Bayesian Bidirectional GAN

Julie Jiang
Tufts University
julie.jiang@tufts.edu

Daniel Dinjian
Tufts University
daniel.dinjian@tufts.edu

Abstract

Generative Adversarial Networks (GANs) are being recognized as a cutting-edge framework for modeling arbitrarily complex data signals, however GANs are generally only concerned with finding the best generator. Recently, Donahue et al. [6] created a Bidirectional GAN (BiGAN) that not only generates signals, but also encodes signals into their latent feature representations. This allows for a much more versatile range of applications, such as label classification, that use the feature representations as a clear, compressed form of the data. Our work further extends the BiGAN by combining it with the Bayesian GAN of Saatchi et al. [2] to construct a Bayesian Bidirectional GAN (BayesBiGAN). Our BayesBiGAN retains the uses of the BiGAN encoder but trains the GAN to converge to a posterior distribution instead of a single point. We demonstrate that our BayesBiGAN can capture a multimodal posterior which is better able to explore the underlying feature-representations of the relevant dataset.

1 Introduction

GANs are beginning to gain major traction in the deep learning community. They've been used for a whole host of purposes, ranging from standard face generation and image classification [3] to cooler style transfers turning pictures of apples into oranges or a still-life picture into a Picasso-esque painting [10]. But development in GANs has predominantly only considered new methods of moving from arbitrary latent space representations to interesting signal space representations. The Bidirectional GAN could be considered completing a half-finished product by providing the reverse mapping from signal space back to latent space [6].

BiGANs introduce an encoder that does the inverse of the generator, mapping sample space to latent space, allowing for unsupervised learning of latent feature representations. BayesGANs train multiple generators in tandem to form a multimodal posterior that avoids mode-collapse. Our work combines these two models into a BayesBiGAN which uses Bayesian networks in the BiGAN architecture. The main difference here will be that rather than networks in the BiGAN making point estimates, they will converge to multi-modal, expressive distributions. Therefore, our primary work will be to explore the quantitative and qualitative effects of expanding the BiGAN into a Bayesian form.

1.1 Hypothesis

Donahue et al.'s BiGAN [6] already achieves reasonable results in a range of metrics. We expect that incorporating the BayesGAN of Saatchi et al. [2] will allow the BiGAN to capture even more expressive power without compromising its numerical performance. BayesGAN models converge to highly multi-modal posterior distributions and avoid mode-collapse. In the BayesGAN paper [2] we see that the various modes of the posterior correspond to a diverse and compelling distribution of sample styles. By combining the BiGAN with the BayesGAN, we expect to have interesting new applications such as generating multiple reconstructions of an encoded image in the various styles of our multi-modal posterior.

Specifically, we will compare the One Nearest Neighbor classification accuracy of the standard and Bayesian BiGAN. One Nearest Neighbor is described as an "ideal metric for evaluating GANs"

Final project report for COMP-150 Bayesian Deep Learning, Fall 2018, Tufts University.

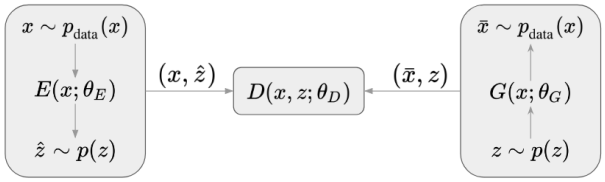


Figure 1: The Bayesian Bidirection GAN.

[8] and we will use it to evaluate our latent space. By predicting the label of a latent encoding to be the same as its nearest neighbor, we can evaluate the feature representations that our encoder is learning by verifying that similarly labeled data are being given similar encodings. We will also use our posterior generator to create multi-modal reconstructions of images and samples of our latent space. We'll visually compare these samples with the samples of the standard BiGAN to show the differences in expressive capabilities between the standard and Bayesian BiGAN.

