2020 25th International Conference on Pattern Recognition (ICPR)
Milan, Italy, Jan 10-15, 2021

Hierarchical Mixtures of Generators for Adversarial Learning

Alper Ahmetoğlu
Department of Computer Engineering
Boğaziçi University
Bebek İstanbul 34342 Turkey
alper.ahmetoglu@boun.edu.tr

Ethem Alpaydın
Department of Computer Science
Özyeğin University
Çekmeköy İstanbul 34794 Turkey
ethem.alpaydin@ozyegin.edu.tr

***Abstract*—Generative adversarial networks (GANs) are deep neural networks that allow us to sample from an arbitrary probability distribution without explicitly estimating the distribution. There is a generator that takes a latent vector as input and transforms it into a valid sample from the distribution. There is also a discriminator that is trained to discriminate such fake samples from true samples of the distribution; at the same time, the generator is trained to generate fakes that the discriminator cannot tell apart from the true samples. Instead of learning a global generator, a recent approach involves training multiple generators each responsible from one part of the distribution. In this work, we review such approaches and propose the hierarchical mixture of generators, inspired from the hierarchical mixture of experts model, that learns a tree structure implementing a hierarchical clustering with soft splits in the decision nodes and local generators in the leaves. Since the generators are combined softly, the whole model is continuous and can be trained using gradient-based optimization, just like the original GAN model. Our experiments on five image data sets, namely, MNIST, FashionMNIST, UTZap50K, Oxford Flowers, and CelebA, show that our proposed model generates samples of high quality and diversity in terms of popular GAN evaluation metrics. The learned hierarchical structure also leads to knowledge extraction.**

I. Introduction

In generative modeling, we are given a data set $\mathcal{X} = \{x^t\}_t$ sampled from some unknown probability distribution $p(x)$ and we want to be able to generate new instances from $p(x)$. This is an unsupervised learning problem and the usual approach is to first build an estimator for $p(x)$ and then sample from that. The generative adversarial network (GAN) [5] is interesting in that it learns a generative model without explicitly modeling $p(x)$ but by using an auxiliary discriminative model, thereby transforming an unsupervised learning problem into a supervised learning problem.

A GAN model is composed of two learners, a generator G and a discriminator D. G takes as input random z drawn from some simple parametric distribution of relatively low dimensionality, e.g., a zero-mean Gaussian with unit covariance, and learns to transform it to a valid instance $\tilde{x}$ from (the unknown) $p(x)$. G is implemented as a deep neural network that takes z as input, generates $\tilde{x}$ as output, and has many layers in between necessary for the transformation; the weights in G are denoted by θ. The $\tilde{x}$ that are generated by G are called *fake* because they are synthetic. The discriminator D is a two-class classifier that learns to discriminate such fakes from true x^t sampled from the training set $\mathcal{X}$. D is another deep neural network with either $\tilde{x}$ or x^t as input and 0 or 1 as the desired output respectively for the single sigmoid output. Again D has as many hidden layers as necessary for the task; the weights in D are denoted by ϕ.

The objective function is

$$\min_{\theta} \max_{\phi} \quad \mathbb{E}_{x^t \sim p(x)}[\log D(x;\phi)] + \mathbb{E}_{z \sim p(z)}[\log (1 - D(G(z;\theta);\phi))] \tag{1}$$

We train the weights of both G and D using gradient-based optimization, alternating between the two. D wants to maximize the likelihood for true instances x^t (drawn from unknown $p(x)$ as represented by the training set $\mathcal{X}$) and minimize the likelihood for fake instances generated by G. At the same time, G wants to generate fakes for which D assigns as high likelihood as possible. As G gets better in generating fakes for which D assigns high likelihood, D is forced to better separate them from true instances, which in turn forces G to generate even better fakes, and so on.

GANs are used successfully especially in image generation. A well-trained GAN can generate images that are almost indistinguishable by humans [2], [12], [13]; still, there are two main difficulties in training: Sometimes G learns only a part of the true $p(x)$ and can generate only a subset of the possible x; this is called *mode collapse* because it is an indication that G does not cover all the modes of $p(x)$. The second problem is that of *vanishing gradients* that we always have in training deep neural networks; note that here because both D and G are deep, G is doubly deep because its gradient needs to be back-propagated through D.

