Variational Energy-Based Models: A Probabilistic Framework for Contrastive Self-Supervised Learning

Tianqi Du[1*] **Yifei Wang**[1*] **Weiran Huang**[2] **Yisen Wang**[3,4†]
[1] School of Mathematical Sciences, Peking University
[2] Huawei Noah's Ark Lab
[3] Key Lab. of Machine Perception (MoE)
School of Intelligence Science and Technology, Peking University
[4] Institute for Artificial Intelligence, Peking University

Abstract

Contrastive self-supervised learning has achieved impressive results in various scenarios. However, there still lacks a probabilistic framework for the principled design of the learning objective. Starting from the energy-based model, we design a variational energy-based model (VEM) for the self-supervised learning setting whose learning objective equals the InfoNCE loss via some slight approximations. Following that, we propose two sampling variants, VEM-Langevin and VEM-SVGD, for crafting better negative samples to help contrastive learning. Extensive experiments show that our VEM variants with SimCLR and MoCo indeed promote the performance on a wide range of inference tasks.

1 Introduction

Recently, contrastive self-supervised learning achieves impressive results in a wide range of scenarios, which follows a general paradigm that pulls the positive samples together while pushing the negative samples apart. In this work, we aim to establish a principled probabilistic framework for contrastive learning, so as to enable diverse statistical tasks with contrastive representations.

In this paper, we advocate for the perspective of energy-based models (EBMs), to establish a probabilistic framework for contrastive learning (CL). We formulate the pretext signals as latent variables v, named as Variational Energy-based Model (VEM). We show the variational inference of VEM is deeply connected to CL and demonstrate their equivalence under mild approximations. In particular, VEM explicitly integrates positive and negative samples and thus provides guidelines for their designs. Thus, we believe VEM is a preferable probabilistic framework for CL.

VEM not only provides a unified probabilistic framework for CL, but also allows us to derive principled new variants of CL by adopting different sampling techniques. As for the former, through VEM, we could integrate the heuristic design of pretext, *e.g.,* data augmentations [1, 7], into a unified framework, where its role is to specify the probabilistic encoder $q(v|x)$. On the other hand, existing works have shown that the negative samples adopted in existing CL methods are inefficient [19, 15] and we can improve the sampling quality by incorporating a fast sampling procedure in the feature space that does not require back-propagation. Extensive experiments show that our sampling-based variants of VEM achieve superior performance than vanilla CL on a wide range of tasks, including image classification, out-of-distribution detection, calibration, and adversarial defense.

[*]Equal contribution.
[†]Corresponding Author: Yisen Wang (yisen.wang@pku.edu.cn).

2 Variational Energy-based Models

In our proposed Variational Energy-based Models (VEMs), we treat the augmented view $\boldsymbol{v} \in \mathbb{R}^n$ as an unobserved *latent* variable and define the joint probability over an input $\boldsymbol{x}$ and a view $\boldsymbol{v}$ as

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}) = \frac{\exp(-E_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}))}{Z(\boldsymbol{\theta})}, \tag{1}$$

with the joint energy function $E_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v})$ and the corresponding normalization constant $Z(\boldsymbol{\theta})$. Inspired by contrastive learning, we specify the energy function as

$$E_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}) = -\log p_d(\boldsymbol{x}) - \log p_d(\boldsymbol{v}) - \operatorname{sim}(f_{\boldsymbol{\theta}}(\boldsymbol{x}), f_{\boldsymbol{\theta}}(\boldsymbol{v}))/\tau, \tag{2}$$

where $f_{\boldsymbol{\theta}} : \mathbb{X} \to \mathbb{R}^m$ denotes an encoder with parameters $\boldsymbol{\theta}$ and $p_d(\boldsymbol{x})$ is the data distribution. We let $F_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}) = \operatorname{sim}(f_{\boldsymbol{\theta}}(\boldsymbol{x}), f_{\boldsymbol{\theta}}(\boldsymbol{v}))/\tau$ and omit τ for brevity of notation in the following. Accordingly, pairs with a large similarity in the feature space will have a lower energy, and thus a higher probability.

2.1 Variational Inference with Augmentation-based Encoders

To learn VEM with latent variables $\boldsymbol{v}$, we propose a novel variational framework that naturally unifies contrastive learning and contrastive divergence.

To begin with, following the conventions of variational inference, we adopt the Evidence Lower Bound (ELBO) as a tractable lower bound of the log-likelihood,

$$\begin{aligned}\mathbb{E}_{p_d(\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) =& \mathbb{E}_{p_d(\boldsymbol{x})} \mathbb{E}_{q_a(\boldsymbol{v}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v})}{q_a(\boldsymbol{v}|\boldsymbol{x})} + D_{\mathrm{KL}}(q_a(\boldsymbol{v}|\boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{v}|\boldsymbol{x})) \\ \geq& \mathbb{E}_{p_d(\boldsymbol{x})} \mathbb{E}_{q_a(\boldsymbol{v}|\boldsymbol{x})} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v})}{q_a(\boldsymbol{v}|\boldsymbol{x})} := \mathrm{ELBO}(\boldsymbol{\theta}).\end{aligned} \tag{3}$$

Here, $q_a(\boldsymbol{v}|\boldsymbol{x})$ denotes a variational posterior for approximating the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{v}|\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v})/p_{\boldsymbol{\theta}}(\boldsymbol{x})$. It can be shown that the ELBO lower bounds the log-likelihood and their gap is the KL divergence between the true and the variational posteriors.

An important design of our VEMs is that we adopt a fixed posterior distribution $q_a(\boldsymbol{v}|\boldsymbol{x})$ instead of adopting an additional variational module as in previous works. We adopt the same manual data augmentation strategies for generating latent views. This makes $q_a(\boldsymbol{v}|\boldsymbol{x})$ a uniform distribution over the support $\mathcal{V}(\boldsymbol{x})$, which is the set of all possible augmented views of $\boldsymbol{v}$ with the augmentation operator set $\mathcal{T} = \{t : \mathbb{R}^n \to \mathbb{R}^n\}$[3], *i.e.,* $q_a(\boldsymbol{v}|\boldsymbol{x}) = 1/|\mathcal{T}|$ if $\boldsymbol{v} \in \mathcal{V}(\boldsymbol{x})$ and $q_a(\boldsymbol{v}|\boldsymbol{x}) = 0$ if $\boldsymbol{v} \notin \mathcal{V}(\boldsymbol{x})$. Applying this variational posterior $q_a(\boldsymbol{v}|\boldsymbol{x})$ to the ELBO objective, we can obtain an equivalent objective by omitting the constants,

$$\mathrm{ELBO}(\boldsymbol{\theta}) = \mathbb{E}_{p_d(\boldsymbol{x}) q_a(\boldsymbol{v}|\boldsymbol{x})} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}), \tag{4}$$

which resembles the log-likelihood of VEMs by regarding $q_a(\boldsymbol{x}, \boldsymbol{v}) := p_d(\boldsymbol{x}) q_a(\boldsymbol{v}|\boldsymbol{x})$ as the *pseudo joint data distribution*. In the next part, we will discuss how to approximate its gradient with contrastive divergence.

