Generator Born from Classifier
-Supplementary Material-

Table of Contents

A Variable Definitions

In Tab. 1, we provide descriptions and corresponding notations for the variables employed throughout the manuscript. Some commonly used variables and their notations have been omitted.

B Further Discussions of the Proposed Method

B.1 Pseudocode of Proposed Methods

In the main text, we detailed the process of deriving a generator from a pre-trained classifier, as well as its extension in scenarios involving multiple classifiers. Herein, we summarize these two methods in Algs. 1 and 2, respectively.

B.2 Special Case for Binary Classification

In the main text, we discuss the transformation of a multi-class classifier trained with cross-entropy into a generator. The theoretical foundation of our discussion is the theory of Maximum-Margin Bias of a quasi-homogeneous model. However, we observe that the theory of Maximum-Margin Bias has a specific version for binary classifiers trained with exponential loss. In this section, we provide a description of this specific version and design our method specifically for binary classification problems. Following the structure of the main text, we first present the description of the Maximum-Margin Bias theory, then provide its corresponding KKT conditions, and finally introduce the loss function for transforming a binary classifier into a generator.

It is worth noting that our discussion on transforming a binary classifier into a generator is not due to the inability of our proposed method to handle binary classification problems. Classifiers trained with cross-entropy can indeed support binary classification tasks, and our method can also transform the corresponding binary classifier into a generator. However, due to the simplicity of binary classification problems, often a neural network with a scalar output is sufficient for prediction,

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Category	Variable	Description
Generator Network	θ	Parameter of the generator network
	$g(\cdot,\cdot;\theta)$ and $g(\cdot,\cdot,\cdot;\theta)$	Generator parameterized by θ
Classifier Network	ζ	Parameter of the classifier network
	$\Phi(\cdot;\zeta)$	Classifier parameterized by ζ
	$\bar{\zeta}$	The normalized version of ζ, such that, $\|\bar{\zeta}\|_\Lambda^2 = 1$.
	$\tilde{\zeta}$	The convergence point of $\bar{\zeta}$ under the gradient flow
	ζ'	The optimization variable used in optimization problem in Theorem 1
	Z_1	The set of parameters whose corresponding element in $\tilde{\Lambda}$ are non-zero
	Z_2	The set of parameters whose corresponding element in $\tilde{\Lambda}$ are zero
In the setting of training a generator using multiple classifiers	T	The number of classifiers
	$\Phi^{(t)}$	The t-th classifier
	$\mathcal{T}_y$	A set of classifier indices, each index pointing to a classifier whose training data contains images labeled as y
Quasi-Homogeneous Model	Λ	The coefficient of a quasi-homogeneous model, which is a (non-zero) positive semi-definite diagonal matrix and offers different rates of change for different parameters.
	Λ_{jj}	The element at the j-th row and j-th column of the matrix Λ
	λ_{max}	The maximum value among the elements in the matrix Λ
	$\tilde{\Lambda}$	A modification of the matrix Λ, retaining its original shape, obtained by setting all elements not equal to λ_{max} to zero.
	α	The scalar used in the definition of quasi-homogeneous model, describing the scaling of the network output and the parameters
	$\|\|\zeta\|\|_\Lambda^2$	The seminorm of ζ corresponding to Λ, $\|\|\zeta\|\|_\Lambda^2 := \zeta^T\Lambda\zeta$
	$\bar{\Lambda}$	A transformation of Λ applied to simplify the notation.
	Λ'	The coefficient of a quasi-homogeneous model, which is derived by taking the derivative of a quasi-homogeneous corresponding to one of the model parameters
Others	$\mathcal{S}_i$	For the sample $\boldsymbol{x}_i$, $\mathcal{S}_i \subset \mathcal{Y}$ is the subset containing labels for which the classifier's prediction probabilities are second highest.
	q_{min}	The minimum classification margin

Table 1: Table of Variables.

Algorithm 1 Training Procedure of the Generative Model with One Classifiers

Input: Pre-trained network: $\Phi(\cdot;\zeta)$, number of training samples: N, batch size M, total epochs: E, and other hyper-parameter: δ and β.

Initialize: Generative model: $g(\cdot,\cdot;\theta)$, μ predictor: $h(\cdot,\cdot;\eta)$, and α.