There is recent work in the literature that focuses on these problems. To solve problems related to training, it has been proposed to use either different objective functions, regularization methods, or architectures; see [3], [9], [15] for good surveys of the state of the art.

The direction we pursue in this study is to use multiple generators each one responsible for generating a local region of $p(x)$. Different local generators will learn to cover different modes and this will help alleviate the mode col-

978-1-7281-8808-9/20/$31.00 ©2020 IEEE

lapse problem. They also help with the problem of vanishing gradients because local generators are simpler, i.e., more shallow, and hence the paths through which the gradient is back-propagated are shorter. We review three previously proposed approaches from the literature, namely multi-agent diverse GAN (MADGAN) [4], mixture GAN (MGAN) [8], and mixtures of experts GAN (MEGAN) [20].

We propose the hierarchical mixture of generators that has a tree structure with internal decision nodes that divide up the latent z space and leaves that are local generators responsible from a local z region generating a subset of $p(x)$. Since the splits are soft, given the tree structure, the split parameters at the internal nodes as well as those of the generators in the leaves can be updated using gradient-descent. Note that it is only G that is modeled this way and split locally, there is still a single D implemented as a deep fully-connected neural network as usual.

The rest of this paper is organized as follows. In Chapter II, we discuss the previously proposed models in the literature that also use multiple generators. We explain our proposed model of hierarchical mixture of generators in detail in Chapter III. Our experimental results on a toy two-dimensional and five real-world image data sets are given in Chapter IV. We conclude in Chapter V.

II. Combining Multiple Generators in GAN

A. Multi-agent diverse GAN

In the multi-agent diverse GAN (MADGAN) [4], there are $K \geq 2$ generators each of which labels the fake data it generates with its index. D does not learn a two-class, true vs. fake classification problem, but a $K+1$-class problem where class 0 is for true instances and classes 1 to K are the different ways of generating fake; in terms of implementation, D has K softmax outputs instead of one sigmoid output as we have for the original GAN.

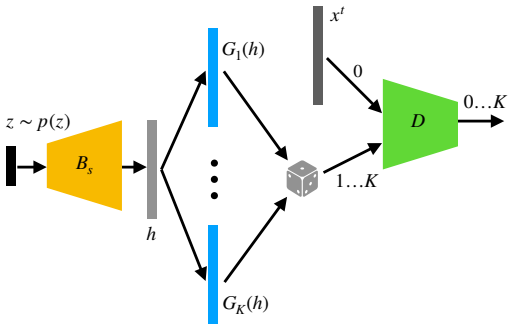


Fig. 1. Multi-agent diverse GAN (MADGAN) [4]. Given the latent z, the shared network B_s (shown in yellow) produces an intermediate representation h. From this representation, each generator G_i (shown in blue) outputs a sample, one of which is randomly chosen and fed to the discriminator D as a fake together with the index of its generator. D (shown in green) is a $K+1$-class classifier.

The model is shown in Figure 1. Given z, first a shared neural network block B_s produces an intermediate representation h. z is a low-dimensional unstructured vector, B_s contains successive deconvolution layers and generates a two-dimensional image using multiple filters in h, which is given as input to a set of generators $G_i, i = 1, \ldots, K$, one of which we choose at random. The discriminator sees either a true x^t with class code 0 or one of the generated fake with its index as the class code. The discriminator should push the different generators to different modes to solve the classification problem successfully: For correct classification, instances from the same class (i.e., fakes generated by the same generator) need to be more similar than instances from different classes (i.e., fakes generated by different generators).

More formally, for the discriminator, the objective is to

$$\max_{\phi} \quad \mathbb{E}_{x^t \sim p(x)} \left[\log D_0(x^t; \phi)\right] + \mathbb{E}_{z^t \sim p(z)} \left[\sum_{i=1}^{K} r_i^t \log D_i(G_i(z; \theta); \phi)\right] \tag{2}$$

where r_i^t is 1 if z^t is processed by generator i and 0 otherwise, and D_i denotes the output of the discriminator for class $i = 0, \ldots, K$.

In updating the weights of G_i, the objective is to

$$\min_{\theta} \mathbb{E}_{z^t \sim p(z)} \left[\log(1 - D_0(G_i(z^t; \theta)))\right] \tag{3}$$

Note that though there are multiple generators, their outputs are not combined in a cooperative manner. We do not partition $p(z)$ and use each local partition for a different generator; for any z, any of the generators can be used. It is more as if each generator produces its own interpretation of $p(z)$; instead of partitioning $p(z)$, we learn alternative generator functions for the same region in $p(z)$.

