maaGMA
Modified-Adversarial-Autoencoding Generator with Multiple Adversaries: Optimizing Multiple Learning Objectives for Image Generation with GANs

Sahil Chopra
Stanford University
353 Serra Mall, Stanford CA
schopra8@stanford.edu

Ryan Holmdahl
Stanford University
353 Serra Mall, Stanford CA
ryanlh@stanford.edu [*]

Abstract

The original formulation of GAN loss leveraged an approximation of Kullback - Leibler (KL) Distance [7]. However, more recent research seems to suggest that Earth Mover (EM) Distance may be better suited for GANs - enabling a greater likelihood of convergence, i.e. more stable training [2, 3, 8]. Herewith, we examine whether it is possible to train a single generator against multiple discriminators using these improved loss formulations. We propose maaGMA (Modified-Adversarial-Autoencoding Generator with Multiple Adversaries), a new GAN architecture, where two discriminators compete against different portions of an autoencoder generator to optimize multiple subtasks. We then apply maaGMA to the MNIST handwritten digits dataset and the Labeled Faces in the Wild (LFW) face recognition dataset [14, 12, 9]. Across these datasets, maaGMA demonstrates an ability to satisfy the objectives of each of its discriminators as well as its generator - suggesting that it is possible to train a single generator with multiple discriminators, with the potential to produce superior outputs.

1. Introduction

Traditionally, Generative Adversarial Networks (GANs) have been difficult to train. The framework relies on training two networks, a generator and a discriminator, to compete against one another. Often the discriminator network learns too quickly. Thus, no meaningful updates can be backpropagated into the generator and learning is stalled indefinitely [7, 2].

Recent research into GANs seems to suggest that the Kullback - Leibler (KL) Distance, utilized in the original formulation of GANs, may not be best suited for the framework's generative goals. Instead, Earth Mover (EM) Distance may provide much more stable learning, though taking longer to train [2, 3, 8].

Most literature utilizes GANs that leverage a single discriminator against a single generator. Herewith, we seek to explore whether these recently proposed EM Distance loss formulations, e.g. that seen in [8], are stable enough to successfully train multiple discriminators against a single generator, where each discriminator competes with different segments of the generator network. We measure success by steady convergence of the adversarial networks, the production of accurate image reconstructions from training samples, and the generation of novel output images that are qualitatively on par with those produced by single-discriminator networks, such as Adversarial Autoencoders.

If successful, this approach of assigning several discriminators to a single generator may be helpful for complex generation tasks. Difficult problems, e.g. sentence generation, rely on an interplay between various substructures that must fit together to produce both meaning and fluency. Each of these substructures can be approached as a learning subproblem, regional to a portion of generated output. Instead of developing a singular loss function that must either rigorously formalize these substructures or ignore them, it may be easier and more effective to train discriminators with simple, well-defined loss functions on each one of these subproblems. We would hope that this would help maintain the integrity of these substructures, while developing a more cohesive output. The development of our proposed architecture, maaGMA (Modified-Adversarial-Autoencoding Generator with Multiple Adversaries), is a first attempt to determine the feasibility of this ”discriminator-subtask” approach to producing improved generative outputs from ad-

[*] This author contributed equally to this work. Both Sahil Chopra and Ryan Holmdahl are first authors.

versarial networks.

2. Background & Related Work

2.1. Generative Networks

In the subfield of generative networks, there are three primary approaches: 1) Restricted Boltzmann Machines (RBMs), 2) Autoencoders (AEs), and 3) Generative Adversarial Networks (GANs).

Boltzmann Machines consist of symmetrically connected neurons that make stochastic decisions as to their activation in order to approximate unknown probability distributions of input data. The learning algorithm for Boltzmann Machines is traditionally very slow but can be sped up by restricting the connections that are possible within these networks [1]. These Restricted Boltzmann Machines (RBMs) consist of a single layer of visible units and a single layer of hidden units, with no visible-visible or hidden-hidden connections - thus restricting the structure of the RBM to that of a bipartite graph [11].

Several deep architectures have been developed on top of RBMs. Specifically, Deep Belief Networks (DBNs) and Deep Boltzmann Machines (DBMs) have been shown to be successful. DBNs maintain an undirected RBM at its top layer, while leveraging previous layers as directed sigmoid belief networks [10]. On the other hand, DBMs are somewhat more general in that they allow for connections between hidden units across layers, while still restricting intra-layer connections to maintain the bipartite graph structure for faster learning [19]. The issue presented by Boltzmann Machines is that their gradients are computationally intractable in most cases, so Markov Chain Monte Carlo (MCMC) methods are often used to approximate these derivatives [7].