2.2 Contrastive Divergence Training

To optimize the ELBO objective (Eq. 4), we apply Contrastive Divergence to approximate the joint log-likelihood and obtain the following objective,

$$D_{\mathrm{KL}}(p_d(\boldsymbol{x}) q_a(\boldsymbol{v}|\boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v})) - D_{\mathrm{KL}}(p_{\boldsymbol{\theta}}^{(t)}(\boldsymbol{x}, \boldsymbol{v})) \| p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v})), \tag{5}$$

with the following gradient

$$-\mathbb{E}_{q_a(\boldsymbol{x}, \boldsymbol{v})} \frac{\partial}{\partial \boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}) + \mathbb{E}_{p_{\boldsymbol{\theta}}^{(t)}(\boldsymbol{x}, \boldsymbol{v})} \frac{\partial}{\partial \boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}), \tag{6}$$

where $p_{\boldsymbol{\theta}}^{(t)}(\boldsymbol{x}, \boldsymbol{v})$ denotes the t-step Langevin samples starting from $q_a(\boldsymbol{x}, \boldsymbol{v})$.

[3]For simplicity, we assume these operations are one-to-one mapping.

Proposition 2.1. *Recall that $F_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}) = \operatorname{sim}(f_{\boldsymbol{\theta}}(\boldsymbol{x}), f_{\boldsymbol{\theta}}(\boldsymbol{v}))$. The gradient of contrastive divergence is equivalent to the following contrastive learning objective*

$$- \mathop{\mathbb{E}}_{\substack{\boldsymbol{x} \sim p_d(\boldsymbol{x}) \\ \boldsymbol{v} \sim q_a(\boldsymbol{v}|\boldsymbol{x})}} F_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v}) + \mathop{\mathbb{E}}_{\operatorname{sg}(\boldsymbol{x}) \sim p_{\boldsymbol{\theta}}^{(t)}(\boldsymbol{x})} \log \mathop{\mathbb{E}}_{\boldsymbol{v} \sim p_d(\boldsymbol{v})} \exp(F_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{v})), \tag{7}$$

where $\operatorname{sg}(\boldsymbol{x})$ means $\boldsymbol{x}$ is a leaf node when calculating the gradient.

Proposition 2.2. *By approximate $p_{\boldsymbol{\theta}}^{(t)}(\boldsymbol{x})$ with $p_d(\boldsymbol{x})$, InfoNCE is a special form of the loss in Eq.7.*

Proposition 2.2 implies that our model is compatible with the existing contrastive learning method. Thus, our model can be utilized to design better learning objectives and negative sample mining methods, which will be elaborated on in the next section.

3 Crafting Better Negative Samples from the Model

3.1 Negative Samples in Contrastive Learning

According to [21], in order to get good representation, features should be roughly uniformly distributed on the unit hypersphere preserving as much information of the data as possible. Point $\boldsymbol{x}$ with higher $p_{\boldsymbol{\theta}}$ will have larger average similarity with other points in the dataset, which indicates that dataset points around $\boldsymbol{x}$ are denser. By pushing the anchor away from points with high $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, the feature of the anchor can be pushed away from dense areas. The whole representation features thus become more separated. However, the probability of points in the support of $p_d(\boldsymbol{x})$ are all the same. Pushing the anchor away from points sampled from $p_d(\boldsymbol{x})$ contributes less to the uniformity of features. Therefore, we propose to get negative samples from $p_{\boldsymbol{\theta}}(\boldsymbol{x})$.

In order to get samples $\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_K$ from $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, we can simply perform Langevin dynamics with $\hat{\boldsymbol{x}}^0$ as the initialization:

$$\hat{\boldsymbol{x}}^{t+1} = \hat{\boldsymbol{x}}^t + \alpha \nabla_{\hat{\boldsymbol{x}}^t} \log \left(\sum_{i=1}^{n} F(\hat{\boldsymbol{x}}^t, \boldsymbol{v}_i) \right) + \sqrt{2\alpha} \cdot \varepsilon, \tag{8}$$

where $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ are samples from $p_d(\boldsymbol{v})$, α is the step size and ε is the intensity of the noise. The last step in Eq. 8 uses Monte Carlo Estimator to estimate the integral over $p_d(\boldsymbol{v})$..

To cover more low density area of $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, we can perform Stein Variational Gradient Descent (SVGD) [13] which has a stronger repulsive term. We perform SVGD starting with a group of particles $\{\hat{\boldsymbol{x}}_i^{(0)}\}_{i=1}^K$. For each step, we update the particles with

$$\hat{\boldsymbol{x}}_i^{t+1} = \hat{\boldsymbol{x}}_i^t + \frac{\alpha}{K} \sum_{j=1}^{K} \left(k(\hat{\boldsymbol{x}}_i^t, \hat{\boldsymbol{x}}_j^t) \nabla_{\hat{\boldsymbol{x}}_j^t} \log \left(\sum_{i=1}^{n} F(\hat{\boldsymbol{x}}_t, \boldsymbol{v}_i) \right) + \nabla_{\hat{\boldsymbol{x}}_j^t} k(\hat{\boldsymbol{x}}_i^t, \hat{\boldsymbol{x}}_j^t) \right), \tag{9}$$

where α is the step size and $k(x, y)$ is an RBF kernel function. Compared to Langevin sampling, SVGD considers the interaction between samples. Hence, SVGD can get samples with higher quality within a few steps. However, both two sampling methods are time-consuming with multi-steps and we need an implementation with low time and memory cost, as it will be shown in the next section.

3.2 Practical Sampling Strategy

To get samples from $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, we can either perform Langevin dynamics with Eq.8 or SVGD with Eq.9. However, performing one step of sampling need K forward propagation steps and K backward propagation steps, which is very time-consuming. It also needs large memory to store the temp values $\hat{\boldsymbol{x}}_i^t$. To overcome the difficulties, we can directly get samples in the feature space. Recall that our energy function is $E_{\boldsymbol{\theta}(\boldsymbol{x},\boldsymbol{v})} = -\log p_d(\boldsymbol{x}) - \log p_d(\boldsymbol{v}) - \operatorname{sim}(f_{\boldsymbol{\theta}}(\boldsymbol{x}), f_{\boldsymbol{\theta}}(\boldsymbol{v}))$. We can redefine the input of the model from $\boldsymbol{x}$ to $\boldsymbol{z} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ through re-parameterization. Therefore, the energy function becomes:

$$E'_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{v}) = -\log p'_d(\boldsymbol{z}) - \log p_d(\boldsymbol{v}) + \operatorname{sim}(\boldsymbol{z}, f_{\boldsymbol{\theta}}(\boldsymbol{v})) \tag{10}$$

For performing Langevin Dynamics, we initialize $\boldsymbol{z}^0$ in the feature space and update it with:

$$\boldsymbol{z}^{t+1} = \boldsymbol{z}^t + \alpha \nabla_{\boldsymbol{z}^t} \log \left(\sum_{i=1}^{n} \operatorname{sim}(\boldsymbol{z}^t, f_{\boldsymbol{\theta}}(\boldsymbol{v}_i)) \right) + \sqrt{2\alpha} \cdot \varepsilon \tag{11}$$

Table 1: Classification Accuracies (%) on CIFAR10, CIFAR100 and Tiny-ImageNet-200 with ResNet50. All experiments are repeated for 3 times.