2 Related Work

2.1 Bayesian GAN

Saatchi et al. [2] proposed to use a variant of Markov chain Monte Carlo (MCMC) sampling on a GAN in order to perform Bayesian inference on the weights of the GAN. Typically, GANs are prone to mode-collapses, which is the generator remembering a few good examples to sufficiently trick the discriminator. A major contribution of the Bayseian GAN is the ability to avoid mode-collapse by exploring a rich, multimodal distribution over the posterior distribution of the generator weights. Bayesian GANs can capture different *styles* of handwriting, whereas non-Bayesian deep convolutional GANs [3] always generates the same, generic style of output no matter how many different times the model was trained.

While MCMC inference is an indispensable tool to modern Bayesian frameworks, they tend to be computationally much more intensive than variational inference, a method to approximate the posterior via optimization [5]. In Bayesian GAN, Saatchi et al. [2] used Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) [4] to overcome the computational complexity of a traditional MCMC inference while also closely resembling the momentum-based optimizer SGD.

2.2 Bidirectional GAN

While the generator of a GAN can create signals from the latent space, GANs have no means of mapping a data sample from the signal space to its latent representation. Donahue et al. [6] and Dumoulin et al. [7] both independently proposed to jointly learn an encoder along with the generator and the discriminator in order to offer a mechanism to project the data to its latent representation. Due to the model setup, the encoder and generator will necessarily have to become the inverse of each other in order to reach an optimum. This framework has several important applications, especially in the field of representation learning. The encoder is an invaluable tool to automatically extract latent features from the data samples.

3 Bayesian Bidirectional GAN

Given a dataset $\mathcal{D} = \{x^i\}$ of variables $x^i \sim p_{\text{data}}(x^i)$, we wish to capture the data distribution $p_{\text{data}}(x^i)$. We will drop the superscript in x^i whenever it is clear that we are referring to one data sample. In our GAN training, in addition to training a generator and discriminator through sampling based inference as in Bayesian GAN [2], we train an encoder similar to the bidirectional GAN training scheme proposed in Donahue et al. [6]. The generator learns to produce *fake* data that is indistinguishable from the real data. The encoder learns to map real data from the data space to the latent space. Finally, the discriminator learns to distinguish the generated data and the real data using inputs from both the data space and the latent space. Donahue et al. [6] proved that while the encoder and the generator never directly interact with one another, they will learn to invert each other in order to reach an optimum.

In the context of Bayesian inference, the generator $G(z; \theta_g)$ is a mapping parameterized by the weight vector θ_g that transforms noise $z \sim p(z)$ to the data space, where the noise prior is the

uniform distribution from -1 to 1. The encoder $E(x;\theta_e)$, parameterized by θ_e, maps a data sample from the data space to the latent space. The discriminator $D(x,z;\theta_d)$, parameterized by θ_d, outputs the probability that the tuple (x,z) came from the real data distribution. The input tuple is either $(G(z),z)$ for a generated data sample, or $(x,E(x))$ for a real data.

The value function describing the game is given by:

$$\min_{G,E}\max_{D} V(D,G,E) = \mathbb{E}_{p_{\text{data}}(x)}[\log D(x,E[x])] + \mathbb{E}_{p(z)}[\log(1-D(G(z),z))] \tag{1}$$

The generator, discriminator and encoder can be any differentiable model, but they are usually convolutional deep neural networks [3]. To incorporate both (x,z), the discriminator takes data x as initial input and, at each linear layer thereafter, adds the latent representation z to the non-linearity through a linear transformation.

It is proved in Donahue et al. [6] that under optimal conditions, the encoder and generator should learn to invert each other, i.e. $x = G(E(x))$ and $z = E(G(z))$.

We place priors $p(\theta_g|\alpha_g)$, $p(\theta_d|\alpha_d)$, and $p(\theta_e|\alpha_e)$ on the generator, discriminator and encoder, respectively, where α_g, α_d and α_e are hyperparameters. All three priors are set to be a normal distribution with mean of zero and standard deviation of 0.02. We then employ SGHMC [4] to iteratively update the weights, which will be discussed later in this section.