Autoencoders (AEs) are neural networks that are trained to produce outputs that are equal to their inputs [5]. Within the field of generative networks, there are two popular variants - Variational Autoencoders (VAEs) and Adversarial Autoencoders (AAEs). Variational Autoencoders consist of two halves, an encoder and a decoder. The goal of the encoder is to produce a low dimensional representation of an input image. The decoder then uses strided convolutions to produce an output image from this embedding, while minimizing some distance between the output and input images, i.e. reconstruction loss. VAEs leverage this encoder-decoder structure to introduce a second loss term, which computes the KL Divergence between the distribution of embedding vectors produced by the encoder and a unit normal distribution. The KL Divergence loss term is included so that the embeddings will fit some target distribution that can then be sampled and decoded to create new outputs [13]. Meanwhile, AAEs explicitly train a discriminator network to force the embedding vector to the target distribution [15]

Generative Adversarial Networks (GANs) rely on optimization over network outputs rather than latent variables. Specifically, GANs consist of a generator network (G) and discriminator network (D). Here D's objective is to correctly discriminate between between true images and generated images, while G's objective is to maximize D's error, i.e. fool the discriminator into believing that the generated images are real [7]. Mathematically, this is formulated as a Minimax game between G and D:

$$\min_G \max_D V(G, D)$$

$$V(G, D) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

While training GANs it is often common to let the generator update once for every n epochs of training the discriminator. Similarly, in practice V may not provide sufficient gradient for G to learn well, as D can reject samples with high confidence during the early stages of training G. As a result, $log(1 - D(G(z)))$ saturates, so instead we often maximize $log(D(G(z))$ to get stronger gradients early on during training [7].

2.2. Advances in GAN Architectures

While the original GAN paper utilized MLPs, there have been many proposed GAN architectures in the subsequent years [7]. The Deep Convolution GAN (DCGAN) is one such commonly used model that has been shown to provide reasonable performance on image generation tasks. Given an input vector Z derived from sampling a uniform distribution, the network projects and reshapes the input before applying four layers of strided convolutions to produce a final output image of appropriate size. The DCGAN does not use pooling layers, instead utilizing strided convolutions in its discriminator networks. Additionally, DCGAN applies batch normalization to both generator and discriminator networks, removes fully connected layers in deeper architectures, uses ReLU activation for all layers of the generator except for a single Tanh at the final output layer, and utilizes a Leaky ReLU activation in the discriminator network across all layers. In our proposed maaGMA model, we use a slightly modified version of the DCGAN architecture within the generator [18].

Other popular GAN formulations include InfoGAN and Conditional GAN. InfoGAN attempts to disentangle different semantic features of input images by additionally maximizing the mutual information between a small subset of latent variables and observations. Thus, it both generates images and disentangles features in an unsupervised fashion [6]. Meanwhile, Conditional GANs provide an explicit encoding of desired semantics to both the generator and discriminator [17]. Recently, there have been many attempts to combine VAEs and GANs together, but this has been met with varied success [16, 4].

2.3. Advances in GAN Loss Formulations

The original GAN formulation tries to learn a differentiable probability density function, P_Θ, optimized through maximum likelihood estimation to approach the true distribution P_r. In the limit, this is equivalent to minimizing the KL divergence $KL(P_r||P_\Theta)$. Rather than learning the probability distribution P_Θ via MLE, another possibility is learning a function G_Θ that transforms an existing distribution Z into P_Θ such that $P_\Theta = G_\Theta(Z)$. The objective now becomes to minimize the distance d between P_r and P_Θ. There several potential choices for this distance function, but as [3] shows, there exist sequences of distributions that don't converge under total variation, KL divergence, or JS divergence, but do under Earth Mover (EM) Distance, also known as the Wasserstein Distance.

The goal of EM distance is to move probability mass from one distribution to another. We want to minimize the effort required from P_Z to P_Θ. Thus, the EM distance can be described as follows.

$$W(P_Z, P_\Theta) = \inf_{\gamma \in \pi(P_Z, P_\Theta)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$$

Here, we are computing the probability mass transport plan γ that requires the least energy, i.e. expected value of the difference in distributions, but ensures that the probability mass that leaves x equals the original amount at x, $P_Z(x)$, and that probability mass that enters y equals the amount of mass that ends up at y, $P_\Theta(y)$, i.e. conservation of probability mass during transport. EM Distance is computationally intractable but the WGAN paper proposes a formula that computes a reasonable and efficient approximation of EM distance up to some multiplicative constant. First, the formula for EM Distance can be rewritten as follows using the Kantorovich-Rubinstein duality:

$$W(\mathbb{P}_Z, \mathbb{P}_\Theta) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_Z}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\Theta}[f(x)]$$

If one replaces the supremum over 1-Lipschitz functions with supremum over K-Lipschitz functions, then the supremum becomes $K * W(P_z, P_\Theta)$. If we have some family of parameterized functions $\{f_{w_{w\in\mathcal{W}}}\}$, where w are weights, $\mathcal{W}$ is the set of all possible weights, and f_w is the function that describes our Discriminator, then an approximation can be derived as follows:

$$\max_{w\in W} \mathbb{E}_{x\sim P_z}[D_w(x)] - \mathbb{E}_{z\sim P(z)}[D_w(G(z))]$$