	CIFAR10	CIFAR100		Tiny-ImageNet-200	
Method	Test Acc	Test Acc1	Test Acc5	Test Acc1	Test Acc5
MoCo	$92.56_{\pm 0.36}$	$69.75_{\pm 0.27}$	$91.13_{\pm 0.20}$	$53.25_{\pm 0.30}$	$77.18_{\pm 0.44}$
SimCLR	$92.65_{\pm 0.35}$	$69.94_{\pm 0.36}$	$91.35_{\pm 0.19}$	$56.28_{\pm 0.43}$	$79.09_{\pm 0.34}$
VEM-Langevin	$\mathbf{93.26}_{\pm 0.23}$	$72.59_{\pm 0.10}$	$92.85_{\pm 0.10}$	$60.17_{\pm 0.16}$	$82.18_{\pm 0.26}$
VEM-SVGD	$93.23_{\pm 0.28}$	$\mathbf{72.79}_{\pm 0.16}$	$\mathbf{93.37}_{\pm 0.27}$	$\mathbf{60.28}_{\pm 0.25}$	$\mathbf{82.46}_{\pm 0.24}$

For this update way, we do not need any forward or backward propagation in each step. Since the size of a feature (commonly 128 to 512) is usually smaller than the size of an image ($3 \times 32 \times 32$ for CIFAR10), we need less memory to save the temp values. It is similar to perform SVGD sampling. For better convergence rate, we can create a memory bank. We update the memory bank with the samples $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K$ in each training iteration. The samples in the memory bank can be used to initialize the Langevin or SVGD sampling in the next training iteration. We name our proposed training methods as VEM-Langevin and VEM-SVGD, respectively. Their pseudocodes are summarized in Appendix C.

4 Experiments

4.1 Experimental Setup

We carry out experiments on a range of real-world image datasets, including well known benchmarks like CIFAR10 [12], CIFAR100 [12], and one ImageNet variant: Tiny-ImageNet-200 (200 classes with image size resized to 64×64) [23]. We adopt ResNet18 [8], ResNet50 [8] and PyramidNet [6] as the backbones for unsupervised pretraining. The other settings will be described in Appendix D. Results on the linear evaluation task with ResNet50 are shown below and the other experiments will be exhibited in Appendix E.

4.2 Performance on the Classification Task

We test models on the corresponding test set after they are finetuned. We compared our proposed VEM-Langevin ad VEM-SVGD variants with other methods in terms of top-1 accuracy. For datasets with numerous classes such as CIFAR100 and Tiny-ImageNet-200, we also adopt top-5 accuracy. Table 1 reports the experiment results with ResNet50 on various datasets.

The results show that the proposed VEMs achieve the best performance among the compared models in most of the settings. The two variants of VEM both greatly outperform MoCo and SimCLR for more than 2% in terms of top-1 accuracy on CIFAR100 using ResNet50 as the backbone. Our method also has a computing time on par with the contrastive models as shown in Appendix G. It is not surprising since the computing overhead for updating the memory bank is negligible compared with that for updating the backbone network. We also find VEM-SVGD performs slightly better than VEM-Langevin. This is due to the higher sampling efficiency of SVGD.

5 Conclusion

In this paper, we proposed variational energy-based model (VEM), a probabilistic framework for contrastive learning. The variational inference of VEM is deeply connected to contrastive learning and their equivalence can be established under mild approximations. Based on the deficiency of sampling negatives from the data distribution, we designed two negative sampling methods in the feature space which does not need forward and backward propagation. Empirically, our training methods achieve superior performance than vanilla contrastive learning on a wide range of inference tasks, indicating the efficacy of our proposed probabilistic framework VEM. Overall, we believe that VEM is a preferable probabilistic framework for contrastive learning and could inspire more studies to design better learning objectives and sampling strategies under the framework.

Acknowledgment

Yisen Wang is partially supported by the NSF China (No. 62006153), Project 2020BD006 supported by PKU-Baidu Fund, and Huawei Technologies Inc.

References

[1] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[2] Bi'an Du, Xiang Gao, Wei Hu, and Xin Li. Self-contrastive learning with hard negative sampling for self-supervised point cloud learning. In *ACMMM*, 2021.

[3] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[4] Will Grathwohl, Kuan-Chieh Wang, Joern-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. In *ICLR*, 2020.

[5] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

[6] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *CVPR*, 2017.

[7] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[9] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*, 2017.

[10] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty, 2019.

[11] Beomsu Kim and Jong Chul Ye. Energy-based contrastive learning of visual representations, 2022.

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[13] Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *NeurIPS*, 2016.

[14] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

[15] Joshua David Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. In *ICLR*, 2020.

[16] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[17] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *CVPR*, 2015.

[18] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding, 2019.

[19] HaiYing Wang, Aonan Zhang, and Chong Wang. Nonuniform negative sampling and log odds correction with rare events data. In *NeurIPS*, 2021.

[20] Kuan-Chieh Wang, Paul Vicol, James Lucas, Li Gu, Roger Grosse, and Richard Zemel. Adversarial distillation of bayesian neural network posteriors, 2018.

[21] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere, 2020.

[22] Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. A unified contrastive energy-based model for understanding the generative ability of adversarial training. In *ICLR*, 2022.

[23] Jiayu Wu, Qixiang Zhang, and Guoxi Xu. Tiny imagenet challenge, 2017.

[24] Mike Wu, Milan Mosse, Chengxu Zhuang, Daniel Yamins, and Noah Goodman. Conditional negative sampling for contrastive learning of visual representations. In *ICLR*, 2021.

[25] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey, 2021.

[26] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2016.

A Proves of Proposition 2.1 and Proposition 2.2

Proposition 2.1. The gradient of contrastive divergence is equivalent to the following contrastive learning objective

$$-\mathop{\mathbb{E}}_{\substack{\boldsymbol{x}\sim p_d(\boldsymbol{x})\\ \boldsymbol{v}\sim q_a(\boldsymbol{v}|\boldsymbol{x})}} F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}) + \mathop{\mathbb{E}}_{\mathrm{sg}(\boldsymbol{x})\sim p_{\boldsymbol{\theta}}(\boldsymbol{x})} \log \mathop{\mathbb{E}}_{\boldsymbol{v}\sim p_d(\boldsymbol{v})} \exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})), \tag{12}$$

where $\mathrm{sg}(\boldsymbol{x})$ means $\boldsymbol{x}$ is a leaf node when calculating the gradient.