$\boldsymbol{Y} \leftarrow$ Random sample of M labels
$\Lambda \leftarrow$ Solution of Eq. (10)
$\lambda_{max} \leftarrow \max_j \Lambda_{jj}$
$\tilde{\Lambda} \leftarrow zero\_like(\Lambda)$
for j **in** $1, \cdots, |\zeta|$ **do**
    if $\Lambda_{jj} == \lambda_{max}$ **then**
        $\tilde{\Lambda}_{jj} = \lambda_{max}$
    end if
end for
for e **in** $1, \cdots, E$ **do**
    $\bar{\Lambda} \leftarrow \tilde{\Lambda} e^{\alpha(2\Lambda - \mathbf{I})}$
    $\epsilon \leftarrow$ Random sample of M Gaussian noise
    $\boldsymbol{X} \leftarrow g(\epsilon, \boldsymbol{Y}; \theta)$
    $\mu \leftarrow h(\boldsymbol{X}, \boldsymbol{Y}; \eta)$
    Compute $L(\theta, \eta, \alpha)$
    Update θ, η, and α using gradient descent
end for

Algorithm 2 Training Procedure of the Generative Model with Multiple Classifiers

Input: Pre-trained networks: $\{\Phi^{(t)}(\cdot;\zeta^{(t)})\}_{t\in[T]}$, number of training samples for each classifier: $\{N^{(t)}\}_{t\in[T]}$, batch size M, total epochs: E, and other hyper-parameter: δ and β.

Initialize: Generative model: $g(\cdot,\cdot,\cdot;\theta)$, μ predictor: $h(\cdot,\cdot,\cdot;\eta)$, and $\{\alpha^{(t)}\}_{t\in[T]}$.

for t **in** $1, \cdots, T$ **do**
    $\boldsymbol{Y}^{(t)} \leftarrow$ Random sample of M labels
    $\Lambda^{(t)} \leftarrow$ Solution of Eq. (10)
    $\lambda^{(t)}_{max} \leftarrow \max_j \Lambda^{(t)}_{jj}$
    $\tilde{\Lambda}^{(t)} \leftarrow zero\_like(\Lambda^{(t)})$
    for j **in** $1, \cdots, |\zeta^{(t)}|$ **do**
        if $\Lambda^{(t)}_{jj} == \lambda^{(t)}_{max}$ **then**
            $\tilde{\Lambda}^{(t)}_{jj} = \lambda^{(t)}_{max}$
        end if
    end for
end for
for e **in** $1, \cdots, E$ **do**
    for t **in** $1, \cdots, T$ **do**
        $\bar{\Lambda}^{(t)} \leftarrow \tilde{\Lambda}^{(t)} e^{\alpha^{(t)}(2\Lambda^{(t)} - \mathbf{I})}$
        $\epsilon^{(t)} \leftarrow$ Random sample of M Gaussian noise
        $\boldsymbol{t} \leftarrow vectorize\ t$
        $\boldsymbol{X}^{(t)} \leftarrow g(\epsilon^{(t)}, \boldsymbol{Y}^{(t)}, \boldsymbol{t}; \theta)$
        $\mu^{(t)} \leftarrow h(\boldsymbol{X}^{(t)}, \boldsymbol{Y}^{(t)}, \boldsymbol{t}; \eta)$
        Compute $L^{(t)}(\theta, \eta, \alpha^{(t)})$
    end for
    Compute $\sum_{t\in[T]} L^{(t)}(\theta, \eta, \alpha^{(t)})$
    Update θ, η, and $\{\alpha^{(t)}\}_{t\in[T]}$ using gradient descent
end for

rendering cross-entropy loss inapplicable. Therefore, to ensure the generality of our method, we provide a special design specifically for binary classification problems. This represents a particular extension of our method.

Let $\Phi(\cdot;\zeta) : \mathcal{R}^d \to \mathcal{Y}$ denote the classifier parameterized by ζ and trained on binary classification dataset $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with each $(\boldsymbol{x}_i, y_i) \in \mathcal{R}^d \times \mathcal{Y}$. Here, $\mathcal{Y} = \{-1, 1\}$ is a binary label space. let $L(\zeta) := \sum_{i=1}^N l(\Phi(\boldsymbol{x}_i;\zeta), y_i)$ denote the exponential loss of Φ on D, where $l(\Phi(\boldsymbol{x}_i;\zeta), y_i) = e^{-y_i}\Phi(\boldsymbol{x}_i;\zeta)$. Again, we require the classifier network to be a Λ-quasi-homogeneous model with similar definitions of seminorm $||\cdot||_\Lambda^2$, maximal element λ_{max}, normalized parameters $\bar{\zeta}$ and $\tilde{\Lambda}$.

