4-11-2023

Resampling Generative Models: An Empirical Study

Prachi Ramesh Jadhav

Resampling Generative Models: An Empirical Study

by

Prachi R. Jadhav

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Major: Computer Science

The University of Memphis

May 2023

Abstract

Despite Generative AI's rapid growth (e.g., ChatGPT, GPT-4, Dalle-2, etc.), generated data from these models may have an inherent bias. This bias can propagate to downstream tasks e.g., classification, and data augmentation that utilize data from generative AI models. This thesis empirically evaluates model bias in different deep generative models like Variational Autoencoder and PixelCNN++. Further, we resample generated data using importance sampling to reduce bias in the generated images based on a recently proposed method for bias-reduction using probabilistic classifiers. The approach is developed in the context of image generation and we demonstrate that importance sampling can produce better quality samples with lower bias. Next, we improve downstream classification by developing a semi-supervised learning pipeline where we use importance-sampled data as unlabeled examples within a classifier. Specifically, we use a loss function called as the semantic-loss function that was proposed to add constraints on unlabeled data to improve the performance of classification using limited labeled examples. Through the use of importance-sampled images, we essentially add constraints on data instances that are more informative for the classifier, thus resulting in the classifier learning a better decision boundary using fewer labeled examples.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Generative learning using deep neural networks [41] has gained significant attention over the last several years. Specifically, in generative learning, the goal is to model the data distribution, and thus, in theory, generative models can generate unlimited samples that can be used in downstream tasks such as classification. Deep generative models (DGMs) have been successful in generating realistic images [11], videos [20], text [5], audio [43] etc.

In the last decade, several types of DGMs have been developed. Prominent among those include Variational Auto Encoders (VAEs) [22], Generative Adversarial Networks (GANs) [11], Autoregressive models (ARMs) [38] and normalizing flow models (NFMs) [8]. All of these models use different approaches to model the data distribution. Specifically, VAEs learn a latent variable model using deep network layers to encode and decode the latent vectors. GANs use a *likelihood-free* approach where the idea is to train a generator-discriminator pair in an adversarial manner with the generator learning better representations of the data to generate samples for the discriminator and the discriminator learning to discriminate between real and generated samples. ARMs generate sequential data based on autoregessions that was widely used for predictions in time-series models. ARMs can compute the likelihood in a tractable manner since they assume the autoregressive property. On the other hand, VAEs cannot compute the likelihood tractably but rather use variational inference to approximate the likelihood using deep encoders and decoder layers. This allows VAEs to learn complex feature representations. In NFMs, the idea is to combine the properties of both VAEs and ARMs. Specifically, NFMs can compute the likelihood in a tractable manner and at the same time, they can also learn complex feature representations like VAEs. This is done by using simple density functions with tractable likelihoods and then mapping them to more complex probability distributions using the data samples.

Regardless of the type of generative model, it is infeasible for the model to learn the data distribution exactly [36]. That is, each type of generative model makes underlying assumptions about the data that may or may not be always valid. For instance, in VAEs, to make variational inference feasible through the neural network layers, we assume that the latent vectors that represent the underlying characteristics of the data are normally distributed [22]. In general, if the data distribution is simple enough, then clearly, we can sample directly from the distribution and may not need complex approaches such as DGMs. Therefore, there is a need to improve the quality of samples that are generated from DGMs. In particular, one way to quantify the quality of samples generated by a DGM is based on the *bias* in the model's distribution. This type of bias can result in problems such as *mode-collapse* [37, 29, 33] where the DGM samples from a single mode of a multi-modal distribution and this results in generating samples that are very similar. A recently proposed approach by Grover et al. [13] addressed this problem and tried to reduce the bias of the samples generated by a DGM. Specifically, here, the main idea is to use an approach called *importance sampling* [25] to generate weighted samples instead of unweighted samples. The importance weights encode the importance of the generated samples with respect to the true distribution. For example, as shown in Fig. 1.1, if the goal is to generate samples from a complex distribution that is hard to sample, in importance sampling, we instead generate samples from a simpler distribution called the *proposal distribution* and weight these samples based on a *ratio* of probabilities. Specifically, for each sample, we divide the probability of the sample original distribution with its probability in the proposal distribution. This ratio acts as the importance weight for a sample. In [13], using the idea of importance sampling, we draw samples from the DGM and weight it based on the data distribution which is the true distribution from which we want to draw samples from.

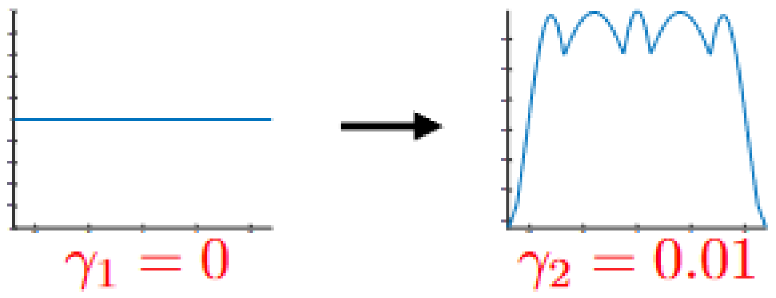

Figure 1.1: Sampling Importance Resampling, adopted from [24]

Unfortunately, to compute importance weights, we need to compute the ratio between the probability of a sample in the true distribution and its probability in the proposal distribution. This is infeasible in DGMs since we do not know how to compute this probability in a tractable manner, i.e., the data distribution is assumed to be a complex unknown distribution. In [39], a clever trick was proposed to weigh samples based on a *calibrated probabilistic classifier* [31, 14]. That is, this classifier is trained to compute the importance weights for samples that are generated from a DGM. While this approach seems generally applicable to *any* DGM, depending on the type of samples that are generated by the DGM, the trained probabilistic classifier could obtain very different results. Thus, the quality of the importance weights depends on how well we can train the probabilistic classifier.

In this thesis, we train a *calibrated* probabilistic classifier to perform importance sampling for images generated from VAEs and ARMs. Specifically, we use the well-known CIFAR10 image dataset [23] to train the generative models. We use convolutional VAEs as our VAE model for image generation and for ARMs, we use the most well-known image generation model called PixelCNN++ [38]. We implement sample importance resampling (SIR) [26, 9] as suggested in [13] to resample from the distribution learned by the probabilistic classifier for the generated samples. However, in [13], it was suggested that SIR can be implemented using a standard *Multinomial roulette* sampling method. in our experiments, we observed that for the weights that were generated, this approach failed to scale and yielded poor results. Therefore, we implemented a novel SIR where we approximate the Multinomial distribution over importance weights which are probability ratios (computed from the probabilistic classifier) using a *Gumbel-softmax* distribution [19, 27]. This allows us to resample images from the importance distribution more efficiently. To empirically compare the quality of the the generated images with and without importance sampling, we use different standard metrics such as the inception score [37], the Fretchet distance [17] and the kernel inception distance [4].

Next, we develop a novel approach to use the generated images to improve downstream classification. Specifically, in [44], a loss function called the *semantic loss* was proposed to add symbolic knowledge to deep network training. Here, we add exactly-one constraints (as specified in [44]) over generated images. Specifically, the idea is that we want each generated image to belong to exactly one class (among all possible classes). Thus, as in semi-supervised learning, we can now treat the generated images as unlabeled data and add a limited number of labeled examples to train the classifier. The semantic loss function learns to separate the labeled examples and at the same time assign classes to the unlabeled examples. Thus, assuming that the unlabeled examples has information about the classes, this will result in a more general classifier even using limited labeled examples. Further, by adding constraints on informative generated images, we can learn a more effective classifier. Using our SIR approach, we resample the generated images which are the most informative for the classifier. We evaluate our approach by comparing the performance of classification on CIFAR10 using limited labeled and a large number of generated images. Our results show that utilizing the importance sampling, we are able to learn a more accurate classifier trained on the semantic loss funtion as compared to using the generated images directly.

To summarize, our contributions in this thesis are as follows.

- We empirically evaluate the approach proposed in [13] to understand how well importance weights can be estimated for VAEs and PixelCNN using the CIFAR-10 dataset.