Proof. The conditional distribution is

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{v}|\boldsymbol{x}) =& \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} = \frac{\exp(-E_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}))/Z(\boldsymbol{\theta})}{\int \exp(-E_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}'))/Z(\boldsymbol{\theta})d\boldsymbol{v}'} \\ =& \frac{\exp(-E_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}))}{\int \exp(-E_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}'))d\boldsymbol{v}'} \\ =& \frac{p_d(\boldsymbol{x})p_d(\boldsymbol{v})\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}))}{\int p_d(\boldsymbol{x})p_d(\boldsymbol{v}')\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}'))d\boldsymbol{v}'} \\ =& \frac{p_d(\boldsymbol{v})\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}))}{\mathbb{E}_{p_d(\boldsymbol{v}')}\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}'))}. \end{aligned} \tag{13}$$

Then we have

$$\begin{aligned} &-\mathbb{E}_{q_a(\boldsymbol{x},\boldsymbol{v})}\frac{\partial}{\partial\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})}\frac{\partial}{\partial\boldsymbol{\theta}}E_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}), \\ =&-\mathbb{E}_{q_a(\boldsymbol{x},\boldsymbol{v})}\frac{\partial}{\partial\boldsymbol{\theta}}F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\int p_{\boldsymbol{\theta}}(\boldsymbol{v}|\boldsymbol{x})\frac{\partial}{\partial\boldsymbol{\theta}}F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})d\boldsymbol{v} \\ =&-\mathbb{E}_{q_a(\boldsymbol{x},\boldsymbol{v})}\frac{\partial}{\partial\boldsymbol{\theta}}F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\int \frac{p_d(\boldsymbol{v})\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}))}{\mathbb{E}_{p_d(\boldsymbol{v}')}\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}'))}\frac{\partial}{\partial\boldsymbol{\theta}}F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})d\boldsymbol{v} \\ =&-\mathbb{E}_{q_a(\boldsymbol{x},\boldsymbol{v})}\frac{\partial}{\partial\boldsymbol{\theta}}F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\frac{\frac{\partial}{\partial\boldsymbol{\theta}}\mathbb{E}_{p_d(\boldsymbol{v})}\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}))}{\mathbb{E}_{p_d(\boldsymbol{v})}\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}))} \\ =&-\mathbb{E}_{q_a(\boldsymbol{x},\boldsymbol{v})}\frac{\partial}{\partial\boldsymbol{\theta}}F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\frac{\partial}{\partial\boldsymbol{\theta}}\log\mathbb{E}_{p_d(\boldsymbol{v})}\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})) \end{aligned} \tag{14}$$

Since the partial derivative operator in the second term is inside $\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}$, we need to treat $\boldsymbol{x}$ as a leaf node when calculating the gradient. Recall that $q_a(\boldsymbol{x},\boldsymbol{v}) = p_d(\boldsymbol{x})q_a(\boldsymbol{v}|\boldsymbol{x})$ and we can easily get the learning objective as mentioned in the theorem. □

Proposition 2.2. By approximate $p_{\boldsymbol{\theta}}^{(t)}(\boldsymbol{x})$ with $p_d(\boldsymbol{x})$, InfoNCE is a special form of the loss in Eq.7.

Proof. We have

$$\begin{aligned} &-\mathbb{E}_{p_d(\boldsymbol{x})q_a(\boldsymbol{v}|\boldsymbol{x})}F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})) + \mathbb{E}_{p_{\boldsymbol{\theta}}^{(t)}(\boldsymbol{v})}\log\mathbb{E}_{p_d(\boldsymbol{v})}\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})) \\ \approx&-\mathbb{E}_{p_d(\boldsymbol{x})q_a(\boldsymbol{v}|\boldsymbol{x})}F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})) + \mathbb{E}_{p_d(\boldsymbol{x})}\log\mathbb{E}_{p_d(\boldsymbol{v})}\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v})) \\ =&-\mathbb{E}_{p_d(\boldsymbol{x})}\left(q_a(\boldsymbol{v}|\boldsymbol{x})F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}) - \log\mathbb{E}_{p_d(\boldsymbol{v})}\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{v}))\right) \\ =&-\frac{1}{n}\sum_{i=1}^{n}\frac{\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\boldsymbol{v}_i^+))}{\sum_{j=1}^{k}\exp(F_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\boldsymbol{v}_{ij}^-))}. \end{aligned} \tag{15}$$

which is just the InfoNCE loss. $\boldsymbol{x}_i$ is the anchor, $\boldsymbol{v}_i^+$ is the augmented view of $\boldsymbol{x}_i$ and $\{\boldsymbol{v}_{ij}^-\}$ are negative samples of $\boldsymbol{x}_i$. The last step uses Monte Carlo Estimator to estimate the integral over $q_a(\boldsymbol{v}|\boldsymbol{x})$ and $p_d(\boldsymbol{v})$. □

B Discussions on Related Work

EBMs and its Training. Among EBMs, JEM [4] is closely related to our VEM, as both works aim to establish a EBM formulation for discriminative learning methods and train the network using SGLD to sample from the model. The difference is that JEM is designed for supervised learning that requires labeled data and VEM incorporates a latent variable $\boldsymbol{v}$ to assist the self-supervised learning process. VEM gains better generalizing ability due to the self-supervised learning, which outperforms JEM in calibration and out of distribution detection tasks.

Contrastive Learning and EBMs. CEM [22] is one closely related work. While CEM focuses on interpreting the generative ability of adversarial training, ours aim to formulate and improve the training strategies of canonical contrastive SSL. Technically, different from theirs, we unify the pretext task of contrastive learning into a variational framework, and derive two new contrastive learning methods from VEM. Besides, a concurrent work [11] also proposes an energy-based formulation for contrastive learning and adopts sampling techniques to generate negative samples. A key difference is that they generate samples in the sample space, which requires 10 more training time than canonical CL methods; instead, our variational formulation is able to perform sampling in the feature space, which brings negligible computation overhead. Besides, we also propose to use the particle-based sampling method [13] to encourage feature diversity, and generally outperform SGLD adopted by theirs.

Contrastive Learning and Negative Sampling. There are many works aiming to craft better negative samples for better representation learning[15, 17, 2, 24]. Some of them intend to find samples that are hard to discriminate from the anchor [15, 2, 24]. [15] proposed a distribution over hard negatives pairs and derived an importance sampling strategy. However, different from theirs, our method does not distinguish hard and easy negative samples directly. We simply sample from $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and aim to get important negative samples.

C Pseudo Code of VEM-Langevin and VEM-SVGD

We present the pseudo codes of VEM-Lamgevin and VEM-SVGD using linear function kernel $k(x, y) = x^T y$ in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 VEM-Langevin training framework

```
1: **Input:** network $f_\theta$, epochs $E$, batch size $N$, memory bank $B$, sample number of memory bank
   $M$, feature dim $k$, noise intensity $\varepsilon$, sampling steps $\eta$, step size $\alpha$
2: **Initialize** memory bank $B$                                         ▷ size of $B : M \times k$
3: **for** $e = 0$ to $E$ **do**
4:     **for** sampled minibatch $\boldsymbol{x}$ **do**
5:         Independently draw augmentation from $\boldsymbol{x}$ and get $\boldsymbol{x}_k$ and $\boldsymbol{x}_q$
6:         $\boldsymbol{q} = f_\theta(\boldsymbol{x}_q)$, $\boldsymbol{k} = f_\theta(\boldsymbol{x}_k)$                    ▷ size of $\boldsymbol{q}, \boldsymbol{k}$: $N \times k$
7:         $\boldsymbol{q}$ = normalize($\boldsymbol{q}$, dim = 1), $\boldsymbol{k}$ = normalize($\boldsymbol{k}$, dim = 1)      ▷ size of $\boldsymbol{q}, \boldsymbol{k}$: $N \times k$
8:         **for** $i \in [1, 2, \dots, \eta]$ **do**                                      ▷ Update $B$
9:             $L = B\boldsymbol{q}^T$                                        ▷ size of $L : M \times N$
10:            $L_{norm} = \text{softmax}(L, \text{dim} = 1)$                  ▷ size of $L_{norm} : M \times N$
11:            $\Delta B = L_{norm}\boldsymbol{q}/N - \text{mean}(L_{norm}. * L, \text{dim} = 1)^T . * B$    ▷ size of $\Delta B : M \times k$
12:            Draw $M \times k$ samples independently from $\mathcal{N}(0, 1)$ to form $Q$   ▷ size of $Q : M \times k$
13:            $B = B + \alpha/i \cdot \Delta B + \sqrt{2\alpha/i} \cdot Q$
14:            $B = \text{normalize}(B, \text{dim} = 1)$
15:        **end for**
16:        Logit-pos = sum($\boldsymbol{q} . * \boldsymbol{k}$, dim = 1)                ▷ size of Logit-pos : $N \times 1$
17:        Logit-neg = $\boldsymbol{q}B^T$                                   ▷ size of Logit-neq : $N \times M$
18:        Logits = concatenate(Logit-pos, Logit-neg)             ▷ size of Logits : $N \times (1 + M)$
19:        Labels = zeros($N$)
20:        Loss = CrossEntropyLoss(Logits, Labels)
21:        Obtain gradient $\frac{\partial \text{Loss}}{\partial \theta}$ for training $f_\theta$
22:    **end for**
23: **end for**
```