B. Mixture GAN

The mixture GAN (MGAN) [8] has some similarities with MADGAN, the main difference being that the classifier and the discriminator are separated. The discriminator is a two-class classifier as usual differentiating between true and fake examples, and there is an additional K-class classifier used only for the fake examples learning the index of the generator used.

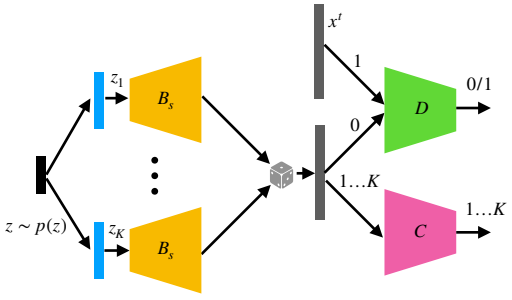


Fig. 2. Mixture GAN [8]. This is similar to MADGAN architecture with one difference being that the multiple generators are used earlier: Each generator G_i creates an abstract representation z_i which is fed to the shared block B_s to generate the output. One of the K generated fakes is selected at random and given to the discriminator; there is also an additional classifier C (shown in purple) that learns the index of the generator of a fake.

The model is shown in Figure 2. There is also the difference that the split of the generators is earlier and the shared deconvolution block B_s comes afterward. The generators $G_i, i = 1, \ldots, K$ transform z to z_i in parallel and for all, the shared B_s produces the final output. Training is formalized as a multi-task learning problem where the discriminator is trained to discriminate between the fake and the real data as usual, and at the same time, for a fake, the K-class classifier tries to predict the index of the generator that produced it.

The overall objective is defined as follows:

$$\begin{aligned}\min_{\theta,\psi}\max_{\phi}\ & \mathbb{E}_{x^t\sim p(x)}\left[\log D(x^t;\phi)\right] \\ &+ \mathbb{E}_{z^t\sim p(z)}\left[\sum_{i=1}^{K} r_i^t \log(1 - D(G_i(z^t;\theta);\phi))\right] \\ &- \mathbb{E}_{z^t\sim p(z)}\left[\sum_{i=1}^{K} r_i^t \log(C_i(G_i(z^t;\theta);\psi))\right]\end{aligned} \tag{4}$$

where C, parameterized by ψ, is the K-class classifier for the fakes whose output for class i is denoted by C_i.

C. Mixtures of experts GAN

In MEGAN [20], inspired from the mixtures of experts architecture [10], there is an additional gating model, which is also trained, that chooses among the different generators.

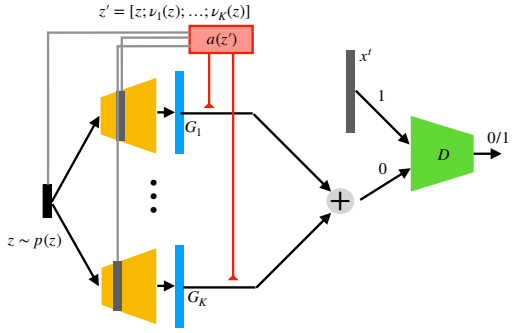


Fig. 3. Mixture of experts GAN [20]. Given z, each generator G_i outputs a sample x_i. The gating network (shown in red) selects one generated sample among all based on z and $\nu_i(z)$, which are the first layer activations of the generators. Unlike the original mixtures of experts [10], the selection is hard; only one generator is used.

The model is shown in Figure 3. In addition to the generators $G_i, i = 1, \ldots, K$, there is also a gating network that takes z and $\nu_i(z), i = 1, \ldots, K$ as its input: $\nu_i(z)$ are the first layer activations of G_i and as such are believed to provide additional information as to how best to choose the responsible generator. Then Straight-Through Gumbel Softmax is applied which only selects one expert while allowing differentiability. The discriminator is still two-class. The gating model also has its parameters that are updated together with the generators. Although all generators generate an output, it is the gating model that decides which one is to be used. Except the way the generator is written as a weighted sum of generators, the training objective is the same as Equation (1) used in the original GAN model.