- We implement the Gumbel-softmax sampling to efficiently perform SIR based on the importance weight distribution from a calibrated probabilistic classifier.

- We apply the semantic loss function proposed in [44] to add constraints over images generated (and resampled) from generative models such that the unlabeled generated data can augment labeled examples within a semi-supervised classifier.
- We develop an open-source implementation in Pytorch that integrates the semantic loss function with importance sampling for generative models for image datasets.

Chapter 2

Background

Variational Autoencoders (VAEs)

Autoencoders are a general architecture that tries to reconstruct the input data. Thus, the loss of an autoencoder is measured by the difference between the original data and the reconstructed data. In VAEs, the main idea is to approximate the data distribution with a *variational* approximation. That is, we assume that there is a simpler distribution that can approximate the true distribution. In regular variational inference, we can compute the parameters of the distribution by minimizing the distance (KL-Divergence) between the original distribution and the variational approximation. The main idea in VAEs is to use deep network layers to perform variational inference. Specifically, the VAE architecture consists of an encoder and a decoder each of which can contain multiple deep network layers. The encoder learns the approximate distribution that encodes the input into a *latent-vector* space. Thus, each input can be represented as a combination of the latent vectors. The decoder learns to reconstruct the input from a sample in the latent-vector space. The closer the decoder is in reconstructing the original input from the latent space, the smaller is its loss. The entire encoder-decoder is trained end-to-end. To ensure that the KL-divergence is minimized, the distribution learned by the encoder for latent-vector space is assumed to be Gaussian. Thus, it has mean and covariance as the parameters to define this distribution.

However, it turns out that neural networks cannot be trained directly since backpropagation does not work when sampling with random variables is needed. Therefore, a reparameterization trick is used to change the Gaussian to one with 0 mean and an identity covariance and this allows us to apply backpropagation to train VAEs. For generating images, the encoder uses convolutional layers that extract image features and the decoder uses deconvolutional layers that reconstruct the image. Fig. 2.1 illustrates a VAE architecture for image generation.

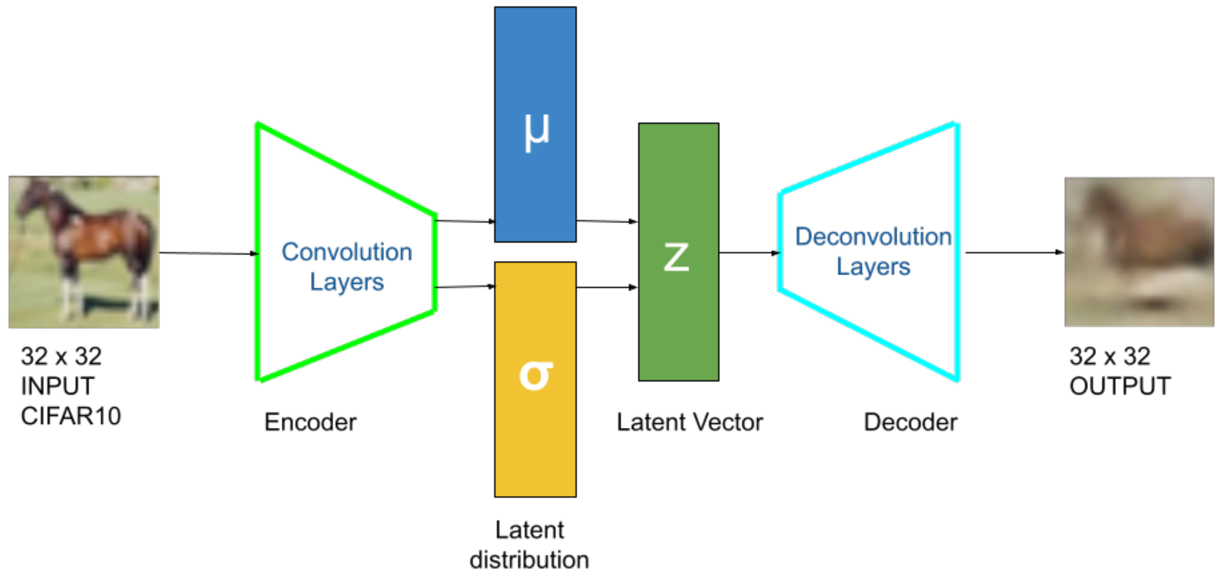

Figure 2.1: Variational Autoencoder (VAE) workflow for our generated samples

PixelCNN

PixelCNN is a generative model for images that is based on generating an image pixel-by-pixel. Specifically, we assume that each pixel is based on pixels that were previously generated. Thus, the image is generated from the top-left corner pixel all the way to the bottom-right corner pixel. PixelCNN formulates the joint distribution over pixels in an image as a product of conditional distributions.

$$P(\mathbf{x}) = \prod_i P(x_i|x_1 \dots x_{i-1}) \tag{2.1}$$

Each conditional distribution is the probability of a pixel value conditioned on the values of pixels that appeared before it. This assumes an ordering over pixels such as starting from the top-left to ending at the bottom-right corner of the image. PixelCNN uses convolutional layers to predict the pixel value from *masked* images. Specifically, the masks are generated to ensure that while generating a pixel, the convolutional layer only has access to pixels that were generated before itself in the ordering. All pixels that are to be generated later are masked in the image. Thus, we have a stack of convolutional layers, each of them trying to predict a pixel value from a masked image. During training, these predictions can be made in parallel which speeds up training. However, to generate images, the prediction needs to be performed in sequence. That is, we generate the image one pixel at a time. Thus, PixelCNNs are slow in generating images compared to approaches such as VAEs.

Importance Sampling

Importance sampling is a general sampling technique that can be used to sample from distributions that are hard-to-sample from. Specifically, let us suppose that we want to sample from a distribution $P(\mathbf{x})$ but cannot do so directly since $P(\mathbf{x})$ is a complex distribution. We instead sample from a simpler distribution $Q(\mathbf{x})$ which is called as the *proposal* distribution. To account for the fact that we sampled from a different distribution, we weight the sample $\mathbf{x}$ as $w = \frac{P(\mathbf{x})}{Q(\mathbf{x})}$. This is called as the importance weight for the sample $\mathbf{x}$. We can now compute the expected value for any function w.r.t distribution $P(\cdot)$ based on samples drawn from $Q(\cdot)$ using the importance weights. That is, suppose we want to compute $\mathbb{E}_P[f(\mathbf{x})]$, we can approximate it with T samples from $Q(\cdot)$, $\mathbf{x}_1 \dots \mathbf{x}_T$ and compute the approximate expectation as,

$$\hat{\mathbb{E}}_P[f(\mathbf{x})] = \frac{1}{T}\sum_{i=1}^{T} w(\mathbf{x}_i) f(\mathbf{x}_i) \tag{2.2}$$

$$w(\mathbf{x}_i) = \frac{P(\mathbf{x_i})}{Q(\mathbf{x_i})}$$

The proposal distribution plays a major role in the type of estimates obtained through importance sampling. Specifically, if the proposal distribution is close to the true distribution, then the estimates computed through importance sampling have a smaller variance. The requirement for the proposal distribution is that it must be non-zero for all samples for which the probability of the true distribution is also non-zero. It can be shown that the estimates obtained through importance sampling are asymptotically unbiased.

Semi-Supervised Multi-Class Classification

Semi-supervised learning is a machine learning technique in which the training data set contains both labeled and unlabeled data. On the other hand, the goal of multi-class classification is to categorize a set of input data into one of several classes. All of the data points in the training set are labeled in traditional supervised learning, which means that each data point has a corresponding output value. In contrast, semi-supervised learning trains only on a subset of the data points are labeled, leaving the remainder unlabeled. It has been used successfully in a variety of applications such as speech recognition, natural language processing, image classification, anomaly detection, etc.