Algorithm 2 VEM-SVGD (with linear kernel function) training framework

```
1: Input: network $f_\theta$, epochs $E$, batch size $N$, memory bank $B$, sample number of memory bank $M$, feature dim $k$, sampling steps $\eta$, step size $\alpha$
2: Initialize memory bank $B$                                      ▷ size of $B: M \times k$
3: for $e = 0$ to $E$ do
4:     for sampled minibatch $\boldsymbol{x}$ do
5:         Independently draw augmentation from $\boldsymbol{x}$ and get $\boldsymbol{x}_k$ and $\boldsymbol{x}_q$
6:         $\boldsymbol{q} = f_\theta(\boldsymbol{x}_q), \boldsymbol{k} = f_\theta(\boldsymbol{x}_k)$                ▷ size of $\boldsymbol{q}, \boldsymbol{k}$: $N \times k$
7:         $\boldsymbol{q} = \text{normalize}(\boldsymbol{q}, \text{dim} = 1), \boldsymbol{k} = \text{normalize}(\boldsymbol{k}, \text{dim} = 1)$   ▷ size of $\boldsymbol{q}, \boldsymbol{k}$: $N \times k$
8:         for $i \in [1, 2, \dots, \eta]$ do                              ▷ Update $B$
9:             $L = B\boldsymbol{q}^T$                                     ▷ size of $L: M \times N$
10:            $L_{norm} = \text{softmax}(L, \text{dim} = 1)$                 ▷ size of $L_{norm}: M \times N$
11:            $\Delta B = L_{norm}\boldsymbol{q}/N - \text{mean}(L_{norm}. * L, \text{dim} = 1)^T. * B$   ▷ size of $\Delta B: M \times k$
12:            $\Delta B = BB^T \Delta B/M + B$
13:            $B = B + \alpha/i \cdot \Delta B$
14:            $B = \text{normalize}(B, \text{dim} = 1)$
15:        end for
16:        Logit-pos $= \text{sum}(\boldsymbol{q}. * \boldsymbol{k}, \text{dim} = 1)$        ▷ size of Logit-pos : $N \times 1$
17:        Logit-neg $= \boldsymbol{q}B^T$                                ▷ size of Logit-neq : $N \times M$
18:        Logits = concatenate(Logit-pos, Logit-neg)        ▷ size of Logits : $N \times (1 + M)$
19:        Labels = zeros($N$)
20:        Loss = CrossEntropyLoss(Logits, Labels)
21:        Obtain gradient $\frac{\partial \text{Loss}}{\partial \theta}$ for training $f_\theta$
22:    end for
23: end for
```

D Experiments Setup

Backbones. We adopt ResNet18 [8], ResNet50 [8] as the backbones for unsupervised pretraining. Besides, PyramidNet [6], which is a network architecture improved from ResNet by gradually increasing the feature map dimension at all units instead of sharply increasing it, has superior generalization ability on supervised discriminative tasks, hence it may also learn better feature representation in unsupervised scenario and we decide to adopt it in our pretraining. The output feature map from the top network block is average-pooled and projects to a 128-D feature vector through two-layer MLP. In the downstream task of the linear classification, a linear fully connected classifier is fine-tuned on top of the frozen feature vector out of the pretrained backbone.

Hyperparameters. For the unsupervised pretraining stage, we use the SGD optimizer [16] with an initial learning rate in $\{0.5, 0.2, 0.15, 0.05\}$ for updating the backbone network and $\{1.0, 0.5\}$ for updating the memory bank, where a weight decay of 10^{-4} and a momentum of 0.9 are also applied. Cosine scheduler [14] is used to gradually decay the learning rate. We choose a temperature of $t = 0.02$ to update the memory bank and $t = 0.12$ for updating the backbone network. The batch size is set to 256 and the size of memory bank is set to 4096. The network is pretrained for 1000 epochs and the linear layer is then trained for 200 epochs in the downstream task, with a learning rate of 5 with the cosine learning rate decay and a batch size of 256.

E More experiments

E.1 Full Table of Linear Evaluation Results

We present the full results of linear evaluation with various network architectures in Table 2. The results show that the proposed VEMs can achieve the best performance among the compared models.

E.2 Evaluation on Model Robustness

A model is considered robust when the accuracy does not change significantly from the baseline accuracy under various conditions [10]. We mainly explore two aspects: (1) the accuracy of model classification when being adversarially attacked, that is, the input image is deliberately added with

Table 2: Classification Accuracies (%) on CIFAR10, CIFAR100 and Tiny-ImageNet-200 with various network architectures. All experiments are repeated for 3 times.

	Dataset	CIFAR10	CIFAR100		Tiny-ImageNet-200	
Network	Method	Test Acc	Test Acc1	Test Acc5	Test Acc1	Test Acc5
ResNet18	MoCo	$90.98_{\pm0.38}$	$65.00_{\pm0.27}$	$88.80_{\pm0.12}$	$50.50_{\pm0.42}$	$75.78_{\pm0.09}$
	SimCLR	$90.22_{\pm0.27}$	$64.81_{\pm0.26}$	$89.03_{\pm0.16}$	$50.99_{\pm0.29}$	$76.51_{\pm0.33}$
	VEM-Langevin	**91.63**$_{\pm0.31}$	$65.73_{\pm0.24}$	**89.74**$_{\pm0.36}$	$51.78_{\pm0.29}$	**77.37**$_{\pm0.09}$
	VEM-SVGD	$91.43_{\pm0.28}$	**65.77**$_{\pm0.19}$	$89.41_{\pm0.41}$	**52.10**$_{\pm0.35}$	$77.19_{\pm0.04}$
ResNet50	MoCo	$92.56_{\pm0.36}$	$69.75_{\pm0.27}$	$91.13_{\pm0.20}$	$53.25_{\pm0.30}$	$77.18_{\pm0.44}$
	SimCLR	$92.65_{\pm0.35}$	$69.94_{\pm0.36}$	$91.35_{\pm0.19}$	$56.28_{\pm0.43}$	$79.09_{\pm0.34}$
	VEM-Langevin	**93.26**$_{\pm0.23}$	$72.59_{\pm0.10}$	$92.85_{\pm0.10}$	$60.17_{\pm0.16}$	$82.18_{\pm0.26}$
	VEM-SVGD	$93.23_{\pm0.28}$	**72.79**$_{\pm0.16}$	**93.37**$_{\pm0.27}$	**60.28**$_{\pm0.25}$	**82.46**$_{\pm0.24}$
PyramidNet	MoCo	$90.35_{\pm0.49}$	$67.65_{\pm0.18}$	$90.89_{\pm0.16}$	$52.97_{\pm0.36}$	$74.20_{\pm0.07}$
	SimCLR	$90.24_{\pm0.26}$	$67.05_{\pm0.46}$	$90.70_{\pm0.35}$	$54.22_{\pm0.26}$	$76.00_{\pm0.37}$
	VEM-Langevin	$91.90_{\pm0.10}$	**69.71**$_{\pm0.25}$	**91.60**$_{\pm0.29}$	$58.12_{\pm0.11}$	$81.05_{\pm0.20}$
	VEM-SVGD	**92.00**$_{\pm0.18}$	$69.42_{\pm0.40}$	$91.58_{\pm0.14}$	**58.19**$_{\pm0.10}$	**81.18**$_{\pm0.24}$