Different from MADGAN and MGAN, here, $p(z)$ is partitioned into local regions that map to local regions of $p(x)$. Thanks to the gating network outputs, each generator G_i is only responsible for a local region of $p(z)$ and generates the corresponding local region of $p(x)$. However, this partitioning is hard since we let only one generator to be used. Besides, the gating network takes processed features $\nu_i(z)$ as extra inputs and this may lead to a partitioning that might be non-smooth in the z space.

III. Hierarchical Mixtures of Generators

All previous approaches use multiple generators, yet these generators do not work cooperatively, and they all train a flat set of generators. We propose the hierarchical mixture of generators that are inspired from the hierarchical mixtures of experts [11], where the generators are organized at the leaves of a tree and they cooperate as defined by the tree structure.

Let us think of a binary decision tree. The generators are at the leaves of this tree. We start from the root and at each internal node m of the tree, there is a gating function $\sigma_m(z)$ with parameters $\{v_m, v_{m_0}\}$ that calculates the probability that we take the left child:

$$\sigma_m(z) = \frac{1}{1 + \exp\left[-(v_m z + v_{m0})\right]} \tag{5}$$

$1 - \sigma_m(z)$ is the probability that we take the right child. We continue recursively until we get to the leaf nodes. If m is a leaf node, the response is given by the generator $G_m(z)$ at that leaf; if m is an internal node, its response is a weighted sum of its left and right children weighted by the gating values (Fig. 4):

$$x_m(z) = \begin{cases} G_m(z) & \text{if } m \text{ is a leaf} \\ x_m^L(z)\sigma_m(z) + x_m^R(z)(1 - \sigma_m(z)) & \text{otherwise} \end{cases} \tag{6}$$

where x_m^L and x_m^R are the responses of the left and the right children respectively, calculated recursively until we get to leaf nodes. The final output hence is a weighted average of all the generators at the leaves where the probability of a leaf is calculated using the gating values on its path; for example, in Figure 4, the weight of $G_2(z)$ is $\sigma_1(z)(1 - \sigma_2(z))$.

The generators at the leaves are simple linear models:

$$G_m(z) = W_m z + w_{m0} \tag{7}$$

Because the gating in Equation (5) is a sigmoid, we take a soft combination of generators at the leaves. This has two uses: First, we have a smooth transition from one local generator to another smoothly interpolating in between. Second, the model is differentiable and therefore given a tree structure, we can use gradient-based optimization to learn all the gating parameters $\{v_m, v_{m0}\}$ in the decision nodes and the parameters of the generators $\{W_m, w_{m0}\}$ at the leaves.

In training, we use the Wasserstein loss [1] which has been shown to work better than the likelihood-based criterion of Equation (1):

$$\min_{\theta}\max_{\phi} \quad \mathbb{E}_{x^t\sim p(x)}[D(x^t;\phi)] - \mathbb{E}_{z^t\sim p(z)}[D(G(z^t;\theta);\phi)] \tag{8}$$

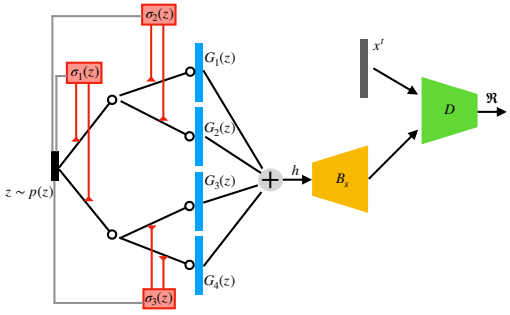


Fig. 4. Hierarchical mixture of generators (HMoG) with depth two and four generators. Each generator creates an intermediate representation from z and their average is calculated with the weights given by gating values on each path. The resulting intermediate representation h is passed through shared deconvolutional layers B_s to generate the image.

Here, D estimates the score of "trueness" for a sample, and the Wasserstein loss checks for the difference between the average scores for true samples and the generated fake samples. D, which is a regressor and not a classifier, is trained to maximize this, and G is trained to minimize it.

Note that Equation (5) defines a binary tree; we can also have a $K > 2$-ary tree by using the softmax instead of the sigmoid in the gating nodes. At the extreme, as a special case, we can have a tree of depth 1 with K generator leaves; see Figure 5. This is the (flat) mixtures of generators (MoG), which is similar to MEGAN with two differences: We keep softmax gating so the combination is soft just like the original mixture of experts model [10], and the input to gating uses only z without any extra features $\nu_i(z)$ extracted from G_i.