Augmented Loss Function using Symbolic Knowledge

Current limits of deep learning include but are not limited to the need for large data, poor generalization on unseen data, lack of reasoning in best fitting the data, and explainability of the trained models [10]. Robust learning can be achieved by integrating additional knowledge into deep learning. Additional knowledge could be scientific, or experiential as identified in the survey study [6]. Experiential knowledge can be represented in logical rules, knowledge graphs, and probabilistic relationships as surveyed in the [6] and integrated at any different steps in the deep learning pipeline–Data-level, architecture-level, training-level, and decision-level. In the training-level integration, the model can be regularized with regularization terms or loss terms derived from the knowledge. [44] introduced a novel methodology for adding symbolic knowledge at the training level in the form of a special loss function to regularize or constrain the model. The symbolic knowledge is in the form of propositional logic and can be as simple as *exactly-one constraint* in the case of classification and as complex as *structured output prediction constraint* in the applications like preference ranking, sub-graphs, or paths as demonstrated in [44]. Conceptually, this proposed loss function is based on a semantic similarity measure between the neural network's output i.e., class prediction (vector of probabilities) $p = [p_1, ..., p_n]$ and symbolic knowledge in the form of a logical sentence α consists of variables $X = \{X_1, ..., X_n\}$ where n is a number of classes in a classification problem.

$$loss = existing\ loss + w \cdot semantic\ loss \tag{2.3}$$

where w is the weight that encodes the degree of importance attached to the semantic loss L^S. Formally semantic loss can be defined as follows.

$$L^S(\alpha, p) \propto -log \sum_{x \models \alpha} \prod_{i:x \models\ X_i} p_i \prod_{i:x \models \neq X_i} (1 - p_i) \tag{2.4}$$

A semantic loss function that is typically used for classification problems is to enforce a *exactly-one constraint*. That is, we want each example to belong to exactly one class (out of all possible classes). Thus, given unlabeled examples, the model is forced to assign each example to a class to minimize semantic loss.

Chapter 3

Related Work

The importance sampling based approach proposed in [13] is quite general since it can be applied to any DGM. Specifically, it estimates importance weights (ratio between distributions) through a binary probabilistic classifier. Similar ideas have also been explored in other DGMs. For instance, in GANs [11], the discriminator acts as a classifier that tries to classify samples from the generator with samples from the real data. Thus, it learns parameters for the generator based on the classifier learned in the discriminator. A similar idea has also been applied in contrastive learning [15]. Importance sampling has also been used to identify the key samples in a dataset. In [21, 7], importance sampling was used to weight points in the dataset to reduce bias in training. Further, importance sampling has also been used to scale up deep learning. For instance, Katharopoulos et al. [21] proposed to use of importance sampling to identify key data instances that are more beneficial when training the model. Thus, instead of training the model over all instances, we could train the model over a smaller subset with similar results. To estimate these weights, Katharopoulos et al. used a pre-sample of the points and estimated the effects of the pre-sample on the gradients when training the model. Other well-known sampling approaches can also be applied to DGM sampling as well. For instance, an MCMC sampler was proposed in [42] for sampling from DGMs. However, MCMC samplers usually are slower in terms of convergence. Another approach is to use rejection sampling which was explored in [1]. However, the use of a classifier to compute importance weights makes the approach in [13] much more general.

The use of symbolic knowledge within deep networks is a fast-growing area of research under the umbrella of *Neuro-Symbolic AI* [10]. In general, deep neural networks are augmented with symbolic knowledge to address their limitations of strong reliance on data, limited interpretability i.e., lack of reasoning, weak generalization, etc. In Neuro-symbolic AI, experiential knowledge (common knowledge gleaned from longtime observations) in the form of symbolic AI such as logic rules, knowledge graphs, or probabilistic dependencies is encoded into deep neural networks at various stages as constraints. Further, the same can also help add specific domain knowledge. For instance, in physics-informed deep learning [6], deep neural networks are guided by theoretical scientific knowledge present in the form of well-established laws pertaining to domains that control how target variables in data behave.

In the Neuro-Symbolic AI paradigm, deep learning techniques enforce arithmetic constraints [34, 28] or logical constraints [35, 18] on neural network outputs by reducing them into differentiable functions. Other methods, such as encoding logic into a factor graph[30], face issues with complex logical constraints. Additionally, some methods use real-world label structures with entropy penalty terms[12]. The semantic loss function proposed in [44] is a fairly general approach that can be used to encode a variety of different constraints into deep network learning. Here, we apply the semantic loss function to add constraints on data from generative models which helps us learn more effective classifiers.

Chapter 4

Importance Sampling for Deep Generative Models

Bias in DGMs

In DGMs, the goal is to learn a model from which we can obtain samples similar to those obtained when we sample from the data distribution. Specifically, suppose p represents the data distribution, the DGM will approximate p_{data} by learning p_θ (where θ represents the model parameters learned by the DGM). Thus, the closer p_θ is to p_{data}, we can obtain samples from the DGM that are similar to samples in the data. This way, we can in theory, have access to unlimited number of samples using the learned DGM. However, since p_θ is not the same as p_{data}, there is a bias in the samples generated by the DGM. By reducing this bias, we can obtain more realistic samples from the DGM, i.e., samples that reflect the true data distribution.

The approach proposed in [13] uses the idea of importance sampling (IS) to weight the samples generated by the DGM which can in turn be used to reduce bias in any downstream task where these samples are used. Specifically, in IS, we compute the ratio between the probability of a sample in true distribution to its probability in the proposal distribution. Similarly, here, suppose we treat the distribution learned by the DGM (p_θ) as the proposal distribution, we need to compute the ratio $p_{data}(\mathbf{x})/p_\theta(\mathbf{x})$ for a given sample $\mathbf{x}$. Next, we discuss a use-case where using this ratio, we can improve a task. Specifically, consider a model-based data augmentation. That is, data is generated by a generative model which can be useful to enrich the dataset to provide sufficient training data for a classification or regression task. The optimization problem in the task of learning a classifier is to minimize expectation of loss over the observed dataset. We define some notation as follows. $f(\mathbf{x})$ is the loss used in the task, e.g., cross entropy loss. $\mathbf{E}_{p_{data}}[f(\mathbf{x})]$ is the expectation of the loss under the data distribution. By using samples from the DGM, we introduce a bias in the training since $\mathbf{E}_{p_{data}}[f(\mathbf{x})] \neq \mathbf{E}_{p_\theta}[f(\mathbf{x})]$

When we augment the data with samples from the DGM, we are essentially sampling from a distribution that is different from the target distribution. Using IS, we can weight the samples using their importance weights. Thus, the expectation can be rewritten as $\mathbf{E}_{p_\theta}[w(\mathbf{x})f(\mathbf{x})]$ where $w(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_\theta(\mathbf{x})}$. In practice, we approximate the expected value with a Monte Carlo estimate. Specifically, suppose we estimate the expectation using T samples, we have the following estimator.

$$\hat{\mathbf{E}}_{p_\theta}[f(\mathbf{x})] = \frac{1}{T}\sum_{i=1}^{T} w(\mathbf{x}_i)f(\mathbf{x}_i)$$

We can show that the expectation that is approximated using the samples from p_θ is asymptotically unbiased. That is, as the number of samples $T \to \infty$, $\hat{\mathbf{E}}_{p_\theta}[f(\mathbf{x})]$ approaches $\mathbf{E}_{p_{data}}[f(\mathbf{x})]$. Thus, by using IS, we can reduce the bias of samples generated by the DGM.

Computing the Importance Weights

While IS provides a general framework for weighting the samples, it turns out that applying it to DGMs is not as straight-forward. Specifically, the main issue is that it is often infeasible to compute $w(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_\theta(\mathbf{x})}$. This is because, in the typical case, the actual data distribution does not have a specific form (e.g., Gaussian, etc.). Further, even the DGM distribution p_θ is often intractable to compute. Therefore, in [13], inspired by techniques that compute density ratios using classifiers [39], the idea is to compute the importance weights by converting it into a classification task.