Unfortunately, to have this distance metric converge, one must use an RMSProp optimizer and employ gradient clipping to prevent the gradients from exploding. Altogether, the training process for WGAN consists of computing an approximation of $W(P_z, P_\Theta)$ by training f_w till convergence, computing the Θ gradient once we have an optimal f_w as $-\mathbb{E}_{z\sim Z}[\nabla_\Theta D_w(G(z))]$ by sampling several $z \sim Z$, and iteratively updating Θ until convergence [3].

Furthering this work, improvements for the WGAN were proposed in subsequent papers to eliminate the need for gradient clipping and to allow for convergence using momentum-based optimizers. This was done by removing batch normalization and adding a gradient penalty term to ensure the required 1-Lipschtiz constraint was met for the distance approximation function. The final loss function from this Improved WGAN formulation is as follows [8]:

$$\mathbb{E}_{\tilde{x}\sim\mathbb{P}_\Theta}[D(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \lambda\,\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(||\nabla_{\hat{x}}D(\hat{x})||_2 - 1)^2]$$

We use this Improved WGAN loss to train the adversarial components of our proposed maaGMA architecture.

3. Methods

We now present the maaGMA architecture, a novel method for optimizing a single generative network via competition with multiple adversaries with different, potentially conflicting objectives. The maaGMA architecture consists of an autoencoder placed into conflict with two adversarial discriminators. One adversary discriminates between the vector embeddings produced by the autoencoder's encoder and vectors sampled from a target probability distribution. The other adversary discriminates between the outputs of the autoencoder's decoder and samples from the training data distribution. The autoencoder, meanwhile, seeks to minimize its reconstruction loss. This architecture, shown in figure 1, forces the generator network to satisfy three distinct and oppositional objectives.

3.1. Generator

The generator network can be any form of autoencoder. Mathematically, the goal for the autoencoding generator is to learn an encoding function f_{enc} and a decoding function f_{dec} to satisfy the following expression, given a data distribution p_d and reconstruction loss L:

$$\arg\min_{f_{enc}, f_{dec}} \mathbb{E}_{x\sim p_d}[L(f_{dec}(f_{enc}(x)), x)]$$

Other variations of the autoencoder model, such as the conditional autoencoder, can be used depending on the given task.