The Quasi-Homogeneous Maximum-Margin Theorem states as follows.

Theorem 2 (Quasi-Homogeneous Maximum-Margin Theorem from Binary Classifier [Kunin et al., 2023])**.** *Let $\Phi(\cdot;\zeta)$ denote a Λ-quasi-homogeneous binary classifier trained on D with exponential loss L. Assume that: (1) for any fixed $\boldsymbol{x}$, $\Phi(\boldsymbol{x};\zeta)$ is locally Lipschitz and admits a chain rule [Davis et al., 2020, Lyu and Li, 2020]; (2) the learning dynamic is described by a gradient flow [Lyu and Li, 2020]; (3) $\lim_{t\to\infty}\bar{\zeta}(t)$ exists; (4) $\exists\kappa > 0$ such that only ζ with $||\zeta||_{\Lambda_{\max}} \geqslant \kappa$ separates the training data; and (5) $\exists t_0$ such that $L(\zeta(t_0)) < N^{-1}$. $\exists\alpha \in \mathcal{R}$ such that $\tilde{\zeta} := \psi_\alpha(\lim_{t\to\infty}\bar{\zeta}(t))$ is a first-order stationary point of the following maximum-margin problem*

$$\min_{\zeta'} \quad \frac{1}{2}||\zeta'||_{\tilde{\Lambda}}^2 \tag{1a}$$

$$\text{s. t.} \quad y_i\Phi(x_i;\zeta') \geqslant 1 \quad \forall i \in [N]. \tag{1b}$$

Again, Theorem 2 implies that the neural network parameters converge to the first-order stationary point (or the Karush–Kuhn–Tucker point (KKT) point) of the optimization problem in Eq. (1). Let $\{\mu_i\}_{i\in[N]}$ denote the set of KKT multipliers, the KKT condition can be written as follows.

$$\tilde{\Lambda}\tilde{\zeta} = \sum_{i\in[N]} y_i\mu_i\nabla_{\tilde{\zeta}}\Phi(x_i;\tilde{\zeta}); \tag{2a}$$

for all $i \in [N]$:

$$y_i\Phi(x_i;\tilde{\zeta}) \geqslant 1, \tag{2b}$$

$$\mu_i \geqslant 0, \tag{2c}$$

$$\mu_i[\Phi(x_i;\tilde{\zeta}) - 1] = 0. \tag{2d}$$

To avoid scaling of the neural network parameters during the evaluation of $\nabla_{\tilde{\zeta}}\Phi(\cdot;\tilde{\zeta})$ and $\Phi(\cdot;\tilde{\zeta})$, similar to the approach in the main text, we rewrite Eq. (2) based on the definition as follows:

$$\bar{\Lambda}\zeta = \sum_{i\in[N]} y_i\mu_i\nabla_\zeta\Phi(x_i;\zeta); \tag{3a}$$

for all $i \in [N]$:

$$\Phi(x_i;\zeta) \geqslant e^{-\alpha}, \tag{3b}$$

$$\mu_i \geqslant 0, \tag{3c}$$

$$\mu_i[\Phi(x_i;\zeta) - e^{-\alpha}] = 0, \tag{3d}$$

where the new scaling parameter $\bar{\Lambda} := \tilde{\Lambda}e^{\alpha(2\Lambda-\mathbf{I})}$.

Let g denote the conditional generator parameterized by θ to generate input $x = g(\epsilon, y;\theta)$ given the corresponding label y and random noise ϵ. To ensure that the generated samples satisfy the stationary condition in Eq. (3a), the $L_{stationarity}$ is designed to be

$$L_{stationarity}(\theta,\eta) := ||\frac{1}{N}\bar{\Lambda}\zeta - \frac{1}{M}\sum_{i\in[M]} y_i\mu_i\nabla_\zeta\Phi(x_i;\zeta)||. \tag{4}$$

To approximate primal feasibility and complementary slackness in Eqs. (3b) and (3d), $L_{duality}$ is designed to be

$$L_{duality}(\theta, \alpha) := \frac{1}{M} \sum_{i \in [M]} [\max\left(\Phi(x_i; \zeta) - e^{-\alpha} - \delta, 0\right) - \min\left(\Phi(x_i; \zeta) - e^{-\alpha}, 0\right)], \quad (5)$$

where $\delta > 0$ is a hyper-parameter.