Posteriors The generator learns to produce data that likely came from the data distribution, and the discriminator learns to discriminate between generated data and real data. The conditional posteriors are thus:

$$p(\theta_g|z,\theta_d) \propto \prod_{i=1}^{n_g} D\big(G(z^i;\theta_g),z^i;\theta_d\big)p(\theta_g|\alpha_g) \tag{2}$$

$$p(\theta_e|\theta_d) \propto \prod_{j=1}^{n_d} \Big(1 - D\big(x^j, E(x^j;\theta_e)\big)\Big)p(\theta_e|\alpha_e) \tag{3}$$

$$p(\theta_d|z,X,\theta_g,\theta_e) \propto \prod_{j=1}^{n_d} D\big(x^j,E(x^j;\theta_e);\theta_d\big) \prod_{i=1}^{n_g} \Big(1 - D\big(G(z^i;\theta_g),z^i;\theta_d\big)\Big)p(\theta_d|\alpha_d) \tag{4}$$

where n_g, n_e, and n_d are the number of mini-batch samples for the generator, encoder, and discriminator, respectively. In each equation the conditional posterior is proportional to the likelihood of the model times the prior.

Intuitively, the first term of Eq. 2 is the probability that the samples generated by G came from the real distribution. For the posterior over the encoder weights, the first term indicates discriminator's output of the probability that a data sample is *generated* . This is to induce the encoder to transform the real data sample to a latent space such that the discriminator believes the real data is actually fake. Lastly, the posterior over the discriminator weights is the discriminative classification likelihood.

Additionally, we marginalize the noise samples z from our posterior updates using Monte Carlo:

$$p(\theta_g|\theta_d) = \int p(\theta_g,z|\theta_d)dz = \int p(\theta_g|z,\theta_d)\overbrace{p(z|\theta_d)}^{=p(z)}dz \approx \frac{1}{J_g}\sum_{i=1}^{J_g} p(\theta_g|z^j,\theta_d),\ z^j \sim p(z) \tag{5}$$

and similarly $p(\theta_d|\theta_g,\theta_e) \approx \frac{1}{J_d}\sum_{j=1}^{J_d} p(\theta_d|z^j,X,\theta_g,\theta_e)$. Here, J_d, J_e and J_g are the number of Monte Carlo (MC) samples we take for the discriminator, encoder and generator, respectively.

Stochastic Gradient Update To run SGHMC, we follow Chen et al. [4] by taking M MC samples of θ_g and update the posterior $p(\theta_g|\theta_g)$ accordingly. For each sample θ_g^m for $m \in \{1,...,M\}$, do:

$$\theta_g^m \leftarrow \theta_g^m + v_g;\ \ v_g \leftarrow (1-\alpha)v_g + \eta\sum_{i=1}^{J_g}\sum_{j=1}^{J_d}\frac{\partial \log p(\theta_g|z^i,\theta_d^{j,m})}{\partial\theta_g} + n_g;\ \ n_g \sim \mathcal{N}(0,2\alpha\eta I) \tag{6}$$

where v is the momentum, n_g is the noise sample, α is the friction term for SGHMC, and η is the learning rate. We update the encoder and the discriminator weights in a similar fashion. In training, we rotate through the updates of the θ_g, θ_e and θ_d in every iteration.

During training, it is necessary to scale the losses according to the different values of J_g, J_d and J_e. For example, if $J_g = 2$ and $J_d = J_e = 1$, then the losses for the discriminator and encoder must be divided by J_g.

4 Experiments

We evaluate the BayesBiGAN on the MNIST and CIFAR datasets. We used deep neural networks for our permutation-invariant MNIST experiments and deep convolutional neural networks for our CIFAR experiments. All experiments were performed on a single Tesla K20 GPU on the Tufts High Performance Computing Clusters. We make our code publicly available at `https://github.com/julie-jiang/bayesbigan`.

4.1 Datasets

MNIST[1] is a very well understood benchmark dataset. Each sample in MNIST is a binary image of a handwritten digit between 0 and 9. The dataset is very well kempt with each sample being a consistent size, and minimal preprocessing necessary to get started. In addition to the ease of access, using this toy dataset makes it much simpler to compare performance against many other benchmark papers and projects.

CIFAR10[2] is a labeled dataset of tiny, colored images taken in real life. Each image is assigned a single label that indicates the type of the image in the foreground, which could be an airplane, bird, ship, etc.

We summarize the details of the datasets in Table 1.

