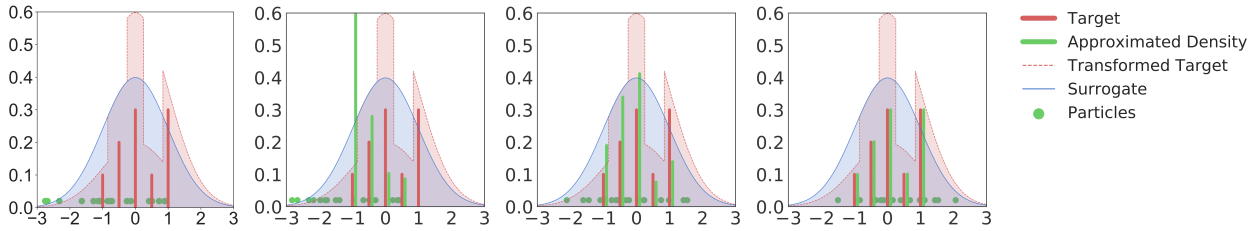
Appendix

A Additional Experimental Result

Result on Categorical Distribution We apply our algorithm to sample from one-dimensional categorical distribution $p_*(z)$ shown in red bars in Fig. 7, defined on $\mathcal{Z} := \{-1, -0.5, 0, 0.5, 1\}$ with corresponding probabilities $\{0.1, 0.2, 0.3, 0.1, 0.3\}$. The blue dash line is the surrogate distribution $\rho(x) = p_0(x)$, where the base function $p_0(x)$ is the p.d.f. of standard Gaussian distribution. The red dash line is the transformed piecewise continuous density $p_c(x) \propto p_0(x)p_*(\Gamma(x))$, where $\Gamma(x) = a_i$ if $x \in [\eta_{i-1}, \eta_i)$ and η_i is $i/5$-th quantile of standard Gaussian distribution. We apply Algorithm 1 to draw a set of samples $\{x_i\}_{i=1}^n$ (shown in green dots) to approximate the transformed target distribution. Then we can obtain a set of samples $\{z_i\}_{i=1}^n$ by $z_i = \Gamma(x_i)$), to get an approximation of the original categorical distribution.

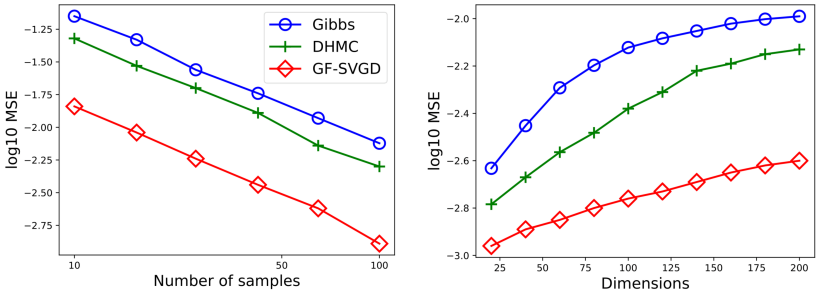


(a) 0th iteration (b) 25th iteration (c) 50th iteration (d) 100th iteration

Figure 7: Evolution of real-valued particles $\{x_i\}_{i=1}^n$ (in green dots) by our discrete sampler in Alg.1 on a one-dimensional categorical distribution. (a-d) shows particles $\{x^i\}$ at iteration 0, 10, 50 and 100 respectively. The categorical distribution is defined on states $z \in \{-1, -0.5, 0, 0.5, 1\}$ denoted by a_1, a_2, a_3, a_4, a_5, with probabilities $\{0.1, 0.2, 0.3, 0.1, 0.3\}$ denoted by c_1, c_2, c_3, c_4, c_5, respectively. $p_*(z = a_i) = c_i$. The base function is $p_0(x)$, shown in blue line. The transformed target to be sampled $p_c(x) \propto p_0(x)p_*(\Gamma(x))$, where $\Gamma(x) = a_i$ if $x \in [\eta_{i-1}, \eta_i)$ and η_i is $i/5$-th quantile of standard Gaussian distribution. The surrogate distribution $\rho(x)$ is chosen as $p_0(x)$. We obtain discrete samples $\{z_i\}_{i=1}^n$ by $z_i = \Gamma(x_i)$.

As shown in Fig 7, the empirical distribution of the discretized sample $\{\boldsymbol{z}_i\}_{i=1}^n$ (shown in green bars) aligns closely with the true distribution (the red bars) when the algorithm converges (e.g., at the 100-th iteration).

Results on Bernoulli RBM The probability model is given in (17) and the score function is derived in Section 5.3 (Han & Liu, 2017). We also evaluate the sample quality based on the mean square error (MSE) between the estimation and the ground truth value. From Fig. 8(a), we can see that when fixing the dimension of the distribution $p_*(\boldsymbol{z})$, our sampling method has much lower MSE than Gibbs and DHMC. In Fig. 8(b), as the dimension of the model increases, our sampling method has relatively better MSE than that of Gibbs and DHMC.