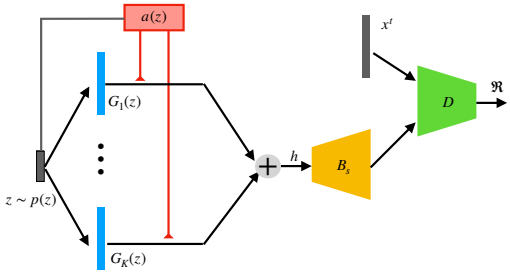
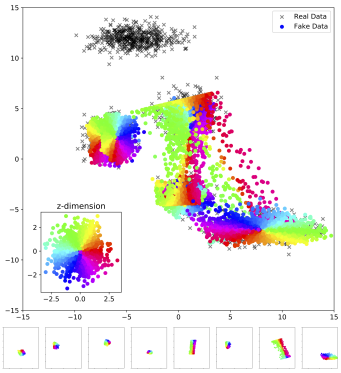


Fig. 5. Mixture of generators (MoG). Given z, each generator outputs an intermediate h. The gating unit chooses among K generators using the softmax that assigns weights that are between 0 and 1 and sum up to 1. Hence, the output is a convex combination of all the generators, which is then passed through B_s to generate the image.

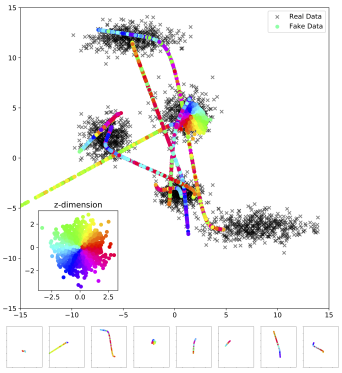
IV. Experiments

A. Results on toy data

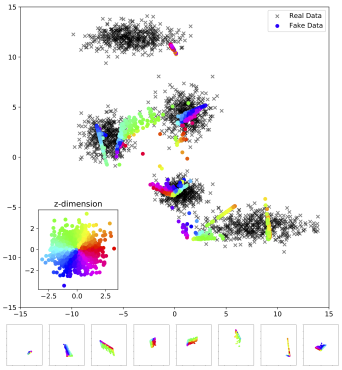
We begin with experiments on a toy two-dimensional data set sampled from a mixture of five Gaussians. The latent z is drawn from a two-dimensional Gaussian distribution with zero mean and unit variance, and we use eight generators. The data and how the generators split the data amongst themselves are shown in Figure 6 for MADGAN, MGAN, MEGAN. We color the different regions of $p(z)$ and the corresponding $p(x)$ so as to show which parts of $p(z)$ generate which part of $p(x)$; we also show in smaller plots below samples generated by individual generators similarly color-coded.



(a) MADGAN



(b) MGAN



(c) MEGAN

Fig. 6. Results using the three competing methods that also use multiple generators. In all three, in the larger figure above, the toy data set is shown with black crosses and the different colors represent different parts of $p(z)$ which is similarly color-coded in the lower left hand side. The small figures below show the part of data generated by the eight individual generators.

We see that because MADGAN and MGAN do not use any gating function, with these two, the output regions of the generators overlap with each other. Each color in z-space corresponds to eight different points in the x-space by the eight models. MEGAN does use a gating function but because the gating also uses extra $\nu_i(z)$, the output regions of generators still overlap with each other. Note that all three miss parts of

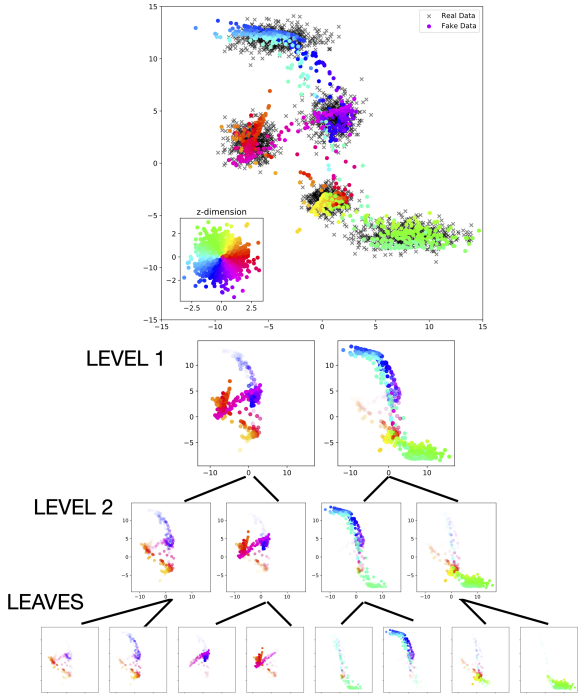


Fig. 7. Result using our proposed hierarchical mixture of generators (HMoG) with depth three and eight generators on the toy data. How the data is split over the tree at various levels and the responsibilities of the generators at the leaves are also shown.