For the KKT multipliers, again, the proxy variables μ_i' can be used by defining $\mu_i := ReLU(\mu_i')$. For each generated sample x_i, $\mu_i' \in \mathcal{R}$ is learned as $\mu_i' = h(x_i, y_i; \eta)$ by network h parameterized by η. For α, we still treat it as a learnable parameter. For Λ, the estimation method discussed in the main text is still feasible for the binary classification case.

By combining $L_{stationarity}$ and $L_{duality}$ the final loss has the same structure as the loss proposed in the main text:

$$L = L_{stationarity}(\theta, \eta) + \beta L_{duality}(\theta, \alpha), \quad (6)$$

where β is the balancing hyper-parameter.

For the scenario in which a generator is trained using multiple classifiers, the extension discussed in the main text can be directly applied to a set of binary classifiers or a set of binary classifiers and multi-class classifiers as well.

B.3 Further Comparison with Other Methods

Comparison with DeepDream [Mordvintsev et al., 2015]. DeepDream's objective is to visualize patterns learned by neural networks, while our method aims to train a generator for the conditional sampling of images. Technically, to generate each image, DeepDream requires an independent gradient optimization process aimed at maximizing a certain activation or network output. By contrast, once our generator is trained, images can be produced by randomly sampling noise. This leads to two outcomes: firstly, DeepDream has larger computational complexity than our method does and takes longer to generate each image compared to our method; secondly, DeepDream has a larger parameter space, resulting in superior performance. These comparative conclusions are summarized in Tab. 3, and a comparison of the generated images is shown in Fig. 1. Admittedly, it is challenging to determine superiority based solely on visual effects.

Comparison with [Haim et al., 2022, Buzaglo et al., 2023] (1) Our method is to train a generator network from sketch, which is capable of transforming random noise into generated data of good perceptual quality. By contrast, the task of [Haim et al., 2022, Buzaglo et al., 2023] is to recover training data. Our task is more challenging in two respects. Firstly, we aim to generate samples that are not present in the original dataset but still maintain good perceptual quality. Secondly, our generation process is controllable and supports conditional sampling. (2) [Haim et al., 2022] only supports binary classifiers, while our work and [Buzaglo et al., 2023] support multi-class classifiers. (3) [Haim et al., 2022, Buzaglo et al., 2023] only support homogeneous networks. By contrast, our method supports quasi-homogeneous networks, which encompass a broader range of networks. (4) [Haim et al., 2022, Buzaglo et al., 2023] optimize directly on the pixel space. Specifically, for each run, the number of optimizable parameters equals the batch size multiplied by the image size. However, our optimizable parameters only include the parameters of the generator. In our implementation, it is significantly lower than those in [Haim et al., 2022, Buzaglo et al., 2023].

B.4 Further Discussion about Limitations

Our method has two primary limitations. (1) Estimating Λ introduces additional computational overhead. Given that our approach requires the classifier to be a quasi-homogeneous model, we need to determine the classifier's Lambda before training the generator. The method we provide in our paper for calculating Lambda can be computationally intensive, especially when the classifier has many parameters. (2) Our method exhibits class bias, with significant variations in the generation quality for different categories. We hypothesize that there might be two reasons leading to this phenomenon. First, different categories may have varying complexities in their data distributions, making it more challenging for the generator to produce certain categories. Second, the generator might inherit the classifier's bias. If the classifier does not fit well for certain categories, the information about that

Figure 1: Generator-produced samples. The generator is trained using a single classifier trained on the MNIST dataset.

Condition	Parameter	Region
Generator	learning rate	$10^{\text{Uniform}(-3.5,-2.5)}$
	batch size	$\{128, 256, 512\}$
	weight decay	$10^{\text{Uniform}(-5,-3.5)}$
	β	$10^{\text{Uniform}(-3,2)}$
	$\beta_{TotalVariation}$	$10^{\text{Uniform}(-3,-2)}$

Table 2: Hyper-Parameters' Searching Regions.

category in the classifier might be limited or inaccurate. This, in turn, would affect the generator's performance in that particular category.