Dataset	Train Samples	Test Samples	Sample Dims	Element Size	Target Space
MNIST	50,000	10,000	28× 28	256×1	1×10
CIFAR10	50,000	10,000	32×32	256×3	1×10

Table 1: Dataset statistics. Sample dimension is the size of the images in pixels. Element size is the dimension of each pixel, where 256×1 essentially means a grayscale image and 256×3 means an image in RGB. Target space is the number of correlated class labels.

4.2 Evaluation

The evaluation of GAN model is an interesting and inherently challenging problem [8]. To qualitatively and quantitatively assess the performance of our model, we will evaluate using the following methods.

One Nearest Neighbors Classification The BiGAN uses a Leave-One-Out (LOO) One Nearest Neighbor (1NN) accuracy metric [6] to evaluate our encoded feature representations. Using a 'nearest neighbor' approach, a distance metric is used to calculate the distance between a single encoded sample and a set of train encodings. This metric tells us which of the train samples our test sample is most similar to and we simply predict that our test sample has the same class as whatever sample it is closest to. This is a quick and effective, albeit also naive way, to evaluate our encoder.

Sample-based evaluation Since we will primarily be using the GAN on image datasets, the performance of the generator is best demonstrated by visually analyzing generated samples of images. We will evaluate our generator qualitatively by comparing original and reconstructed images side-by-side.

4.3 Permutation-invariant MNIST

Following the MNIST experiment set up in Donahue et al. [6], we use a permutation-invariant MNIST [14], in which each 28×28 image is flattened into a 784D vector. Each of the modules is a multi-layer perceptron that disregards the underlying spatial structure of the images. The latent distribution is a 50D continuous uniform distribution from $U(-1, 1)$. D, G and E each consist of two

[1] `http://yann.lecun.com/exdb/mnist/`

[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

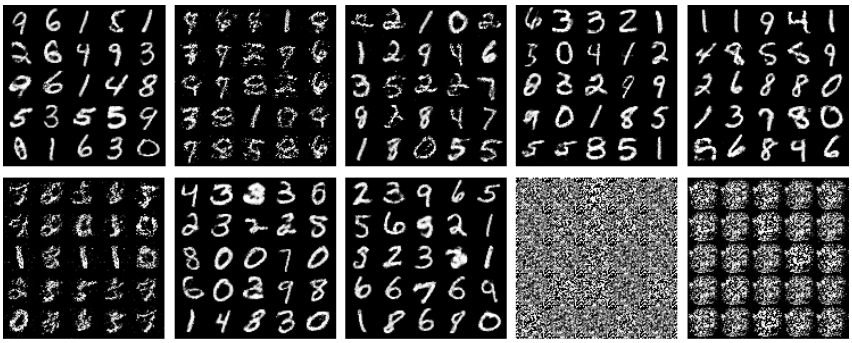
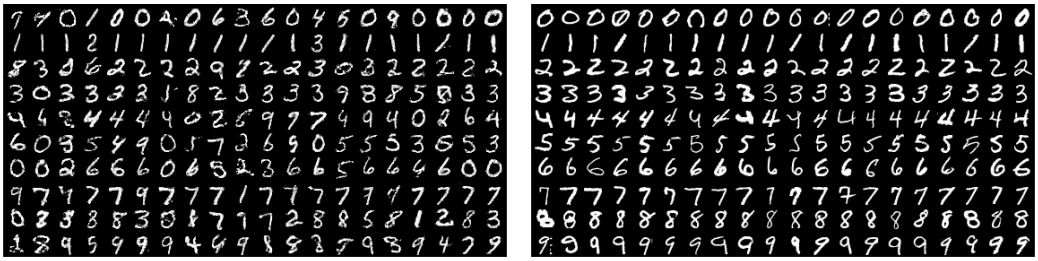


Figure 2: BayesBiGAN generated images from 10 MC samples of the generator for the MNIST dataset after 600 training epochs, illustrating a multi-modal generation. There are two "slow learners" that didn't learn quickly enough at the beginning stages.



