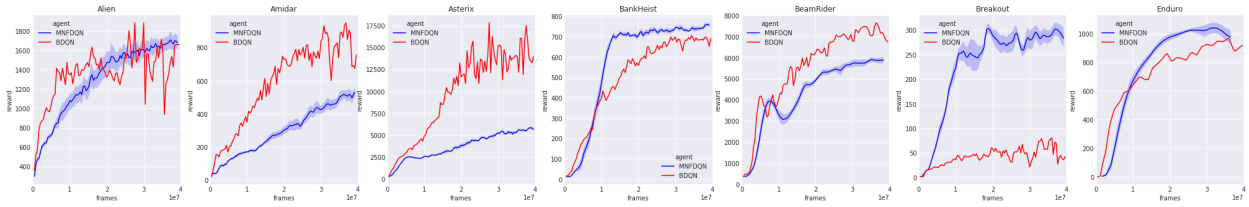

Figure 6: Comparison of the training curves of MNF-DQN agent versus the Bayesian DQN (Azizzadenesheli et al., 2018) for various Atari games.