Table 3: (a) Robust accuracies (%) against FGSM on CIFAR10, CIFAR100 and Tiny-ImageNet-200 using ResNet50 as backbone. (b) Transfer learning accuracies (%) with CIFAR10 to CIFAR100 and CIFAR100 to CIFAR10.

(a) Adversarial robustness

Dataset	Method	$\varepsilon = 4$	$\varepsilon = 8$
CIFAR10	MoCo	58.14	48.22
	SimCLR	58.23	48.46
	VEM-Langevin	**59.35**	**49.16**
	VEM-SVGD	59.22	48.94
CIFAR100	MoCo	20.22	12.47
	SimCLR	21.35	13.17
	VEM-Langevin	22.30	**15.11**
	VEM-SVGD	**22.42**	14.47
ImageNet200	MoCo	14.42	8.91
	SimCLR	14.45	8.14
	VEM-Langevin	**16.42**	9.24
	VEM-SVGD	16.22	**9.28**

(b) Transfer learning

Network	Method	C100 $\to$ C10	C10 $\to$ C100
ResNet18	MoCo	79.04	54.42
	SimCLR	79.20	56.32
	VEM-Langevin	**79.32**	56.36
	VEM-SVGD	79.29	**56.42**
ResNet50	MoCo	83.22	65.24
	SimCLR	82.59	65.68
	VEM-Langevin	**86.22**	**66.25**
	VEM-SVGD	86.04	65.94
PyramidNet	MoCo	83.44	64.12
	SimCLR	83.68	64.13
	VEM-Langevin	83.88	65.01
	VEM-SVGD	**83.92**	**65.12**

some subtle interference that people cannot detect; (2) the accuracy of model classification when the input sample distribution changes, *e.g.,* testing a model on CIFAR100, which is trained on CIFAR10.

Model robustness with respect to adversarial robustness. Since all models are not trained to resist the adversarial attack, we only perform FGSM attack [3] for evaluation. For FGSM, we perform one step on the input image $\boldsymbol{x}$ with $\boldsymbol{x}_{adv} = \boldsymbol{x} + \varepsilon/255 \cdot \operatorname{sign}(\nabla_{\boldsymbol{x}} L(\boldsymbol{\theta}, \boldsymbol{x}, y))$, where L is the loss function, y is the label and ε is the perturbation. In this experiment, we set $\varepsilon = 4$ and $\varepsilon = 8$. We all adopt the ResNet50 backbone since its performance is the best in the linear classification task. The results are shown in Table 3(a). Although the accuracies of all models suffer from the adversarial attack, our proposed VEMs still outperform the compared models on all datasets. This indicates that our method can slightly strengthen the adversarial robustness of the model.

Model robustness with respect to transfer learning. We also evaluate the pretrained models for cross-dataset transfer learning. We keep the pretrained backbone frozen, finetune the linear classifier on the target train set and test the finetuned model on the target test set. The experiments are carried on two tasks: CIFAR10 to CIFAR100 and CIFAR100 to CIFAR10. Table 3(b) shows the test results of different methods on various networks. The results show the proposed VEM achieves much competitive results on both tasks, suggesting VEM has better generalizing ability to the transfer learning tasks than compared methods. For example, it has achieved a much higher accuracy of 86.22% for CIFAR100 to CIFAR10 task than the SimCLR (82.59%) and MoCo (83.22%).

Table 4: ECE and MCE of classifiers pretrained using SimCLR and our method on CIFAR10 and CIFAR100.

	Network	ResNet18		ResNet50		PyramidNet	
Dataset	Method	ECE↓	MCE↓	ECE↓	MCE↓	ECE↓	MCE↓
CIFAR10	SimCLR	0.0129	**0.0812**	0.0163	0.767	0.0167	0.270
	VEM-Langevin	**0.00842**	0.172	**0.0136**	**0.138**	**0.0159**	**0.0970**
CIFAR100	SimCLR	0.0222	0.0751	0.0248	0.0988	0.0496	0.105
	VEM-Langevin	**0.0221**	**0.0691**	**0.0240**	**0.0748**	**0.0390**	**0.0736**

Table 5: AUROC of various models trained on CIFAR10 in OOD detection, where CIFAR100 is the out-of-distribution dataset.

Method	SimCLR	MoCo	JEM	VEM-Langevin	VEM-SVGD
AUROC	0.895	0.884	0.872	0.908	**0.909**

Table 6: Classification Accuracies of VEM-image and VEM-feature with ResNet50 on CIFAR100.

Method	Test Acc	Pretraining Time
SimCLR(Baseline)	65.14	6.02h
VEM-Feature	67.23	6.28h
VEM-Image	66.88	48.77h

E.3 Calibration

A classifier model is defined as calibrated when the predicting confidence of the classifying results, *i.e.*, $\max_y(p(y|\boldsymbol{x}))$ is consistent with the real empirical probability [5]. Thus, in a classification task, if a large number of samples with a probability of 0.6 predicted to have a label of y by the classifier are taken out, 60% of them should actually have a label of y. The calibration is an important feature of a classifier in the real-world scenarios. For high-risk applications, the confidence of classifiers in their predictions is critical. **Expected Calibration Error (ECE)** and **Maximum Calibration Error (MCE)** are usually used to measure the calibration of a classifier [5]. These two values should be close to 0 for a well-calibrated classifier. More introduction to these metrics and evaluating details are shown in Appendix F.

We evaluate both ECE and MCE of the finetuned classifiers pretrained using SimCLR and our method using Langevin on CIFAR10 and CIFAR100. We find that both classifiers are well calibrated due to their high accuracies, but our method still improves the calibration on the two datasets as can be seen in Table 4. In all settings, the ECE of our method is lower than that of SimCLR by 0.5% to 34%. In most of the settings, the MCE of our method is much lower than that of SimCLR. This may be because our models are trained on a probabilistic framework.