The main idea is that instead of explicitly computing the probabilities to compute the importance weights, we can estimate the weights based only on samples from the DGM as well as samples from the real data. Such an approach to compute weights is called *likelihood-free* since we do not compute the probabilities explicitly. To estimate the weights, we formulate a classification problem as follows. We want to learn a binary classifier that classifies a sample from the real data as 1 and a sample from the DGM as 0. More specifically, we learn a probabilistic binary classifier that outputs the probability of belonging to the positive class, i.e., the input sample is from the real dataset. Suppose, $c_\phi(\mathbf{x})$ denotes the probability output by the classifier (with parameters ϕ) that the sample $\mathbf{x}$ is from p_{data}, the importance weight can now be approximated as follows.

$$w(x) = \frac{c_\phi(\mathbf{x})}{1 - c_\phi(\mathbf{x})}$$

Note that the classifier used to estimate $c_\phi(\mathbf{x})$ should output accurate probabilities. That is, the classifier needs to be a *well-calibrated* classifier. This means that if $c_\phi(\cdot)$ outputs a probability p when roughly $p\%$ of the training data used to learn c_ϕ belongs to the positive class, i.e., in this case they correspond to real samples from the dataset. To learn such a calibrated classifier, we can once again use a neural network. It is shown in [31] that a neural network with a single hidden layer is well-calibrated [14]. Thus, we can train a single hidden layer neural network over a balanced dataset that consists of both real and samples generated by the DGM. We can then compute the importance weights of a new sample generated by the DGM by predicting its probability by the trained neural network. Naturally, the type of samples generated by the DGM has a major influence on the quality of the probabilistic classifier we learn which in turn influences the bias. That is, if the probabilistic classifier produces accurate probabilities, then the importance weights learned are optimal and therefore any downstream task that uses these weights have smaller bias and vice-versa.

Importance Resampling using the Gumbel Approximation

Sample Importance Resampling [26, 9] (SIR) is a technique where the importance weighted samples are resampled according to their normalized importance weights. Specifically, suppose the DGM generates samples $\mathbf{x}_1 \ldots \mathbf{x}_n$, i.e., we assume that these samples are generated from the model distribution p_θ. Let $w(\mathbf{x}_1)$, $w(\mathbf{x}_2) \ldots w(\mathbf{x}_n)$ be the importance weights computed by the probabilistic classifier over the samples $\mathbf{x}_1 \ldots \mathbf{x}_n$. We now resample $\mathbf{x}_i$ with probability proportional to $w(\mathbf{x}_i)/Z$, where $Z = \sum_{i=1}^{n} w(\mathbf{x}_i)$. Thus, the weighted samples are drawn from a multinomial distribution based on their importance weights. If we compute the probability density over the resampled data, we have the following:

$$\hat{p}_\theta(\mathbf{x}) \propto p_\theta(\mathbf{x}) w(\mathbf{x})$$

In [13], it is shown that the KL-divergence between $\hat{p}_\theta(\mathbf{x})$ and the data distribution $p_{data}(\mathbf{x})$ is smaller than that between the original DGM distribution $p_\theta(\mathbf{x})$ and $p_{data}(\mathbf{x})$. In other words, SIR helps us obtain a better fit for the data distribution as compared to using the DGM generated samples directly.

To implement SIR, we have different options. The basic approach also known as roulette sampling is implemented as follows. We compute the cumulative sum of the probabilities for the samples (since we are dealing with very small numbers, these are computed in log-space). Thus, the cumulative weight for the i-th sample is given by $\sum_{j=1}^{i} w(\mathbf{x}_j)/Z$. We now want to draw a sample from this cumulative distribution based on a uniformly distributed random number. That is, if we draw a random number U and this number lies in the interval between the cumulative probabilities that are computed for samples $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$, we sample the data instance $\mathbf{x}_{i+1}$. To search for such an interval, we can use binary search for improving efficiency. However, with this approach, the samples depend upon the random number generator. Further, since in our case, it turns out that the numbers are very small, they need to be maintained in log-space to avoid underflow. When we take sums in log-space, the resulting sum (in log-space) is an approximation that results in errors when computing the cumulative distribution. Thus, the samples generated from this approach do not closely correspond to the true distribution that we need to sample from. Instead, we utilize another approach to sample from the importance distribution based on the Gumbel approximation.

The Gumbel-softmax function is used to sample from a Multinomial distribution with log probabilities. Specifically, note that when we sample from the distribution, for each sample drawn, we are essentially computing the following function.

$$\max_i \left(\left(\sum_{j=1}^{i-1} w(\mathbf{x}_j/Z) \right) \leq U \right)$$

That is, we are computing the index of the instance to sample based on the generated random number U. The Gumbel-softmax function approximates this using a continuous function known as the Gumbel function. Specifically, the standard Gumbel distribution is given by the following equation.

$$Gumbel(0,1) = \exp^{-(x+\exp(-x))}$$

Using $Gumbel(0,1)$, we can now approximate the sampling as follows.

$$\max_i (Gumbel(0,1) + log(w(\mathbf{x}_i/Z))$$

Next, we can approximate the max in the above function with a smooth, differentiable approximation that is commonly used, called as softmax. Thus, the Gumbel-softmax probability approximation is given as follows:

$$\hat{p}(\mathbf{x}_i) = \frac{\exp(Gumbel(0,1) + log(w_i))/\tau}{\sum_j \exp(Gumbel(0,1) + log(w_j))/\tau}$$

where w_i is an abbreviation for $w(\mathbf{x}_i/Z)$ and τ is called as a temperature parameter. As $\tau \to 0$, the softmax approximation more smoothly approximates the max function. The *Gumbel trick* is in fact widely used to sample from discrete multinomial distributions in Neural Networks. For example, for a neural network that generates language, the network typically learns a discrete distribution over words. To sample the words to generate, we need to sample from this distribution and the Gumbel trick helps us rewrite this as a continuous discrete distribution which is differentiable [32]. This is a necessary condition for neural network learning algorithms such as backpropagation. Similarly, in reinforcement learning, when we want to sample an action from a set of discrete actions, once again the Gumbel trick is used to perform this sampling [19]. In our case, we use this to sample from the log importance distribution. Thus, once we compute the importance weights for the samples generated by the DGM, we reparameterize the importance weights as a Gumbel-softmax probability and then sample from this new distribution. The samples from this distribution constitute the resampled data for the DGM.

Improving Downstream Classification with Generated Data

The main idea is to use generated images as unlabeled examples to improve classifiers. Specifically, as is shown in [44], unlabeled examples can help guide the decision boundary of the classifier to improve generalization. Thus, in cases where labeled examples are limited/expensive but unlabeled examples are cheap, we can provide the model with a large number of unlabeled examples along with a smaller number of labeled examples (semi-supervised classification). Here, we use the semantic-loss function to add constraints into the deep model based on the unlabeled examples. Specifically, we use the exactly-one constraint where each unlabeled example is to be assigned a single class. Thus, the decision boundary of the classifier using the semantic-loss will be optimized to both correctly classify the labeled examples and also to adapt itself to the structure within the unlabeled examples resulting in better generalization. However, it is important to choose informative unlabeled examples on which we add the exactly-one constraints in the semantic loss function. To do this, we use the importance-sampled images since these correspond to the ones that have most information (according to the probabilistic classifier) and add exactly-one constraints for all these resampled images.

The block diagram of our architecture is shown in 4.1. Specifically, we implement a ResNet-based [16] semi-supervised multi-class classifier consisting of several key components.