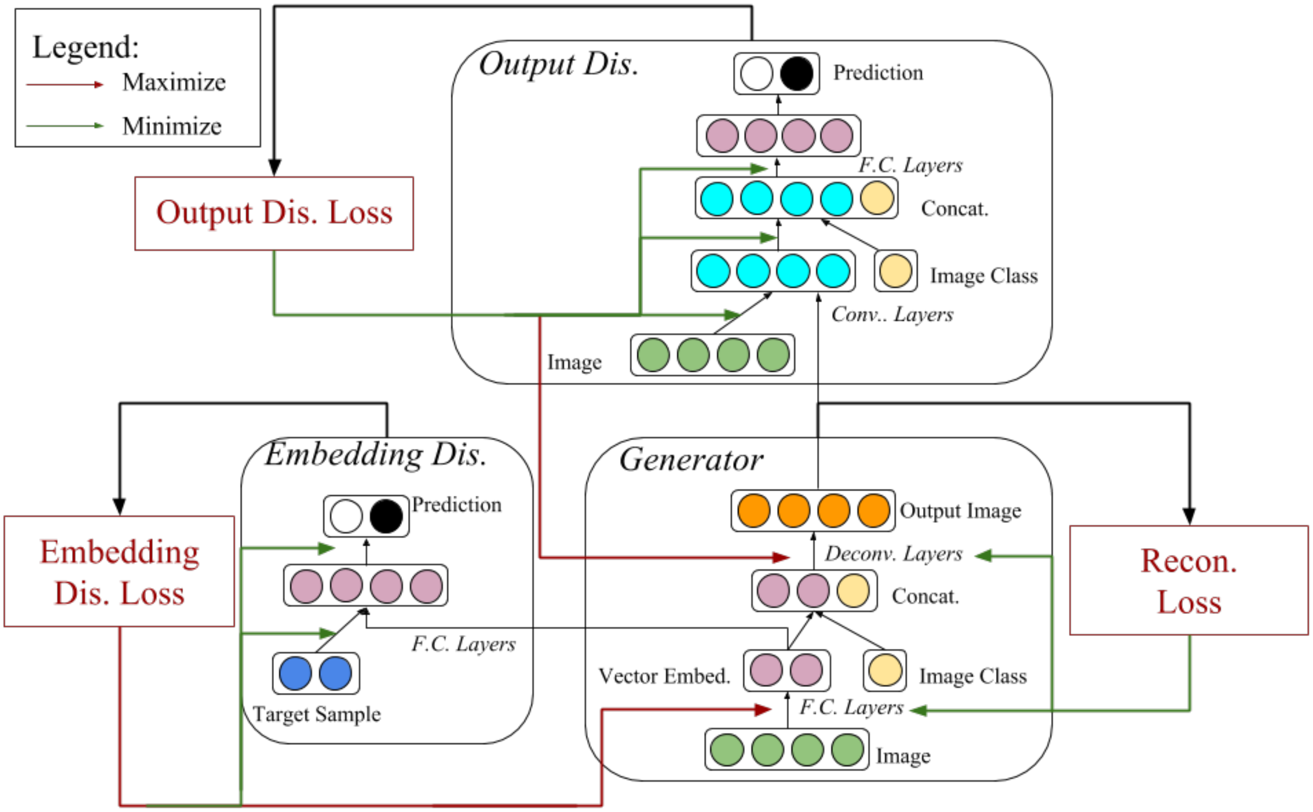

Figure 1. **maaGMA Architecture.** maaGMA consists of three components: the generator, the embedding discriminator, and the output discriminator. The generator is an autoencoder (here, a conditional autoencoder) which seeks to minimize its reconstruction loss. The embedding discriminator is a binary classifier which seeks to minimize its Improved WGAN discriminator loss. The generator's encoding layers (here, fully connected layers) are updated to maximize this loss. The output discriminator seeks to minimize its Improved WGAN discriminator loss. The generator's decoding layers (here, deconvolution/strided convolution layers) seek to maximize this loss. [8]

3.2. Embedding Discriminator

The embedding discriminator attempts to discriminate between samples from the target probability distribution q and embeddings produced by the encoder of the generator network. More specifically, given a target distribution q, the embedding discriminator attempts to learn a classification function d_{em} to satisfy

$$\arg\min_{d_{em}} \mathbb{E}_{x\sim p_d}[d_{em}(f_{enc}(x))] - \mathbb{E}_{z\sim q}[d_{em}(z)]$$

While the embedding discriminator learns to distinguish between samples from the two distributions, the encoding function f_{enc} is updated to confuse the embedding discriminator; that is, it seeks to satisfy

$$\arg\min_{f_{enc}} -\mathbb{E}_{x\sim p_d}[d_{em}(f_{enc}(x))]$$

The functions are learned adversarially, with the discriminator updating to decrease the expectation of d_{em} on the encoder outputs. Meanwhile, the encoder updates to both satisfy its reconstruction objective and to confuse the discriminator. These networks combined constitute an Adversarial Autoencoder [15]. Ultimately, this competition is intended to force

$$\int_x p_{enc}(z|x)p_d(x)dx = q(z)$$

for all possible embeddings z, where $p_{enc}(z|x)$ is the probability of producing the embedding z applying f_{enc} to x.

3.3. Output Discriminator

The output discriminator attempts to discriminate between samples from the data distribution, p_d, and samples from the generator output distribution. More specifically, the output discriminator attempts to learn a classification function d_{out} to satisfy

$$\arg\min_{d_{out}} \mathbb{E}_{x\sim p_d}[d_{out}(f_{dec}(f_{enc}(x)))] - \mathbb{E}_{x\sim p_d}[d_{out}(x)]$$

While the output discriminator learns to distinguish between samples from the two distributions, the decoding function f_{dec} is updated to confuse the output discriminator; that is, it seeks to satisfy

$$\arg\min_{f_{dec}} -\mathbb{E}_{x\sim p_d}[d_{out}(f_{dec}(f_{enc}(x)))]$$

As with the embedding discriminator, the functions are learned adversarially. Note that, while we could update f_{enc} in this adversarial relationship, we do not. In practice, we found that the direct competition of three objectives over the same parameters made convergence and quality results more difficult to achieve. Instead, f_{enc} is affected indirectly, as the changes in the definition of f_{dec} warrant changes in f_{enc} to continue to satisfy the reconstruction objective.