(a) Generated images **(b)** Real images

Figure 3: BayesBiGAN reconstructed images from the MNIST dataset after 600 training epochs using one of the MC samples of the generator.

hidden layers with 1024 units. The first hidden layer is followed by a non-linearity, and the second hidden layer is followed by a parameter-free batch normalization [15] and another non-linearity. The final output layer transforms the output to the appropriate size. The discriminator and the encoder uses a LeakyReLU activation with a leak of 0.2, and the generator uses a ReLU activation. In all of our experiments, we fix $M = J_d = J_e = 1$ and experiment with various settings for J_g.

We train the BayesBiGAN on the permutation-invariant MNIST for 600 epochs with a mini-batch size of 128. As was done in BayesGAN [2] to speed up the "burn-in" process, we use the ADAM optimizer for the first 300 epochs before switching to a momentum based SGHMC optimizer. The learning rate is first set to be 2×10^{-4}, then decayed exponentially to 2×10^{-6} after 300 epochs. Additionally, we apply an l_2 weight decay of 2.5×10^{-5} to all multiplicative weights in the linear layers.

Figures 2-5 show results with the setting $J_g = 10$. Figure 2 displays generated images from 10 MC samples of generators. It is clear that our model can capture the multi-modal distribution of styles of handwriting. However, we also observe from our experiments that when J_g is too big, there are always generator samples who fail to learn in time. Once the discriminator and some of the other generators reach a certain threshold of performance, the discriminator becomes too strong for these slower generators to learn. Recall that the discriminator propagates gradients to the generator. Therefore, when the discriminator is too strong, the vanishing gradient problem will occur on the slower generators, thus widening the gap between the "slow learners" and the faster ones. This is an inevitable consequence by design.

We show an example of reconstruction $G(E(x))$ in Figure 3 with one of the ten MC samples of the generator, and the rest of the generators not including the two "slow learners" performed similarly. In Figure 4, we visualize the latent encodings $E(x)$ for 1500 randomly selected images from the test set, using tSNE [13], and in Figure 5 we plot the sigmoid cross entropy losses for each model. The discriminator and encoder losses are normalized with respect to the number of MC generators, and the generator loss is averaged over all of its MC samples.

The 1NN classification accuracy on the test set using the features learned by the BayesBiGAN is listed and compared with that of BiGAN in Table 2. We used the reported results in the BiGAN paper

	BiGAN	BayesBiGAN				
		$J_g = 1$	$J_g = 2$	$J_g = 3$	$J_g = 5$	$J_g = 10$
Accuracy	97.39	97.05	96.92	96.97	96.75	96.52

Table 2: Permutation-invariant MNIST One Nearest Neighbors (1NN) classification accuracy (%) of the encoder output on the test set. The BiGAN result is copied from Donahue, et al.[6]. The BayesBiGANs are trained with 600 epochs with varying numbers of MC samples of generator.

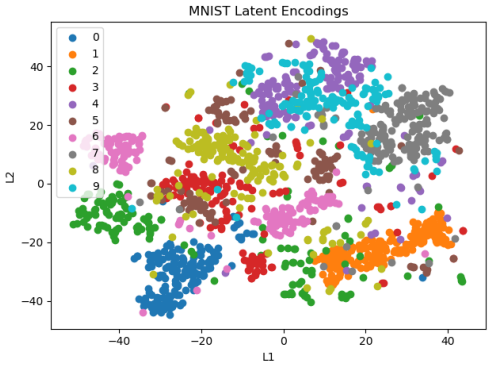


Figure 4: BayesBiGAN latent encoding visualizations of 1500 images from the MNIST test set. TSNE [13] is used to reduce the dimension of the encodings to 2.

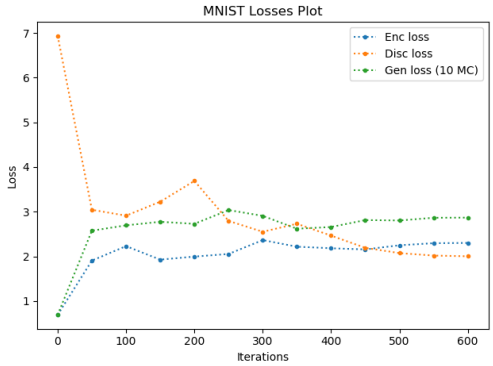


Figure 5: BayesBiGAN plot of the training losses against training epochs. The generator's loss is averaged over its 10 MC samples.

in Table 2, however we also corroborated their results by running it on our machine. The BiGAN classification accuracy was obtained after training for 400 epochs, whereas we let BayesBiGAN run for 600 epochs. Overall, we observe that more MC generators led to marginally lower classification accuracy for BayesBiGAN. We believe this is because having multiple generators – each learning at different paces and converging to one of the many modes of the distribution – confuses the encoder.