E.4 Out-of-Distribution Detection

Out-of-distribution (OOD) detection is a binary classification problem, where the model is required to judge whether an example is an OOD example [25]. The classifier's predictive distribution can be approached for OOD detection [20]. A useful OOD score that can be derived from this distribution is the maximum prediction probability $\max_y p(y|\boldsymbol{x})$ [9]. For evaluation, we adopt AUROC [9], which is a threshold-free metric. We use models trained on CIFAR10 with ResNet50 and let CIFAR100 be the OOD dataset. Besides, we also evaluate JEM since it also uses EBM to form the probability distribution of the data [4]. Results are shown in Table 5.

In this setting, our VEM-Langevin and VEM-SVGD both slightly outperform the baseline methods, indicating that our methods learn better representation. Surprisingly, all the SSL methods have higher scores than the SL method JEM, which may be due to the strong generalizing ability of SSL methods.

Table 7: ECE and MCE of classifiers trained using JEM and VEM-Langevinour method on CIFAR10 and CIFAR100

Method	Network	Params	CIFAR10		CIFAR100	
			ECE	MCE	ECE	MCE
JEM	WideResNet-28-10[26]	36.5M	2.29%	20.8%	4.87%	10.12%
VEM-Langevin	ResNet50	25.5M	**1.36%**	**13.8%**	**2.40%**	**7.48%**

E.5 Comparison between Feature Sampling and Image Sampling

In Section 3.2, we have discussed about whether to sample features (VEM-Feature) or sample images (VEM-Image) and we conduct an experiment to further demonstrate. We adopt CIFAR100 as the dataset and ResNet50 as the backbone. We both use the Langevin dynamics for sampling. Batch size is set to 128 and pretraining epoch is set to 400. For the size of the memory bank, we set it to 128 for VEM-Image and set it to 3072 for VEM-Feature since the size of one image equals to 24 times as much as one feature. The other settings are the same as in Section 6.1. We evaluate the classification results and record the pretraining time as shown in Table 6.

We can see that the pretraining time of VEM-Image is about 8 times as much as VEM-Feature while the test accuracies are almost the same. This indicates that it is better to sample features during the pretraining stage.

F More Details on Calibration

Expected Calibration Error (ECE) and Maximum Calibration (MCE). The two metrics work by first computing the predicting confidence $\max_y(p(y|\boldsymbol{x}_i))$ for all x_i in the test set. Then all the confidence scores of the test set are binned into M distinct bins $[0, 1/M), [1/M, 2/M), \ldots, [(M-1)/M, 1]$. For each bin B_m we can calculate its accuracy

$$acc(B_m) = \frac{1}{|B_m|}\sum_{i \in B_m} \mathbf{1}_{y_i=\hat{y}_i} \tag{16}$$

and its confidence

$$conf(B_m) = \frac{1}{|B_m|}\sum_{i \in B_m} \hat{p}_i, \tag{17}$$

where $|B_m|$ denotes the number of elements in B_m, y_i denotes the real label of $\boldsymbol{x}_i$, $\hat{y}_i = \arg\max_y(p(y|\boldsymbol{x}_i))$ denotes the predicted label of $\boldsymbol{x}_i$ and $\hat{p}_i = \max_y(p(y|\boldsymbol{x}_i))$ denotes the predicting confidence of $\boldsymbol{x}_i$.

Expected Calibration Error (ECE) simply takes a weighted average over the absolute difference between the confidence and the accuracy:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n}|acc(B_m) - conf(B_m)|. \tag{18}$$

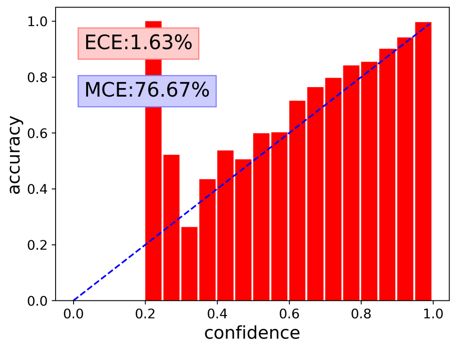
Maximum Calibration Error (MCE) is more commonly used in safety critical application. It measures the maximum discrepancy between the confidence and the accuracy:

$$MCE = \max_m |acc(B_m) - conf(B_m)|. \tag{19}$$

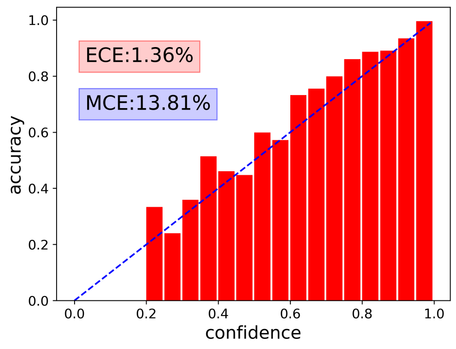
These two values should be close to 0 for a well-calibrated classifier. For a perfect calibrated classifier, they can reach 0 for any choice of M. In practice, we choose $M = 20$ for our experiments.

A reliability diagram shows the relation between the accuracy and the confidence with a bar chart. Each bar contains the images of which the classifier has the confidence of the same bin. The height of the bar denotes the average accuracy of the images in the bar. A perfect calibrated classifier will have a reliability diagram with a shape closed to the line $y = x$. We give intuitive reliability diagrams on ResNet50 in Figure 1 for better understanding.

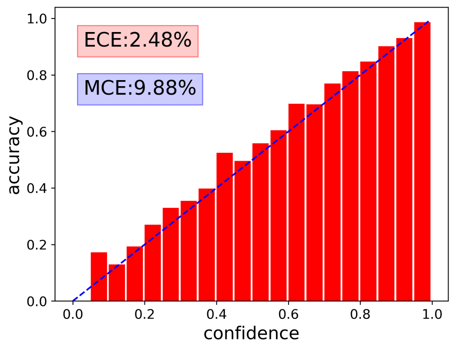
We also find an interesting point about the improvement by using self-supervised training. According to [4], Joint Energy-Based Model (JEM) also improves calibration using supervised energy-based method. We compare the results of ECE and MCE between JEM and our unsupervised method



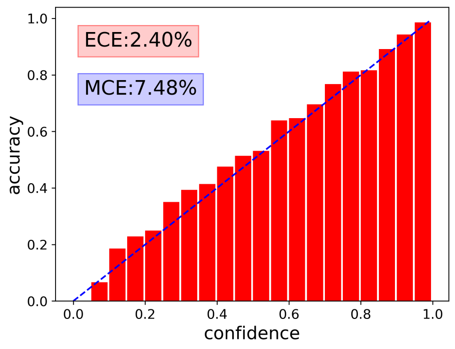
(a) SimCLR on CIFAR10



(b) VEM-LangevinOurs on CIFAR10



(c) SimCLR on CIFAR100



(d) VEM-LangevinOurs on CIFAR100

Figure 1: Reliability diagrams of SimCLR and VEM-Langevinour method on CIFAR10 and CIFAR100

Table 8: Pretraining time using ResNet50 on CIFAR100.

Method	VEM-Image	VEM-Langevin	VEM-SVGD	SimCLR	MoCo
Time/Epoch(sec)	1230.4	154.4	160.9	140.3	147.2

in Table 7. We find that our proposed unsupervised energy-based model outperforms JEM on calibration.