3.4. Optimization

The interdependent and conflicting objectives of the learnable functions f_{enc}, f_{dec}, d_{em}, and d_{out} require they be learned simultaneously. Practically, this can be done via an iterative loss minimization. We minimize the following set of functions, listed in arbitrary sequence:

$$\mathbb{E}_{x\sim p_d}[L(f_{dec}(f_{enc}(x)), x)] \tag{1}$$

$$-\mathbb{E}_{x\sim p_d}[d_{em}(f_{enc}(x))] \tag{2}$$

$$-\mathbb{E}_{x\sim p_d}[d_{out}(f_{dec}(f_{enc}(x)))] \tag{3}$$

$$\begin{aligned}(\mathbb{E}_{z\sim q}[d_{em}(z)] - \mathbb{E}_{x\sim p_d}[d_{em}(\hat{z})] + \\ \lambda\,\mathbb{E}_{x\sim p_d}[(||\nabla_{\hat{z}} d_{em}\hat{z})||_2 - 1)^2])\end{aligned} \tag{4}$$

where $\hat{z} = f_{enc}(x)$;

$$\begin{aligned}(\mathbb{E}_{x\sim p_d}[d_{out}(x)] - \mathbb{E}_{x\sim p_d}[d_{out}(\hat{x})] + \\ \lambda\,\mathbb{E}_{x\sim p_d}[(||\nabla_{\hat{x}} d_{out}\hat{x})||_2 - 1)^2])\end{aligned} \tag{5}$$

where $\hat{x} = f_{dec}(f_{enc}(x))$.

Eq. 1 propagates to f_{enc} and f_{dec} and represents the reconstruction loss of the autoencoder. Eq. 2 propagates to f_{enc} and confuses the embedding discriminator. Eq. 3 propagates to f_{dec} and confuses the output discriminator. Eq. 4 propagates to d_{em} and represents the Improved WGAN loss of the embedding discriminator. Eq. 5 propagates to d_{out} and represents the Improved WGAN loss of the output discriminator.

In practice, these updates can be weighted according to the values typical to the given generation task. One can also delay the start of training for one of these functions if it is able to memorize training data faster than its competitors.

4. Experiments[1]

4.1. Evaluation Metrics

In our experiments, we endeavored to see if results on par or superior to those from adversarial autoencoders could be achieved with maaGMA. We are particularly interested in the autoencoder's reconstruction quality, the vector embedding's conformity to the target distribution, and the quality of images produced by inserting random samples into the decoder. Our assessments are necessarily qualitative, as recent work has shown that usual quantitative assessments of generative networks are flawed [20]. We do not argue that our outputs are better than those produced by other models, but merely comment on notable features. We are more interested in whether a generator affected by multiple adversaries can learn at all without collapsing.

4.2. MNIST

4.2.1 Data

The MNIST Handwritten Digits Database consists of 60,000 examples of handwritten numerals across 10 image classes, which we used to train maaGMA for our first experiment. The training samples are 28 x 28 grayscale images [14].

4.2.2 maaGMA Hyperparameters

When training maaGMA we used a conditional autoencoder and a conditional output discriminator, where the condition was a 1-hot-vector representative of the desired digit class, as proposed in [15]. This encouraged the vector embedding to capture the digit's style rather than its identity and allowed the desired digit to be created when using a randomly sampled vector embedding. The encoder consisted of fully-connected layers, while the decoder was a deconvolutional network structured after the DCGAN; the autoencoder optimized against the L2 reconstruction loss.

The embedding discriminator was trained to distinguish between samples from a ten-dimensional standard Gaussian distribution and the vector embeddings created by the autoencoder. We trained this discriminator five times on each minibatch, while the other networks were given only one iteration on each. This was to compensate for difficulties in competitiveness the embedding discriminator seemed to be experiencing.

For this experiment, we found that the output discriminator was able to overfit the training set fairly quickly when faced with early outputs of the adversarial autoencoder;

[1] Our implementation leverages code from the Stanford 224N model.py framework and the Improved WGAN loss formulation from [8], available at `github.com/igul222/improved_wgan_training/blob/master/gan_mnist.py`. We leveraged the loss formulations and reimplemented it for our maaGMA model.

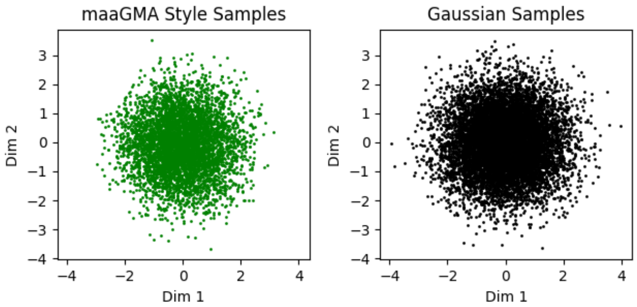

Figure 2. **MNIST maaGMA embedding distribution and target distribution.** Withheld test samples from the MNIST dataset were inserted into the maaGMA encoder, and the resulting embeddings are projected in the plot on the left. On the right are projected samples from a standard Gaussian distribution. The distributions are remarkably similar, indicating that maaGMA was successful in satisfying the embedding discriminator's objective.