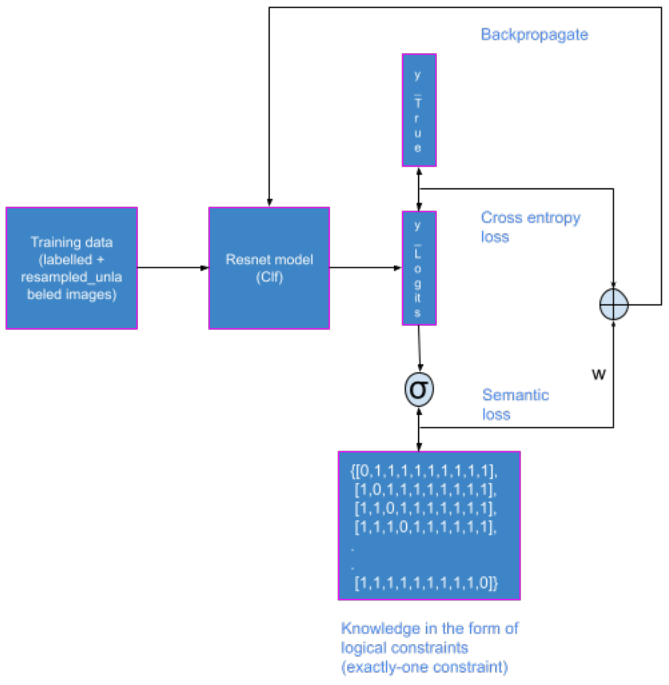

Figure 4.1: ResNet-based semi-supervised multi-class classifier for labeled and unlabeled_*resampled* CIFAR10 data using *Semantic loss function*

The input to the network is a set of labeled and unlabeled images from the image dataset and resampled images drawn from importance sampling framework respectively, which are passed through a series of convolutional layers to extract features. The ResNet architecture is used to enable deeper networks with residual connections, which can help alleviate the problem of vanishing gradients in deep networks. The standard supervised loss function; *Cross entropy loss* has used for the labeled data, and an augmented loss function i.e., *Semantic loss* that incorporates the unlabeled data. The semantic loss represents how close output probabilities–normalized logits (applied Sigmoid activation function)–are to satisfying the constraints. Here, constraints are set of one hot encoded binary variables. Further, semantic loss is having associated weight of $w = 0.0005$, best suggested through hyperparameter tuning experiments in [44]. To update the model parameters, we use backpropagation through the entire network, including the augmented loss and constraint terms. This is done using the Adam optimizer.

Chapter 5

Implementation

Implementation

Another goal of this thesis is to investigate the effectiveness of importance sampling on more complex datasets than MNIST, so we chose the CIFAR10 dataset. It contains color images of various objects, has greater variation in terms of object appearance, color, and orientation, and the dataset includes 10 different classes of objects (such as airplanes, horses, dogs, ships, etc). We have used a Python development environment for implementation. The models are built using the machine learning framework Pytorch version >=1.1.0, and TensorFlow version 1.14.0. Pytorch was chosen for this thesis due to its ease of use, functionalities, flexibility, and strong community support. Due to the requirement of computational resources, all experiments are conducted in Google's Colab pro+ [3] on the Google Cloud Platform for hardware acceleration. Standard NVIDIA GPUs like P100 and V100 get assigned dynamically and recently they have introduced premium GPU A100 which is more powerful. For experiment tracking and logging, we have used a WandB [2] tool, which has a simple interface for visualization artifacts, logging output, system utilization, tracking of all runs, etc. For efficient storing and reading images into deep neural networks, Pytorch supports abstract *Dataset* and *Dataloader* class respectively to inherit later. In our case, we had to generate many samples of size $150K \cdot 3 \cdot 32 \cdot 32$ from each deep generative model and save them to an NPZ (numpy zipfile) file. Due to the non-trivial dataset, we had to inherit these abstract dataset classes to concatenate real (CIFAR10) and generated data from different files, and apply transformations. Next, according to our needs, we had to engineer a *Dataloader* abstract class (which efficiently iterates different batches in every epoch) to create a customized batch of half of the real data and half of the generated data by making sure real data is balanced, and this process repeats for varying sizes of real data e.g., 1000, 4000 as shown in the table 6.4. Moreover, We implemented knowledge-augmented *Semantic loss function*[1] in Pytorch in the functional programming style. The original codebase has a few missing parts, to our knowledge we have engineered and streamlined an end-to-end workflow from implementing deep generative models to generate a large set of samples, integrating tools like WandB, implementing the SIR algorithm using a Gumbel Max Trick to analyze the methodology in a detailed manner for the first time. Our code is available on GitHub at `https://github.com/jprachir/resampling_dgms.git`

[1]Originally written in Tensorflow for different dataset at https://github.com/UCLA-StarAI/Semantic-Loss.git

Chapter 6

Experiments

Experiment Setup

Our goal is to empirically evaluate the utility of importance sampling in DGMs that generate image data. Specifically, we use one of the most widely-used benchmarks called CIFAR10 [23] in our evaluation. CIFAR10 is a labeled, colored image dataset, consisting of 60000 images in total. The dataset consists of 10 classes and each image is labeled with its class. Each class has 10,000 images. Fig. 6.1 is a grid consisting of example CIFAR10 images. The dimension for each image is 32x32x3 (height x width x channels). The train/val/test splitting proportion of data is 45k/5k/10k.

We use two different DGMs in our evaluation. The first is a convolutional VAE. Here, the architecture consists of an encoder and a decoder. The encoder has three convolutional layers and generates a latent vector of size 180 dimensions. The latent vectors are distributed according to a Gaussian distribution. The decoder samples a latent vector and decodes it back into the image through deconvolutional layers. We train the VAE on the training portion of the CIFAR10 dataset for 500 epochs on the Google Colab+ environment with a single GPU. To generate images, we sample the latent vectors from the standard Gaussian distribution and then use the decoder to generate the images according to the distribution learned by the VAE.

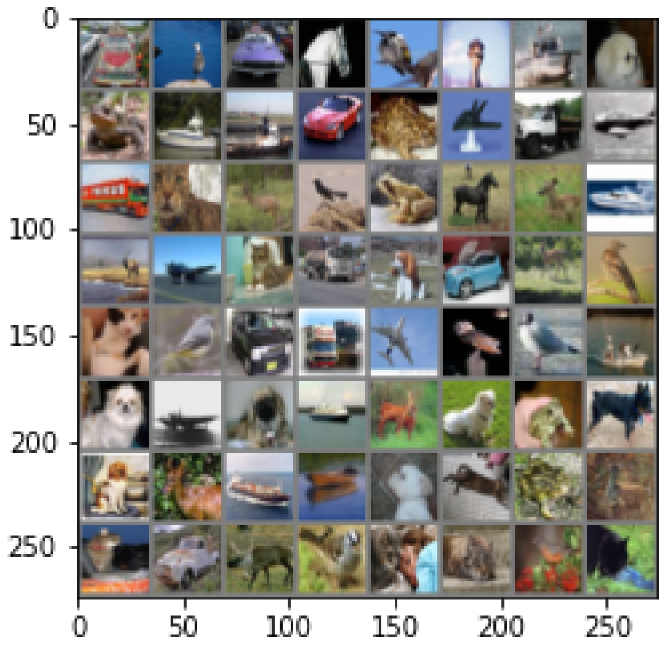

Figure 6.1: Grid of real samples

The next DGM we use in our evaluation is PixelCNN++ [38], which is a state-of-the-art open-source model; its implementation is based on the PixelCNN autoregressive model for image generation. Here, since training the network is considerably expensive even with a GPU, we use a pre-trained network. The pre-trained model[1] of PixelCNN++ is taken at epoch 889, learning rate 0.0004, 160 filters used across the model, and 5 blocks of residual blocks at each stage of the PixelCNN++ model. In PixelCNN++, the images are generated pixel-by-pixel.

Image Generation

We show examples of the images generated by VAE and PixelCNN++ in Fig. 6.2 and 6.3 respectively. Note that, the quality of images generated by PixelCNN++ is far superior to those generated by VAE. In general, it is a known drawback of VAEs that they generate images that are quite blurry as can be observed in the examples shown in the figures. Table 6.1 showcases various

[1]https://github.com/pclucas14/pixel-cnn-pp

Table 6.1: Performance comparison on CIFAR10

Model	Data	Training Time	generation(250 samples) time	NOP	GPU
VAE	Cifar10	63mins	0.7mins	<2M	T4
PixelCNN++	Cifar10	Pretrained	57mins	<54M	P100

elements of deep generative models. Each model was trained on CIFAR10 dataset with respective training times. The sampling time required to generate a batch of 250 samples has been provided. Clearly, the sampling process for VAE is faster than PixelCNN++ since in PixelCNN++ inference is done pixel by pixel. NOP is an abbreviation for the total number of parameters to be trained to draw inferences in each respective model. Inference is done using different accelerators due to the dynamic allocation of the Google cloud platform (they both have the same configurations).