(a) Fix dimension (b) Fix sample size

Figure 8: Bernoulli RBM with number of visible units $M = 25$. In (a), we fix the dimension of visible variables $d = 100$ and vary the number of samples $\{\boldsymbol{z}^j\}_{j=1}^n$. In (b), we fix the number of samples $n = 100$ and vary the dimension of visible variables d. We calculate the MSE for estimating the mean $\mathbb{E}[z]$ (lower is better).

B Training BNN Algorithm

In this section, we provide the procedure of our principled ensemble algorithm to train binarized neural network. We train an ensemble of n neural networks (NN) with the same architecture ($n \geq 2$). Let $\boldsymbol{w}_i^b$ be the binary weight of model i, for $i = 1, \cdots, n$, and $p_*(\boldsymbol{w}_i^b; D)$ be the target probability model with softmax layer as last layer given the data D. Learning the target probability model is framed as drawing n samples $\{\boldsymbol{w}_i^b\}_{i=1}^n$ to approximate the posterior distribution $p_*(\boldsymbol{w}^b; D)$. We apply multi-dimensional transform $\boldsymbol{F}$ to transform the original discrete-valued target to the target distribution of real-valued $\boldsymbol{w} \in \mathbb{R}^d$. Let $p_0(w)$ be the base function, which is the product of the p.d.f. of the standard Gaussian distribution over the dimension d. Based on the derivation in Section 3, the distribution of $\boldsymbol{w}$ has the form $p_c(\boldsymbol{w}; D) \propto p_*(\text{sign}(\boldsymbol{w}); D)p_0(\boldsymbol{w})$ with weight $\boldsymbol{w}$ and the sign function is applied to each dimension of $\boldsymbol{w}$. To backpropagate the gradient to the non-differentiable target, we construct a surrogate probability model $\rho(\boldsymbol{w}; D)$ which approximates $\text{sign}(\boldsymbol{w})$ in the transformed target by $\sigma(\boldsymbol{x})$ and relax the binary activation function $\{-1, 1\}$ by σ, where σ is defined by (**??**), denoted by $\widetilde{p}(\sigma(\boldsymbol{w}); D)p_0(\boldsymbol{w})$. Here $\widetilde{p}(\sigma(\boldsymbol{w}); D)$ is a differentiable approximation of $p_*(\text{sign}(\boldsymbol{w}); D)$. Then we apply GF-SVGD to update $\{\boldsymbol{w}_i\}$ to approximate the transformed target distribution of $p_c(\boldsymbol{w}; D)$ of $\boldsymbol{w}$ as follows, $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i + \frac{\epsilon_i}{\Omega}\Delta\boldsymbol{w}_i$, $\forall i = 1, \cdots, n$,

$$\Delta\boldsymbol{w}_i \leftarrow \sum_{j=1}^{n} \gamma_j [\nabla_{\boldsymbol{w}} \log \rho(\boldsymbol{w}_j; D_i) k(\boldsymbol{w}_j, \boldsymbol{w}_i) + \nabla_{\boldsymbol{w}_j} k(\boldsymbol{w}_j, \boldsymbol{w}_i)] \tag{18}$$

where D_i is batch data i and $\mu_j = \rho(\boldsymbol{w}_j; D_i)/p_c(\boldsymbol{w}_j; D_i)$, $H(t) \stackrel{\text{def}}{=} \sum_{j=1}^n \mathbb{I}(\mu_j \geq t)/n$, $\gamma_j = (H(\boldsymbol{w}_j))^{-1}$ and $\Omega = \sum_{j=1}^n \gamma_j$. Note that we don't need to calculate the cumbersome term $p_0(\boldsymbol{w})$ as it can be canceled from the ratio between the surrogate distribution and the transformed distribution. In practice, we find a more effective way to estimate this density ratio denoted by γ_j. Intuitively, this corresponds to assigning each particle a weight according to the rank of its density ratio in the population. Algorithm 3 on Appendix B can be viewed as a new form of ensemble method for training NN models with discrete parameters.

Algorithm 3 GF-SVGD on training BNN

Inputs: training set D and testing set D_{test}
Outputs: classification accuracy on testing set.
Initialize full-precision models $\{\boldsymbol{w}^i\}_{i=1}^n$ and its binary form $\{\boldsymbol{w}_i^b\}_{i=1}^n$ where $\boldsymbol{w}_i^b = \text{sign}(\boldsymbol{w}^i)$.
while not converge **do**
-Sample n batch data $\{D_i\}_{i=1}^n$.
-Calculate the true likelihood $p_c(\boldsymbol{w}_i; D_i) \propto p_*(\text{sign}(\boldsymbol{w}_i); D_i)p_0(x)$
-Relax $\boldsymbol{w}_b^i$ with $\sigma(\boldsymbol{w}_i)$
-Relax each sign activation function to the smooth function defined in (**??**) to get $\widetilde{p}$
-Calculate the surrogate likelihood $\rho(\boldsymbol{w}^i; D_i) \propto \widetilde{p}(\sigma(\boldsymbol{w}_i); D_i)p_0(\boldsymbol{x})$
-$\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i + \Delta\boldsymbol{w}_i$, $\forall i = 1, \cdots, n$, where $\Delta\boldsymbol{w}_i$ is defined in (18).
-Clip $\{\boldsymbol{w}_i\}$ to interval $(-1, 1)$ for stability.
end while
-Calculate the probability output by softmax layer $p(\boldsymbol{w}_i^b; D_{\text{test}})$
-Calculate the average probability $f(\boldsymbol{w}_b; D_{\text{test}}) \leftarrow \sum_{i=1}^n p(\boldsymbol{w}_i^b; D_{\text{test}})$
Output test accuracy from $f(\boldsymbol{w}_b; D_{\text{test}})$.