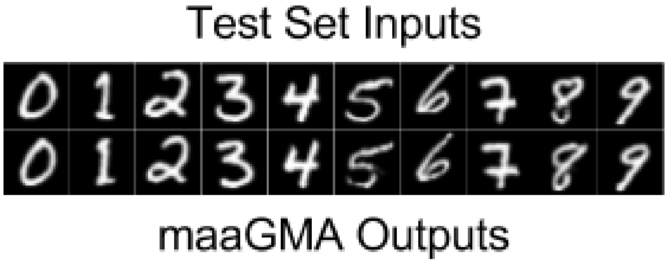

Figure 3. **MNIST maaGMA reconstruction.** Withheld test samples from the MNIST dataset were inserted into maaGMA's autoencoding generator, and the resulting outputs are shown on the second row, below the inputted images. The autoencoder is able to accomplish its objective of producing outputs similar in style and appearance to its inputs. These samples were randomly chosen and were not cherry-picked.

therefore, we ran 300 epochs with only the adversarial autoencoder, and then trained it in tandem with the output discriminator for two epochs. At epoch 302, the generator begins attempting to confuse the output discriminator. By this point, the outputs of the adversarial autoencoder were of high enough quality that the output discriminator could not easily detect them and collapse the training of the other networks.

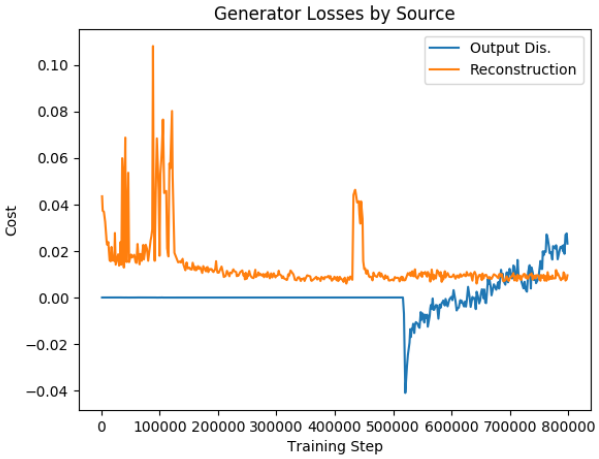

Figure 4. **MNIST maaGMA generator costs.** The generator's reconstruction cost and the cost on it imposed by the output discriminator are shown. A small increase in reconstruction cost is visible when the output discriminator begins around batch 500000, indicating the competitive nature of the two networks. The cost imposed by the output discriminator is very low initially, but grows as the network learns to better understand the data. The potential for eventual overfitting is evident in the constant climb of the output discriminator's imposed cost. Note that the cost imposed by the embedding discriminator was relatively constant and for clarity not included here.

4.2.3 Results

maaGMA successfully fit the vector embeddings to a Gaussian distribution, as shown in figure 2. The autoencoder portion is also able to faithfully recreate input images, as shown in figure 3. When decoding random samples from the target Gaussian distribution, maaGMA produces slightly sharper images than an adversarial autoencoder, as shown in figure 5. Losses from training are shown in figure 4; note that we do not include the discriminator losses, which tended to hover around static values.

4.3. LFW

4.3.1 Data

The LFW dataset consists of over 13000 images of faces collected from the web using the Viola-Jones face detector [12]. The faces are captured in various poses and rotations. We use the LFW3D dataset, a "frontalization" of LFW [9]. Each image in this dataset is a front-on estimation of an image in the original LFW dataset, created via 3D approximation. The images are downscaled to 32 x 32 grayscale images for training in maaGMA.