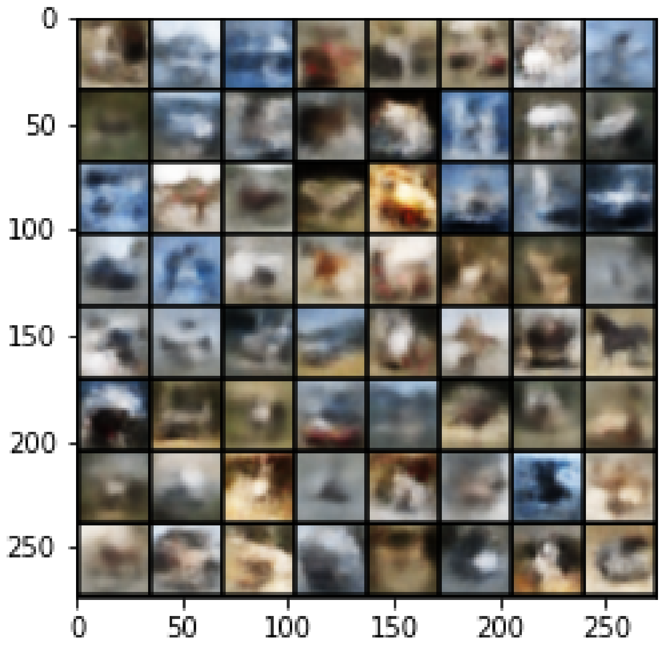

Figure 6.2: Grid of VAE generated samples

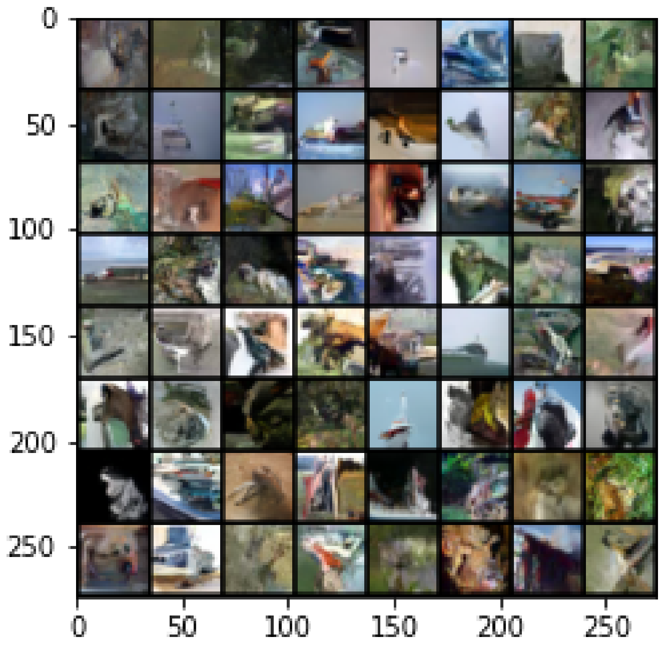

Figure 6.3: Grid of PixelCNN++ generated samples

Importance Weights

We use a calibrated probabilistic classifier to compute the importance weights for the generated images. To do this, first, we extract features for images using a well-known pre-trained model called as the *Inception V3* model [40]. On top of the extracted features, there is a dense layer of size 1000 nodes which act as the hidden layer for the classifier. Finally, this is connected to a single output (binary classifier) which is trained using the cross-entropy loss. To train the model, we use a balanced dataset, i.e., we have 45K image samples from the real dataset and 45K generated samples. Further, we added a random image baseline just as a sanity check. Specifically, instead of training the classifier with generated images from VAE/PixelCNN, we just train it with random grayscale images. The accuracy results of training the classifier are shown in Table 6.2. As expected, the classification accuracy is higher for VAEs since the images are blurrier than real images. Thus, the probabilistic classifier can distinguish between real and generated images quite easily. On the other hand, PixelCNN++ produces very realistic images. Therefore, when the classifier tries to distinguish between real images and those generated through PixelCNN++, it has much lower accuracy.

Table 6.2: Various performance metrics of a classifier

Model	Data	training acc	testing acc	training time
VAE	Cifar10	99.33%	99.89%	66mins
PixelCNN++	Cifar10	80.92%	84.46%	69mins

Since our goal is to estimate importance weights through the probabilities of the classifier, more important than the accuracy of the classifier, we want to know how well-calibrated is the classifier. That is, does it produce probabilities that reflect the true probabilities since only then will our resulting importance weights be more accurate. To analyze this, we plot calibration curves. In these curves, we measure the average probability output by the classifier as we evaluate on varying proportions of instances belonging to the positive class (in our case, this corresponds to images from the real dataset). Thus, if we use $p\%$ of the data in our evaluation to be the real images, then the average probability output by a well-calibrated classifier over all instances used in the evaluation is equal to p. We use a total of 20K images (10K real images and 10K generated images) to plot the calibration curve shown in Fig. 6.4. The x-axis shows the proportion of positive samples and the y-axis shows the average probability. As shown here, for the perfectly-calibrated classifier, we should obtain a straight line (shown as a dotted line) since the average probability is exactly equal to the proportion of positive samples. As seen from our results, the classifier trained on real and PixelCNN++ samples (blue curve) in fig 6.4 indicates that it's a well-calibrated model as visually it's close to the diagonal. An orange curve for a classifier trained on real and VAE-generated samples is quite not as consistent and has a worse calibration. The Green curve is for a model trained on random noise that is obviously far from the diagonal since it is poorly calibrated. Thus, in summary, training with the PixelCNN++ samples gave us the most calibrated classifier to estimate the importance weights. The Gumbel approximated weights are visualized in Fig. 6.5. As shown here, the distribution of weights has a similar shape for both VAEs and pixelCNN++.

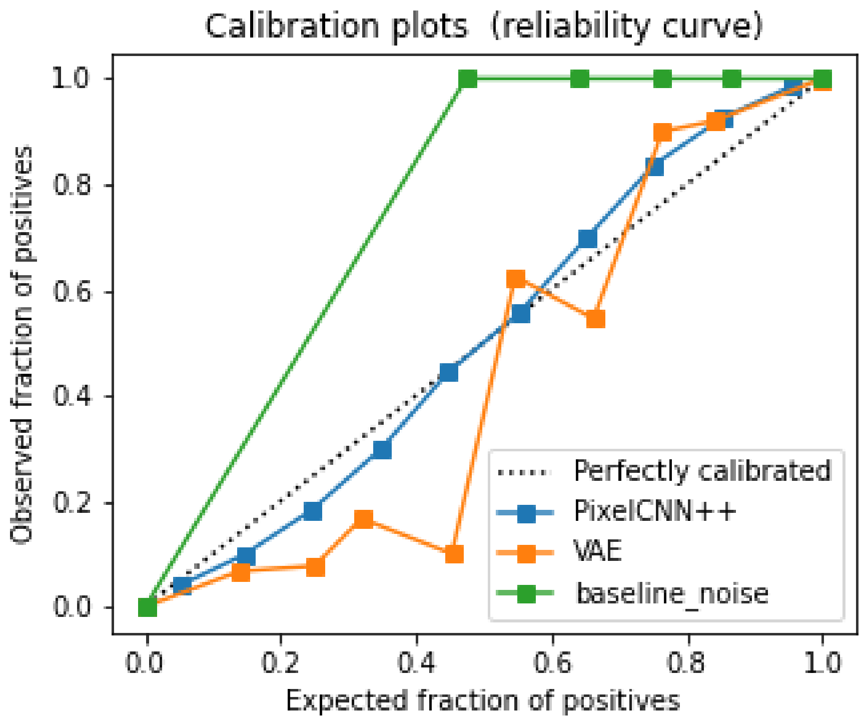

Figure 6.4: Calibration plot

Evaluating Generated Samples

To evaluate the generated samples, we use three standard metrics that measure the quality of the generated samples, Inception Score (IS), Frechet Inception Distance (FID), and Kernel Inception Distance (KID). To compute these metrics, we use the Tensorflow implementation of the Inception V3 Network. All images are transformed i.e. resized to 299x299 to the size of the original training data of the Inception model. The standard errors for each score are computed for 100K DGM-generated data over 10 runs. In each of these metrics, the generated images are passed through the inception V3 network to extract features. Then, the images are classified based on labels in the inception network. The idea is that if the generated images are of high quality, then the inception network should predict the labels more confidently over the generated images.