C Transform Discrete Samples to Continuous Samples for Goodness-of-fit Test

Let F be the c.d.f. of Gaussian base density p_0. Let us first illustrate how to transform one-dimensional samples $\{z_i\}_{i=1}^n$ to continuous samples.

1. Given discrete data $\{z_i\}_{i=1}^n$. Let $\{a_j\}_{j=1}^K$ are possible discrete states. Assume K is large so that for any z_i, we have $z_i = a_j$ for one j.
2. For any z_i such as $z_i = a_j$, randomly sample $y_i \in [\frac{j-1}{K}, \frac{j}{K})$. We obtain data $\{y_i\}_{i=1}^n$.
3. Apply $x = F^{-1}(y)$, we obtain data $\{x_i\}_{i=1}^n$.

For $\boldsymbol{x} = (x^1, \cdots, x^d)$, let $F(\boldsymbol{x}) = (F_1(x^1), \cdots, F_d(x^d)$, where F_i is the c.d.f. of Gaussian density $p_{0,i}(x^i)$. We apply the above one-dimensional transform to each dimension of $\{\boldsymbol{z}_i\}_{i=1}^n$, $\boldsymbol{z}_i = (z_i^1, \cdots, z_i^d)$. We can easily obtain the continuous data $\{\boldsymbol{x}_i\}_{i=1}^n$.

D Proofs

In the following, we prove proposition 4.

Proposition 4 Assume Γ is an even partition of $p_0(\boldsymbol{x})$, and $p_c(\boldsymbol{x}) = Kp_0(\boldsymbol{x})p_*(\Gamma(\boldsymbol{x}))$, where K severs as a normalization constant, then $(p_c,\ \Gamma)$ is a continuous parameterisation of p_*.

Proof. We just need to verify that (9) holds.

$$\begin{aligned}
&\int p_c(\boldsymbol{x})\mathbb{I}[\boldsymbol{a}_i = \Gamma(\boldsymbol{x})]d\boldsymbol{x} \\
&= K\int p_0(\boldsymbol{x})p_*(\Gamma(\boldsymbol{x}))\mathbb{I}[a_i = \Gamma(\boldsymbol{x})]d\boldsymbol{x} \\
&= K\int p_0(\boldsymbol{x})p_*(\boldsymbol{a}_i)\mathbb{I}[\boldsymbol{a}_i = \Gamma(\boldsymbol{x})]d\boldsymbol{x} \\
&= Kp_*(\boldsymbol{a}_i)\int p_0(\boldsymbol{x})\mathbb{I}[\boldsymbol{a}_i = \Gamma(\boldsymbol{x})]d\boldsymbol{x} \\
&= p_*(\boldsymbol{a}_i),
\end{aligned}$$

where the last step follows (10). □

E Detail of Experiments and Network Architecture

In all experiments, we use RBF kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/h)$ for the updates of our proposed algorithms; the bandwidth h is taken to be h=med$^2/(2\log(n+1))$ where med is the median of the current n particles. Adam optimizer Kingma & Ba (2014) is applied to our proposed algorithms for accelerating convergence. $\epsilon = 0.0001$ works for all the experiments.

We use the same AlexNet as Zhu et al. (2018), which is illustrated in the following.

Layer	Type	Parameters
1	Conv	Depth: 96, K: 11×11, S: 4, P:0
2	Relu	-
3	MaxPool	K: 3×3, S: 2
4	BatchNorm	-
5	Conv	Depth: 256, K: 5×5, S: 1, P:1
6	Relu	-
7	MaxPool	K: 3×3, S: 2
8	BatchNorm	-
9	Conv	Depth: 384, K: 3×3, S: 1, P:1
10	Relu	-
11	Conv	Depth: 384, K: 3×3, S: 1, P:1
12	Relu	-
13	Conv	Depth: 256, K: 3×3, S: 1, P:1
14	Relu	-
15	MaxPool	K: 3×3, S: 2
16	Dropout	$p = 0.5$
17	FC	Width=4096
18	Relu	-
19	Dropout	$p = 0.5$
20	FC	Width=4096
21	Relu	-
22	FC	Width=10

Table 1: Architecture of AlexNet. "K" denotes kernel size; "S" denotes stride; "P" denotes padding.