G Training Time

We record the training time in the unsupervised pretraining stage under the settings mentioned in Section 6.1. We calculate the training time per epoch and show the results of using ResNet50 on CIFAR100 in Table 8. We can see that VEM-Image costs 7-8 times as much time as VEM-Langevin and VEM-SVGD do, which indicates the advantage of updating features instead of images on saving time. VEM-Langevin and VEM-SVGD also have the training time comparable to SimCLR's and MoCo's.

H Experiments on ImageNet100

We also conduct experiments on a large-scale dataset ImageNet100 [18], which is a subset of ImageNet with 100 classes. We adopt ResNet18 and ResNet50 as the backbones. Due to the computational constraint, we only train 400 epochs for the unsupervised pretraining stage and finetune the

Table 9: Classification Accuracies (%) on ImageNet100 with ResNet18 and ResNet50.

Network	Method	Test Acc1	Test Acc5
ResNet18	MoCo	72.14	92.42
	SimCLR	71.54	92.16
	VEM-Langevin	**73.88**	**92.60**
	VEM-SVGD	73.79	92.55
ResNet50	MoCo	77.88	94.98
	SimCLR	78.12	94.92
	VEM-Langevin	79.64	**95.18**
	VEM-SVGD	**79.68**	95.12

linear layer for 200 epochs in the downstream task. The initial learning rate is chosen to be 0.10 for the pretraining stage and the other settings are the same as mentioned in Section 6.1.

Classification Task. We test all the models on the test set of ImageNet100 after they are finetuned. We compared out proposed methods with SimCLR and MoCo in terms of top-1 accuracy and top-5 accuracy. Results are shown in Table 9. VEM-Langevin and VEM-SVGD both outperform the compared methods with at least 1.5% in the terms of top-1 accuracy, which indicates that our proposed methods also have better representation learning ability on the large-scale dataset.

I Experiments on Hyperparameters

In this section, we conduct experiments on the influence of the hyperparameters. We choose CIFAR100 as the dataset and adopt ResNet50 as the backbone. We explore the influence of different training epoch, batch size, memory bank size, dimension of the feature space, and the sampling method adopted in SGLD or SVGD. Based on the experimental setting mentioned in Section 6.1, we tune the hyperparameters one by one and meanwhile fix the others. In each experiment, the best learning rate is searched in $\{0.5, 0.2, 0.15, 0.05\}$. The linear evaluation accuracies (%) on the CIFAR100 test dataset are reported.

Epochs. We choose the training epoch from {400, 1000}. Results are shown below.

Table 10: Comparison between different choices of the training epoch on CIFAR100 with ResNet50. Test accuracies (%) are reported.

Epochs	400	1000
SimCLR	69.72	70.32
MoCo	69.24	70.03
VEM-Langevin	**71.44**	72.70
VEM-SVGD	71.02	**72.88**

We can see that all methods benefit from a longer training procedure. Even with a smaller training epoch, our methods can still outperform the baseline methods by 1% to 2%.

Batch size. We choose the batch size from {128, 256, 512}. Results are shown below.

Table 11: Comparison between different choices of the batch size on CIFAR100 with ResNet50. Test accuracies (%) are reported.

Batch Size	128	256	512
SimCLR	67.14	70.32	70.64
MoCo	67.85	70.03	70.47
VEM-Langevin	**71.36**	72.70	73.02
VEM-SVGD	71.12	**72.88**	**73.12**

We can see that while SimCLR and MoCo have a drop of around 3% by changing the batch size from 256 to 128, our methods only drop around 1.5%. Although all methods benefit from a larger batch size, ours are less sensitive to the smaller batch size.

Memory bank size. We choose the memory bank size from $\{1024, 2048, 4096, 8192, 16384\}$ and compare the linear evaluation results. Standard deviations of two individual trials are also reported.

Table 12: Comparison between different choices of the memory bank size on CIFAR100 with ResNet50. Test accuracies (%) are reported.

Memory Bank Size	1024	2048	4096	8192	16384
VEM-Langevin	$71.52_{\pm 0.04}$	$72.04_{\pm 0.22}$	$72.60_{\pm 0.14}$	$72.24_{\pm 0.16}$	$72.26_{\pm 0.18}$
VEM-SVGD	$71.24_{\pm 0.13}$	$71.64_{\pm 0.18}$	$72.74_{\pm 0.20}$	$72.61_{\pm 0.22}$	$72.54_{\pm 0.19}$

It can be seen that VEM-Langevin will perform better with a smaller memory bank size (1024 and 2048), while VEM-SVGD will perform better with a larger one (4096, 8192, and 16384). However, we can see that the test accuracy will not grow when the memory bank size is larger than 4096. Therefore, in our experimental setting, we just set the memory bank size to 4096.

Dimension of the feature space. We choose the dimension of the feature space from $\{64, 128, 256\}$. Results are shown below.

Table 13: Comparison between different choices of the feature space dimension on CIFAR100 with ResNet50. Test accuracies (%) are reported. $\sim$ means that the network falls into the trivial point.

Feature Dimension	64	128	256
SimCLR	$\sim$	70.32	70.44
MoCo	67.73	70.03	70.26
VEM-Langevin	**71.07**	72.70	72.51
VEN-SVGD	70.67	**72.88**	**72.64**

The results imply that the best feature dimension in all options for VEM is 128 since the accuracies do not grow and the training time grows when changing the feature dimension from 128 to 256. When the feature dimension declines to 64, SimCLR breaks down and the performance of MoCo drops by 2.3%. VEM-Langevin only drops by around 1.7%, which is more stable to the reduction of the feature dimension.

Sampling method. We now study how to choose the hyperparameters for the SGLD and SVGD algorithms. We focus on the following settings:

(1) SGLD: Steps=10, Step size for the i-th step: $1/i$, Noise intensity for the i-th step: $\sqrt{2/i}$

(2) SGLD: Steps=10, Step size for the i-th step: 1, Noise intensity for the i-th step: 0.03

(3) SGLD: Steps=20, Step size for the i-th step: $1/i$, Noise intensity for the i-th step: $\sqrt{2/i}$

(4) SGLD: Steps=20, Step size for the i-th step: 1, Noise intensity for the i-th step: 0.03

(5) SVGD: Steps=10, Step size for the i-th step: $1/i$

(6) SVGD: Steps=10, Step size for the i-th step: 1

(7) SVGD: Steps=20, Step size for the i-th step: $1/i$

(8) SVGD: Steps=20, Step size for the i-th step: 1

The fixed step size and the fixed noise intensity for SGLD are also adopted in JEM[4] and many other methods. We compare VEM using these sampling methods in the terms of test accuracy and the results are shown below.

Table 14: Comparison between different choices of the sampling method on CIFAR100 with ResNet50. Test accuracies (%) are reported.

Methods (SGLD)	(1)	(2)	(3)	(4)
Test Acc (%)	72.70	72.14	72.78	71.82
Methods (SVGD)	(5)	(6)	(7)	(8)
Test Acc (%)	72.88	72.45	72.96	72.23

We can see that there is no big difference between the results of different methods. However, we can find that decreasing step size is better than a fixed step size. Therefore, we adopt a decreasing step size in our paper. More sampling steps will not lead to significantly better results and will lead to longer training time, so we choose the sampling step to be 10.