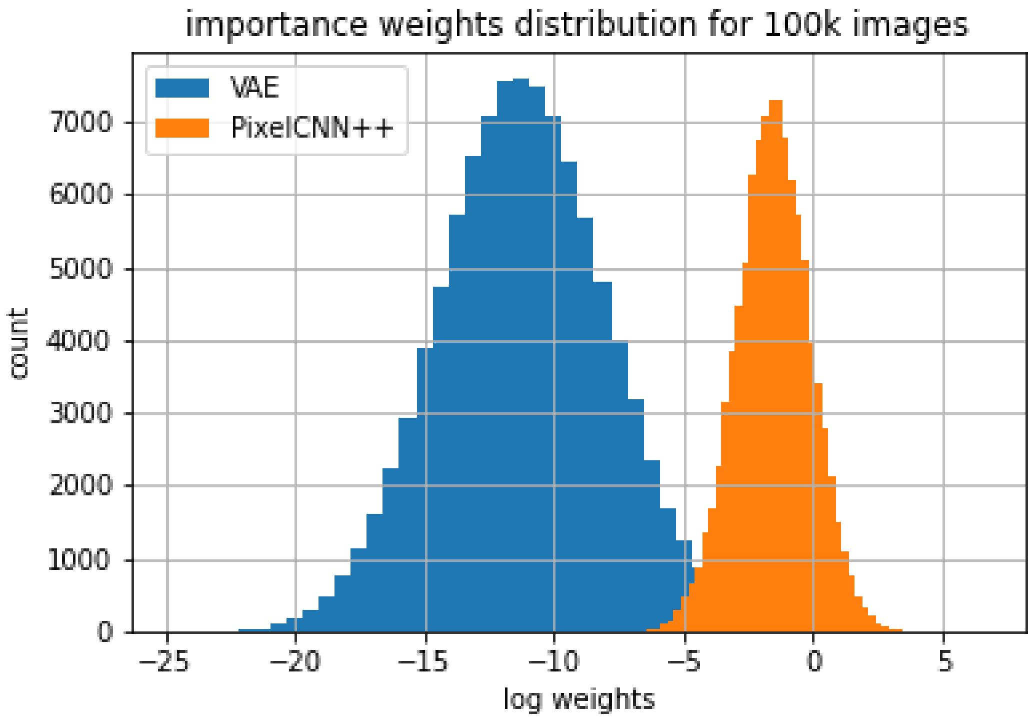

Figure 6.5: weights distribution

In terms of the metrics, better-quality images have higher IS and lower FID/KID scores. Table 6.3 shows the results over different metrics comparing the images generated by VAE/PixelCNN++ with images resampled using the importance distribution with the Gumbel approximation. In the table, the Reference value indicates the best possible scores that can be achieved, i.e., these are computed using the images from the real data (the true data distribution). As shown from the results, in both VAE and PixelCNN++, importance weights based resampling has a significant impact on the quality of images. Overall, since PixelCNN++ generates better quality images, they have better scores over all metrics. However, as it turns out, the resampling helps improve the quality of VAEs by a relatively large margin (as seen in the massive decrease in the FID value). Thus our experimental results suggest that resampling has a larger impact when the generated images are of lower quality. This is an interesting observation since, when the images are of lower quality, then naturally, it is more important that we select the right samples for a downstream task.

Table 6.3: Performance metrics to assess the quality of images generated by different generative models

Model	Evaluation	IS(↑)	FID(↓)	KID(↓)
-	Reference	11.16 ± 0.021	7.60 ± 0.017	0.010 ± 0.0001
PixelCNN++	No Importance Sampling	4.39 ± 0.0550	64.16 ± 1.4722	0.318 ± 0.0055
	Importance Sampling	**6.73** ± 0.0233	**42.95** ± 0.0320	**0.121** ± 0.0008
VAE	No Importance Sampling	1.14 ± 0.0285	418.11 ± 5.032	0.755 ± 0.0068
	Importance Sampling	**2.56** ± 0.2514	**301.09** ± 18.47	**0.330** ± 0.0005

Semantic-Loss Augmented Semi-Supervised Classification

We used the well-known ResNet (Residual Network) model[2] for performing image classification in CIFAR10. In a ResNet, a residual block contained multiple convolutional layers, batch normalization, and rectified linear unit (ReLU) activations. ResNet is trained using backpropagation, and Adam stochastic gradient descent. The semantic loss function specified in Eq. 2.4 is used to train our model. Other fixed hyperparameters include 20 epochs, a learning rate of 0.001, batch size of 32. Labeled data is always a balanced dataset with equal number of images for each of the 10 classes. On average each set took around 140 minutes to train the model. The accuracy is reported on the test data which has associated labels. Table 6.4 shows the performance measure of the classifier on the varying size of labeled and unlabeled data for different DGMs.

$D_{real} + D_{gen}$: model trained on CIFAR10 & generated data

$D_{real} + D_{gen} + IS$: model trained on CIFAR10 and importance resampled generated data

If we take 1K labeled and 45K unlabeled data (Importance resampled data) the accuracy improves by 1.26% for VAE and 1.41% for PixelCNN++ model against just generated 45K samples. Again, if we take 4K labeled and 42K unlabeled (Importance resampled data) the accuracy improves by 1.03% and 0.09% for VAE and PixelCNN++ models respectively. We compared with the baseline accuracy which is calculated over an entire 46K CIFAR10 labeled data. As seen from our results in 6.4, when we use SIR to resample the generated images (as our large set of unlabeled data) the classifier accuracy improves. This is true for both VAE and PixelCNN++ generated images. We

[2]Implementation referenced from https://pytorch-tutorial.readthedocs.io/en/latest/

noticed that the improvement is more pronounced in VAEs compared the PixelCNN++. This is because in VAEs, the quality of generated images is poor and therefore, it is important to choose the right type of generated images on which we add the exactly-one constraints in the semantic loss function. Thus, using importance sampling helps us obtain such images. On the other hand, PixelCNN++ generates very high quality images (but at a high computational cost compared to VAEs). Therefore, it is less important to add constraints on the most informative images. Thus, we can envision this approach as being useful in cases where we have access to a large number of lower quality generated images from an inexpensive generative model and we want to boost the downstream classifier for which limited labeled examples are available using the generated images. In future, we can perform more studies in real applications that have these settings.

Table 6.4: Test accuracy measure of an augmented semi-supervised multi-class classifier

Model	Dataset	No. of used labels		
		1000	4000	ALL
VAE	$D_{real} + D_{gen}$	66.69	73.65	81.48
	$D_{real} + D_{gen} + IS$	**67.53**	**74.41**	
PixelCNN++	$D_{real} + D_{gen}$	68.03	75.55	
	$D_{real} + D_{gen} + IS$	**68.99**	**75.62**	

Chapter 7

Future Work

A potential future direction is to extend this work to flow-based and energy-based generative models. Further, we could also explore other more advanced approaches to estimating the importance weights (apart from the probabilistic classifier). The proposed approach can also be used in applications such as Multimodal AI (Visual Question Answering, Image Captioning, etc.) to sample images for training more efficiently. More generally, the idea of importance sampling can also be used to scale up training.

Another possible future direction is to use importance weights to explain model behavior (explainable AI). Specifically, we can understand which images are critical to training the model based on the importance weights which will also offer insights into how the model works. We can also extend the framework for semi-supervised learning by adding more complex constraints during training. For instance, constraints that are related to a domain such as images of X-rays for disease detection. Finally, we can extend this work to other types of data including language datasets and graph datasets.

Chapter 8

Conclusion

Deep generative models are capable of generating realistic and diverse samples, allowing for tasks such as data augmentation, semi-supervised learning, and unsupervised feature learning among the most important ones. Deep generative modeling learns an approximation of the data distribution that captures the main characteristics of the data and generates new samples that are similar to the real data. However, the quality of generated samples affects because samples do not represent the full distribution of the real data.