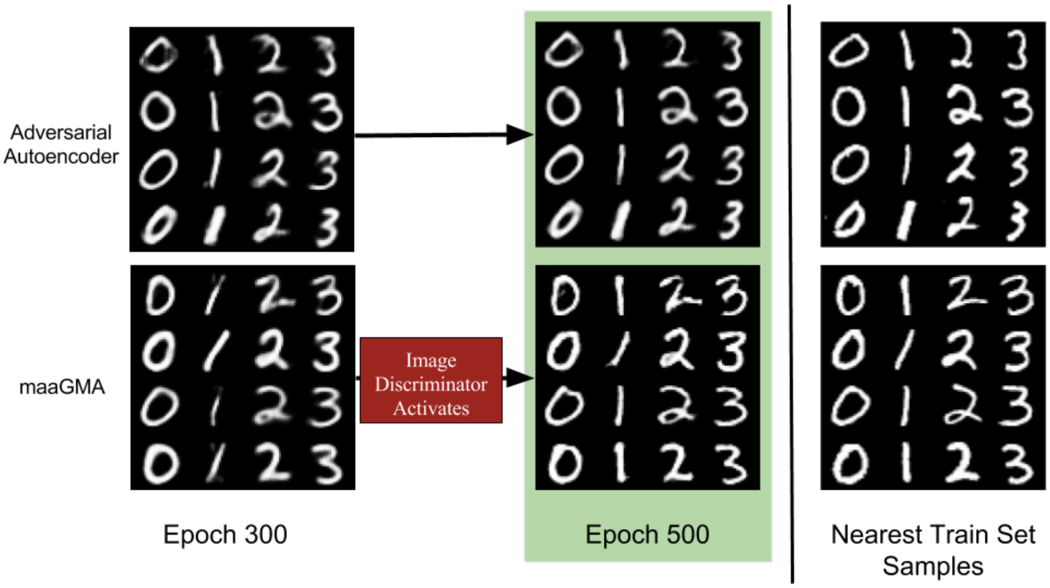

Figure 5. **MNIST Adversarial Autoencoder & maaGMA Random Outputs**. On the top row, samples from a standard Gaussian distribution were fed into the decoder of an adversarial autoencoder at epochs 300 and 500. On the bottom row, samples from a standard Gaussian distribution were fed into the decoder of maaGMA at epochs 300 and 500. The maaGMA output discriminator activated at epoch 302. The images at epoch 500, after the output discriminator activated, are significantly sharper than those of the adversarial autoencoder at the same epoch. Neither network is memorizing samples from the training set, as shown on the right.

4.3.2 maaGMA Hyperparameters

For this experiment, we did not use conditional variant of maaGMA, as there was no useful class label to apply. The encoder was convolutional rather than fully-connected. We trained the output discriminator from the first epoch and began confusing the generator at the 20th, as we found that overfitting on the LFW dataset was less of a problem than on the MNIST dataset. Once again, the vector embedding was forced to a ten-dimensional standard Gaussian distribution. The embedding and output discriminators were only trained once on each batch.

4.3.3 Results

While experiencing more difficulty on the LFW task than on MNIST, maaGMA was still able to optimize towards its competing objectives. Figure 6 shows the vector embeddings did not perfectly conform to the target distribution but were still forced towards it. Figure 8 shows some success on the reconstruction task, with skin tones and simpler features of the image being learned successfully, but more fine facial features being lost. This may be due to the low dimensionality of embedding vector. Figure 7 compares the outputs of the decoder given random samples to the outputs of the adversarial autoencoder. The maaGMA outputs showed sharper lines than the more blurred AAE outputs but were also pixelated and grainy. Figure 9 illustrates the generator losses from the output discriminator and reconstruction.

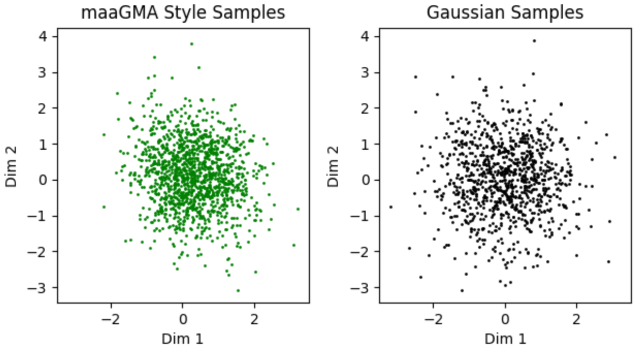

Figure 6. **LFW maaGMA embedding distribution and target distribution.** Withheld test samples from the LFW dataset were inserted into the maaGMA encoder, and the resulting embeddings are projected in the plot on the left. On the right are projected samples from a standard Gaussian distribution. The maaGMA encoder captures the target distribution to some extent, though not entirely.

5. Conclusion

We found that a single generator network can successfully accommodate the pressures of multiple adversarial discriminators, satisfying the imposed objectives to some extent without a collapse in training. maaGMA, in particu-