4.4 CIFAR

Next, we present results from training the BayesBiGAN on CIFAR, a dataset of natural images. Here, each of D, G and E is a convolutional neural network. Similar to the Imagenet experiment setup in BiGAN [6], most hidden layers are batch normalized [15]. The latent distribution is a 100D continuous uniform distribution from $U(-1, 1)$. The model architecture is described in full detail in Table 3.

The BayesBiGAN is trained on the CIFAR dataset for 800 epochs with a mini-batch size of 128, using ADAM for the first 400 epochs and SGHMC for the rest. The learning rates and decay rates are the same as the experiment on MNIST, and we also set $M = J_d = J_e = 1$. Due to the much higher computational cost, we only experimented with small values of J_g.

Figure 6 illustrates the reconstruction result with $J_g = 2$. In this experiment, the second MC sample of generator is a slower learner compared to the first. For comparison, we also ran the BiGAN model on the same dataset and show the results in Figure 7. While GANs cannot perfectly reconstruct these images, both BiGAN and BayesBiBGAN can effectively capture some of the crucial aspects of the images and show us some interesting results. We can see from Figure 6 that BayesBiGAN can capture different stylistic elements of the images and reproduce more vibrant images, while still keeping the main object of focus distinguishable.

We also report the 1NN classification accuracies on CIFAR in Table 4, confirming our belief that more encoders lead to lower classification accuracy. However, we believe that with a less naive classification training and ablation (by freezing weights of the encoder layer by layer and training a classifier on top of the frozen weights), we can achieve comparable performance with the encoder. We leave these experiments for future work.

Operation	Kernel	Strides	Padding	Group(s)	Output	BN?	Nonlinearity
$G(z)$ – $100 \times 1 \times 1$ input							
Dense	-	-	-	-	4096	✓	ReLU
Reshape	-	-	-	-	$256 \times 4 \times 4$	-	-
Deconvolution	5×5	2×2	Same	-	$128 \times 8 \times 8$	✓	ReLU
Deconvolution	5×5	2×2	Same	-	$128 \times 8 \times 8$	✓	ReLU
Deconvolution	5×5	1×1	Same	-	$32 \times 16 \times 16$	✓	ReLU
Deconvolution	5×5	2×2	Same	-	$3 \times 32 \times 32$	×	Sigmoid
$E(x)$ – $3 \times 32 \times 32$ input							
Convolution	11×11	4×4	Same	1	$96 \times 8 \times 8$	×	LeakyReLU
Max Pool	3×3	2×2	Same	-	$96 \times 4 \times 4$	-	Linear
Convolution	5×5	1×1	2	2	$256 \times 4 \times 4$	✓	LeakyReLU
Max Pool	3×3	2×2	Same	-	$256 \times 2 \times 2$	-	Linear
Convolution	3×3	1×1	1	1	$384 \times 2 \times 2$	✓	LeakyReLU
Convolution	3×3	1×1	1	2	$384 \times 2 \times 2$	✓	LeakyReLU
Convolution	3×3	1×1	1	2	$256 \times 2 \times 2$	✓	LeakyReLU
Reshape	-	-	-	-	1024	-	-
Dense	-	-	-	-	100	-	Linear
$D(x, z)$ – $3 \times 32 \times 32$ input x and $100 \times 1 \times 1$ input z							
Convolution	5×5	2×2	Same	1	$32 \times 16 \times 16$	×	LeakyReLU
Convolution	5×5	2×2	Same	1	$64 \times 8 \times 8$	×	Linear
Dense (add z)	-	-	-	-	64	✓	LeakyReLU
Convolution	5×5	2×2	Same	1	$128 \times 4 \times 4$	×	Linear
Dense (add z)	-	-	-	-	128	✓	LeakyReLU
Dense	-	-	-	-	1	×	Sigmoid