B.5 Further Discussion about Social Impact

Our method, which trains a generator from deep learning models, may have risks related to privacy leakage and data misuse. If classifiers are trained on data containing personal information, such as medical records, personal photos, or social media posts, the generation of this data can directly compromise individual privacy. Beyond privacy concerns, the generated data can be maliciously exploited for purposes like creating fake news, fraud, or identity theft. In summary, generating training data for deep learning models not only threatens privacy but also opens the door to various forms of data misuse, underscoring the ethical and security considerations when working with the proposed method.

C Additional Experimental Results and Implementation Details

C.1 Further Experimental Results

In this section, we show more experimental results on various datasets. First, in Fig. 1 and Fig. 2, we show more generated images of the MNIST and CelebA datasets, respectively. Then, in Fig. 3, we provide the generated images of the Fashion MNIST dataset.

C.2 Implementation Details

The description of the training set constructed from the MNIST and CelebA datasets is discussed in the main text. Here, we discuss the training set for our experiments on the Fashion MNINT [Xiao

Figure 2: Generator-produced samples. The generator is trained using a single classifier trained on the CelebA dataset.

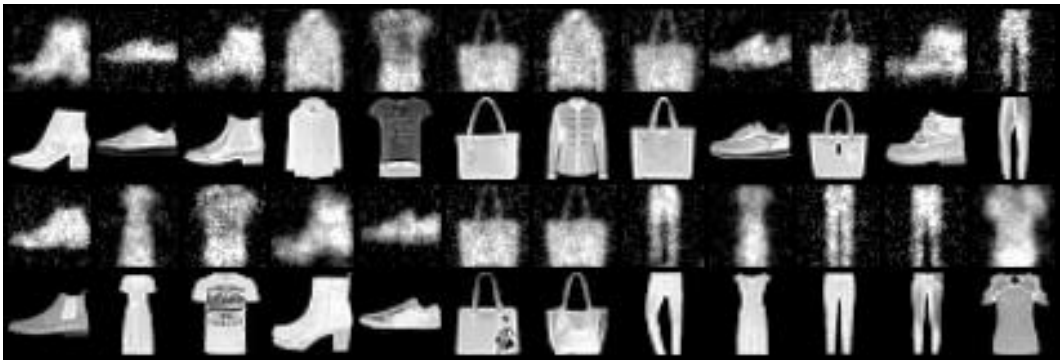

Figure 3: Generator-produced samples. The generator is trained using a single classifier trained on the Fashion MNIST dataset.

et al., 2017].For the Fashion MNINT dataset, we set up a classification task with 10 classes and 500 training data (50 images per class) randomly sampled from the original training set.

For all the classifiers, network parameters were initialized using Kaiming initialization [He et al., 2015] and trained until the classification loss converges using full batch gradient descent with a learning rate of 0.01. The weight decay of the learning is set to be 0.0001. For the generators, network parameters were initialized using Kaiming initialization [He et al., 2015] and trained for $50,000$ epochs using Adam optimizer [Kingma and Ba, 2015]. The values or the search regions of other important hyperparameters are shown in Tab. 2. The architecture of the classifiers and generators are the same for all our experiments and has been discussed in the main text.

All of our experiments are deployed on NVIDIA RTX A5000.

References

Gon Buzaglo, Niv Haim, Gilad Yehudai, Gal Vardi, and Michal Irani. Reconstructing training data from multiclass neural networks, 2023. 5

Damek Davis, Dmitriy Drusvyatskiy, Sham M. Kakade, and Jason D. Lee. Stochastic subgradient method converges on tame functions. *Found. Comput. Math.*, 20(1):119–154, 2020. 4

Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022. 5

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015. 7

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 7

Daniel Kunin, Atsushi Yamamura, Chao Ma, and Surya Ganguli. The asymmetric maximum margin bias of quasi-homogeneous neural networks. *International Conference on Learning Representations (ICLR)*, 2023. 4

Kaifeng Lyu and Jian Li. Gradient descent maximizes the margin of homogeneous neural networks. In *International Conference on Learning Representations (ICLR)*, 2020. 4

Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks, 2015. URL https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html. 5

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. 6