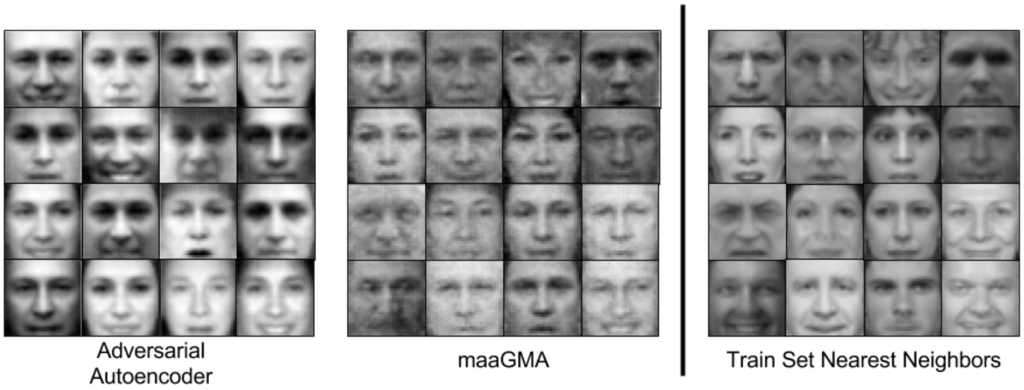

Figure 7. **LFW Adversarial Autoencoder & maaGMA Random Outputs.** On the left, samples from a standard Gaussian distribution were fed into the decoder of an adversarial autoencoder at epoch 800. In the middle, samples from a standard Gaussian distribution were fed into the decoder of maaGMA at epoch 800. The images from maaGMA have noticeably sharper features, where the bounds of the faces and the regions surrounding the eyes, noses, and mouths are more well defined. The faces from the adversarial autoencoder, meanwhile, seem ”smoother” at the edges, due to the low cost of blurring in L2-loss reconstructions. The maaGMA photos have ”dot-like” artifacts throughout the images, which appeared early on during training but significantly diminished by epoch 800. If the model was trained over a greater number of epochs with more fine-tuned learning rates, these pixelation artifacts might disappear. Finally, on the right, we have the nearest neighbors from the training set to the maaGMA produced images. As we can see the faces are not identical, so maaGMA is not simply memorizing the data.

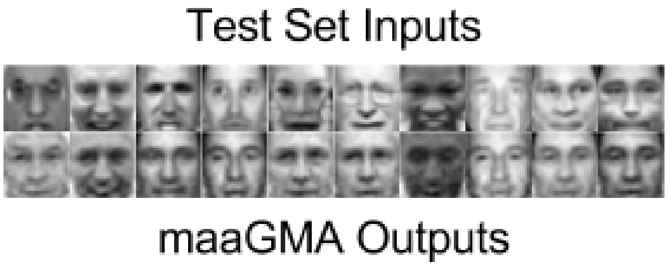

Figure 8. **LFW maaGMA reconstructions.** Withheld test samples from the LFW dataset were inserted into maaGMA's autoencoding generator, and the resulting outputs are shown on the second row, below the inputted images. The autoencoder is able to accomplish its reconstruction objective somewhat, capturing here skin color and obvious image artifacts like corner colors, but loses facial features and emotions.

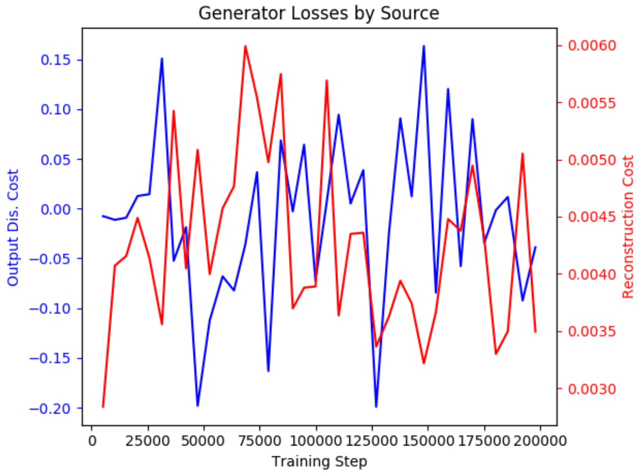

Figure 9. **LFW maaGMA generator costs.** The generator's reconstruction cost and the cost on it imposed by the output discriminator are shown. Reconstruction cost hovers around 0. Cost from output discriminator is more variable. Lower learning rates may lead to greater convergence.

lar, was able to reconstruct input images, force its vector embedding to resemble a Gaussian distribution, and create convincing new outputs that confused a discriminator. Additionally, maaGMA was able to produce qualitatively sharper images on the MNIST and LFW datasets than the baseline Adversarial Autoencoder.

Future work can use our findings to apply multiple adversaries to different generator structures and tasks. In addition, although the Improved WGAN loss is fairly stable, there is still a significant amount of hyperparameter tuning required for multiple adversaries to compete effectively with one another. Hyperparameters such as the relative learning rates of the networks, when each network should begin training, the importance of confusing each adversary, etc., significantly affect performance and are currently selected by hand. An algorithmic means for learning these values would facilitate training generative networks with multiple adversaries.

References

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines*. *Cognitive Science*, 9(1):147–169, 1985.

[2] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks, 2017.

[3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan, 2017.

[4] J. Bao, D. Chen, F. Wen, H. Li, and G. Hua. Cvae-gan: Fine-grained image generation through asymmetric training, 2017.

[5] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294, 1988.

[6] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets, 2016.

[7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[8] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans, 2017.

[9] T. Hassner, S. Harel, E. Paz, and R. Enbar. Effective face frontalization in unconstrained images, 2014.

[10] G. E. Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

[11] H. Hu, L. Gao, and Q. Ma. Deep restricted boltzmann networks, 2016.

[12] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report.

[13] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[15] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders, 2015.

[16] L. Mescheder, S. Nowozin, and A. Geiger. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks, 2017.

[17] M. Mirza and S. Osindero. Conditional generative adversarial nets, 2014.

[18] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[19] R. Salakhutdinov and G. Hinton. Deep boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455, 2009.

[20] L. Theis, A. v. d. Oord, and M. Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.