Table 3: CIFAR Architecture. The generator is the DCGAN generator from Radford et al. [3], a 4-layer deconvolutional neural network with ReLU activation functions and a sigmoid output layer.The encoder architecture follows AlexNet [11] through the fifth and last convolution layer, with the exception of batch normalization (BN) applied to each hidden layer and local reponse normalization removed. The architecture of AlexNet uses two separate filter paths (groups) for computational efficiency. Finally, the discriminator is 3-layer convolutional neural network on x with z added through a fully connected layer before normalization. The final dense layer of the discriminator transforms the output into a singleton. All LeakyReLU nonlinearities are applied with a leak of 0.2.

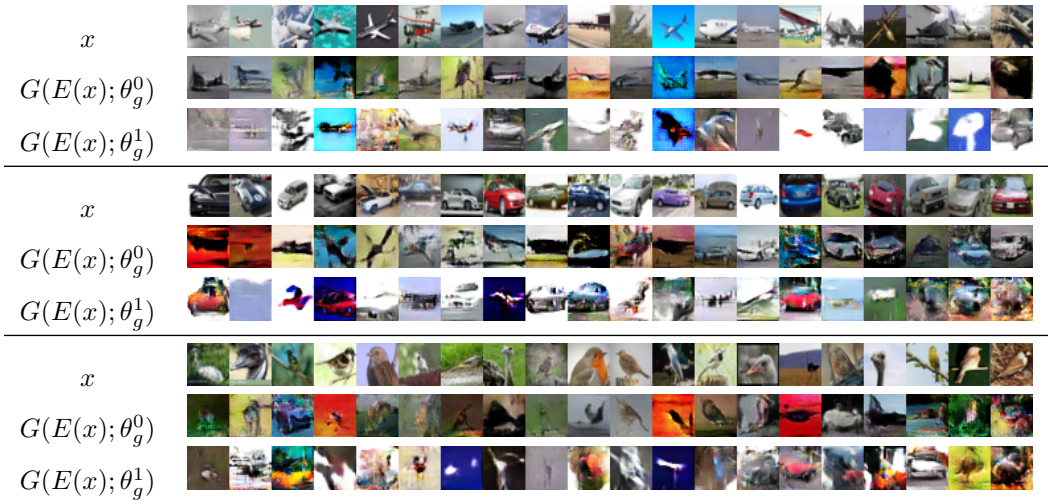


Figure 6: BayesBiGAN CIFAR reconstruction results after 800 training epochs and $J_g = 2$. We show images and reconstructions of 3 classes of images.

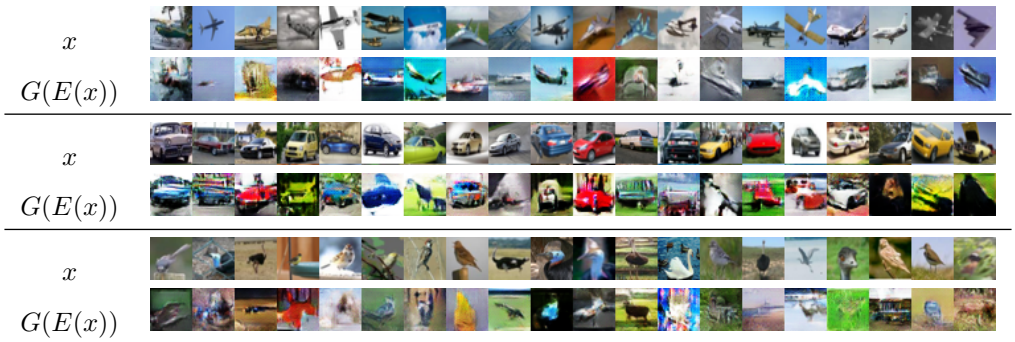


Figure 7: BiGAN CIFAR reconstruction results after 800 training epochs. We show images and reconstructions of 3 classes of images.