In addition to importance sampling, other sampling techniques like Markov chain Monte Carlo, and rejection sampling have been employed to perform an extra operation on the generated sample like transforming or rejecting it to ensure the sample's quality. However, these methods demand high computational power and are slower to converge. And other methods focuses on learning the model parameters unlike the following case, where the quality of samples is quantified in the form of bias in the model's distribution. [13] proposed a bias reduction framework that uses the standard approach of importance sampling to generate weighted samples instead of unweighted samples. Essentially, this weight for each sample encodes the related importance with respect to the true distribution. Weight is estimated by using a shallow, dense neural network making sure it's a calibrated one.

In this thesis, we analyzed and compared the importance weights learned by the calibrated probabilistic classifier for two different DGMs specifically VAEs and PixelCNN++. Next, we deployed a Gumbel softmax sampling trick to efficiently perform resampling, based on the importance weight distribution. We developed an open-source implementation in Pytorch of the probabilistic classifier architecture. Measures of better quality samples like *Inception score, FID, and KID* suggested improved scores for both methods. In addition to that we demonstrated improved performance of the augmented semi-supervised multi-class classifier with the semantic loss for simple constraint when we select images through importance sampling.

In essence, the workflow of importance sampling followed by sampling importance resampling using the softmax Gumbel trick provides us with better quality samples and is generally applicable to any DGM. As a result, we believe this work can serve as the foundation for future work validating performance in different domains e.g., healthcare where labeled data is limited.

References

[1] Samaneh Azadi et al. “Discriminator Rejection Sampling.” In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=S1GkToR5tm.

[2] Lukas Biewald. *Experiment Tracking with Weights and Biases*. 2020. URL: https://www.wandb.com/.

[3] Ekaba Bisong. “Google Colaboratory.” In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 59–64. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_7. URL: https://doi.org/10.1007/978-1-4842-4470-8_7.

[4] Mikołaj Bińkowski et al. “Demystifying MMD GANs.” In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=r1lUOzWCW.

[5] Samuel R. Bowman et al. “Generating Sentences from a Continuous Space.” In: *Conference on Computational Natural Language Learning*. 2015.

[6] Zijun Cui et al. “Knowledge-augmented Deep Learning and Its Applications: A Survey.” In: *CoRR* abs/2212.00017 (2022). DOI: 10.48550/arXiv.2212.00017. arXiv: 2212.00017. URL: https://doi.org/10.48550/arXiv.2212.00017.

[7] Maurice Diesendruck et al. “Importance weighted generative networks.” In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2019, pp. 249–265.

[8] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using Real NVP.” In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=HkpbnH9lx.

[9] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. “On sequential Monte Carlo sampling methods for Bayesian filtering.” In: *Statistics and computing* 10.3 (2000), pp. 197–208.

[10] Artur S. d’Avila Garcez and Luís C. Lamb. “Neurosymbolic AI: The 3rd Wave.” In: *CoRR* abs/2012.05876 (2020). arXiv: 2012.05876. URL: https://arxiv.org/abs/2012.05876.

[11] Ian Goodfellow et al. “Generative Adversarial Nets.” In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[12] Yves Grandvalet and Yoshua Bengio. “Semi-supervised Learning by Entropy Minimization.” In: *Advances in Neural Information Processing Systems*. Ed. by L. Saul, Y. Weiss, and L. Bottou. Vol. 17. MIT Press, 2004. URL: https://proceedings.neurips.cc/paper_files/paper/2004/file/96f2b50b5d3613adf9c27049b2a888c7-Paper.pdf.

[13] Aditya Grover and Jiaming Song. “Bias Correction of Learned Generative Models using Likelihood-Free Importance Weighting.” In: *Advances in Neural Information Processing Systems*. 2019.

[14] Chuan Guo et al. “On Calibration of Modern Neural Networks.” In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and

Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1321–1330. URL: `https://proceedings.mlr.press/v70/guo17a.html`.

[15] Michael U Gutmann and Aapo Hyvärinen. "Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics." In: *Journal of machine learning research* 13.2 (2012).

[16] Kaiming He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[17] Martin Heusel et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium." In: *NIPS*. 2017.

[18] Zhiting Hu et al. "Harnessing Deep Neural Networks with Logic Rules." In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 2410–2420. DOI: `10.18653/v1/P16-1228`. URL: `https://aclanthology.org/P16-1228`.

[19] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical Reparameterization with Gumbel-Softmax." In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: `https://openreview.net/forum?id=rkE3y85ee`.

[20] Nal Kalchbrenner et al. "Video pixel networks." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1771–1779.

[21] Angelos Katharopoulos and Francois Fleuret. "Not All Samples Are Created Equal: Deep Learning with Importance Sampling." In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 2525–2534. URL: `https://proceedings.mlr.press/v80/katharopoulos18a.html`.

[22] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes.” In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: `http://arxiv.org/abs/1312.6114`.

[23] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “CIFAR-10 (Canadian Institute for Advanced Research).” In: (). URL: `http://www.cs.toronto.edu/~kriz/cifar.html`.

[24] cse lab.ethz.ch. *Sampling Importance Resampling (SIR)*. 2019. URL: `https://www.cse-lab.ethz.ch/wp-content/uploads/2019/03/hpcse2-19_tmcmc.pdf`.

[25] Jun S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Publishing Company, Incorporated, 2008. ISBN: 0387763694.

[26] Jun S Liu and Rong Chen. “Sequential Monte Carlo methods for dynamic systems.” In: *Journal of the American statistical association* 93.443 (1998), pp. 1032–1044.

[27] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables.” In: *International Conference on Learning Representations*. 2017. URL: `https://openreview.net/forum?id=S1jE5L5gl`.

[28] Pablo Márquez-Neila, Mathieu Salzmann, and Pascal Fua. “Imposing hard constraints on deep networks: Promises and limitations.” In: *arXiv preprint arXiv:1706.02025* (2017).

[29] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. “Which training methods for GANs do actually converge?” In: *International conference on machine learning*. PMLR. 2018, pp. 3481–3490.

[30] Jason Naradowsky and Sebastian Riedel. “Modeling exclusion with a differentiable factor graph constraint.” In: *ICML (Workshop Track)*. 2017.

[31] Alexandru Niculescu-Mizil and Rich Caruana. "Predicting good probabilities with supervised learning." In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 625–632.

[32] Weili Nie, Nina Narodytska, and Ankit Patel. "Relgan: Relational generative adversarial networks for text generation." In: *International conference on learning representations*. 2019.

[33] Augustus Odena, Christopher Olah, and Jonathon Shlens. "Conditional image synthesis with auxiliary classifier gans." In: *International conference on machine learning*. PMLR. 2017, pp. 2642–2651.

[34] Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell. "Constrained convolutional neural networks for weakly supervised segmentation." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1796–1804.

[35] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. "Injecting logical background knowledge into embeddings for relation extraction." In: *Proceedings of the 2015 conference of the north American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2015, pp. 1119–1129.

[36] Ruslan Salakhutdinov. "Learning deep generative models." In: *Annual Review of Statistics and Its Application* 2 (2015), pp. 361–385.

[37] Tim Salimans et al. "Improved Techniques for Training GANs." In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, 2234–2242. ISBN: 9781510838819.

[38] Tim Salimans et al. "PixelCNN++: A PixelCNN Implementation with Discretized Logistic Mixture Likelihood and Other Modifications." In: *ICLR*. 2017.

[39] Masashi Sugiyama, Taiji Suzuki, and Takafumi Kanamori. *Density ratio estimation in machine learning*. Cambridge University Press, 2012.

[40] Christian Szegedy et al. “Rethinking the inception architecture for computer vision.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[41] Jakub M Tomczak. *Deep Generative Modeling*. Springer Nature, 2022.

[42] Ryan Turner et al. “Metropolis-hastings generative adversarial networks.” In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6345–6353.

[43] Aäron van den Oord et al. “WaveNet: A Generative Model for Raw Audio.” In: *Proc. 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*. 2016, p. 125.

[44] Jingyi Xu et al. “A semantic loss function for deep learning with symbolic knowledge.” In: *International conference on machine learning*. PMLR. 2018, pp. 5502–5511.