	BiGAN	BayesBiGAN		
		$J_g = 1$	$J_g = 2$	$J_g = 3$
Accuracy	43.69	43.65	43.16	36.36

Table 4: CIFAR One Nearest Neighbors (1NN) classification accuracy (%) of the encoder output on the test set. All results obtained after training on our machine for 800 epochs.

5 Discussion

In this project, we extended and combined the works of Bayesian GAN [2] and Bidirectional GAN [6] into our Bayesian Bidrectional GAN. Our BayesBiGAN effectively avoids mode-collapse and generates rich, multi-modal data distribution, while also providing a natural way to encode data into a latent space.

While we demonstrated competitive results of BayesBiGAN on the MNIST and CIFAR datasets, we also identify several limitations of the BayesBiGAN. First of all, hyperparameter tuning and model architecture setup are nontrivial feats on GANs and especially on BayesBiGAN. One of the most important architectural consideration is the choice of the discriminator, which jointly discriminates the input x and encodings z. Due to the nature of GAN training, if the discriminator learns too quickly, then one or both of the generator and the encoder will encounter the vanishing gradient problem and never learn effectively. On the contrary, if the discriminator learns too slowly, then the generator and encoder can fool the discriminator with poorly generated or encoded images. Donahue et al. [6] compared BiGANs with autoencoders, and by extension a natural alternative to BayesBiGAN are variational autoencoders. We believe that a variational autoencoder for these tasks might be much easier to implement while also retaining the same benefits.

Secondly, having multiple generators working at the same time is prone to result in some generator samples becoming "slow-learners" and furthermore confusing the encoder. We believe an easy improvement on our current pipeline is to take a larger number of MC samples, averaging out the effect of "slow-learners". However, this leads into our third and last limitation.

Finally, approximating intractable distributions with Hamiltonian Monte Carlo is computationally expensive, prohibiting us from running larger and/or longer experiments in a timely manner. With a single Tesla K20 GPU, the BayesBiGAN takes about 2 hours to run on the MNIST dataset with $J_g = 2$ and 20 hours with $J_g = 10$. On the CIFAR dataset, the BayesBiGAN takes nearly 2 days to run with $J_g = 2$. However, we believe this is also attributable to our machine, because BiGAN [6] reports that their model takes 30 minutes on the MNIST dataset but it took us 2 hours to replicate their experiment. Nonetheless, the exponential growth in time complexity with respect to the number of MC samples taken suggest that the HMC method is not particularly well-suited for large-scale image tasks.

6 Acknowledgements

The authors thank Michael Hughes for his teaching, guidance and helpful discussion throughout this work, as well as the Tufts High Performance Computing Cluster for providing us with the relevant computing resources and support.

References

[1] Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).

[2] Saatchi, Y., & Wilson, A. G. (2017). Bayesian GAN. In *Advances in neural information processing systems (pp. 3622-3631)*.

[3] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

[4] Chen, T., Fox, E., & Guestrin, C. (2014, January). Stochastic gradient hamiltonian monte carlo. In *International Conference on Machine Learning* (pp. 1683-1691).

[5] Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association, 112*(518), 859-877.

[6] Donahue, J., Krähenbühl, P., & Darrell, T. (2016). Adversarial feature learning. In *International Conference on Learning Representations*, 2017.

[7] Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., Courville, A. (2016). Adversarially learned inference. In *International Conference on Learning Representations*, 2017.

[8] Xu, Q., Huang, G., Yuan, Y., Guo, C., Sun, Y., Wu, F., & Weinberger, K. (2018). An empirical study on evaluation metrics of generative adversarial networks. *arXiv preprint arXiv:1806.07755*.

[9] Liu, M. Y., & Tuzel, O. (2016). Coupled generative adversarial networks. In *Advances in neural information processing systems* (pp. 469-477).

[10] Zhu, Jun-Yan and Park, Taesung and Isola, Phillip and Efros, Alexei A (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *Computer Vision (ICCV), 2017 IEEE International Conference*.

[11] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

[12] S. Yang, P. Luo, C. C. Loy, and X. Tang, From Facial Parts Responses to Face Detection: A Deep Learning Approach, in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[13] Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov), 2579-2605.

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE,* 1998.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.