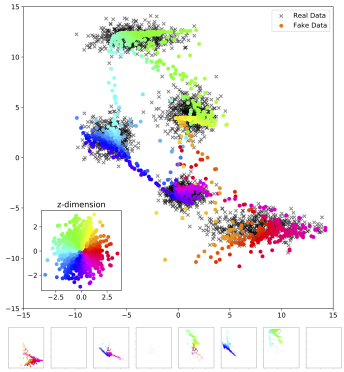


Fig. 8. Result using the flat mixture of generators (MoG) on the toy data. The small figures below show the part of data generated by each generator.

the underlying distribution; MADGAN and MEGAN miss the top component, and MGAN miss the one at the bottom.

The tree learned by our proposed hierarchical mixture of generators, HMoG, is shown in Figure 7; this is a tree of depth three that also has eight generators at its leaves. We calculate each decision node's responsibility by counting the (soft) gating values and an instance is drawn in the box of the expert having the highest responsibility. We see that the tree has learned a hierarchical soft clustering of the data with the leaves learning parts of $p(x)$ each corresponding to a part of $p(z)$. We see that this model covers the data completely and has not missed any of the components.

The results using a flat mixture of generators, MoG, is shown in Figure 8. Because the combination is soft and only depends on the input z, here too each generator operates in a local region of z. This model too learns the distribution without dropping any modes of the data; note that here, we see that some generators do more of the work with some not used at all. This we believe is the advantage of a hierarchical model which dissects the problem into two at each level, easing the problem in a divide-and-conquer fashion. The hierarchical organization also leads to discovering structure in the data.

B. Results on image data sets

We test and compare our proposed mixture models HMoG and MOG with MADGAN, MGAN, and MEGAN, on five image data sets that are widely used in the GAN literature: MNIST [16], FashionMNIST [24], UTZap50K [25], Oxford Flowers [19], and CelebA [17]. We resize MNIST and FashionMNIST data set to 32×32 and the other data sets with more detail to 64×64. All image pixels are normalized to the range $[-1, 1]$.

It is known that using a convolutional architecture for tasks that involve images increases the performance dramatically, and we incorporate transposed convolutional (also known as deconvolutional or fractionally strided convolution) layers in each model. More specifically, we use the (transposed) convolutional part of DCGAN [21] as the shared part of generators, denoted by B_s above. Instead of generating samples directly in the data domain x, each model generates an abstract representation h which is given to the shared block B_s that produces the output x. For any data set, the same B_s is used in all models.

All these variants combine multiple *local* models; we also define the fully connected (FC) model that uses a fully connected layer, which stands for the standard *distributed* alternative having one global generator, which we take as the baseline against which we compare all the localized variants.

In training HMoG, MoG, and the baseline model FC, the Wasserstein loss with gradient penalty [6] is used. For MADGAN and MGAN, we use the original likelihood-based loss; the Wasserstein loss is not applicable with these since they require D to be a classifier. MEGAN can be used with either the Wasserstein loss or the original loss; we use the original loss because it performed better in our preliminary experiments. For all methods, we used the Adam optimizer [14] with amsgrad option [22]. The learning rate is set to 0.0001 with beta values of Adam set to $(0.5, 0.999)$. The batch size is set to 128.

For each model, the training on a single NVIDIA V100 GPU took around 2 hours for MNIST and FashionMNIST sets, and 20 hours on UTZap50K, Oxford Flowers, and CelebA sets. Note that the additional time-complexity introduced by the tree structure is negligible in practice as we can compute the probability of each node on a level in parallel. This is a nice feature as we can easily scale up the model to thousands of generators with depth 8 ($2^8 = 1024$), given enough memory.

For evaluating the performance of the variants, we use the Fréchet Inception Distance (FID) [7] and the two-sample test (C2ST) [18], here, 5-nearest neighbor (5-NN) leave-one-out accuracy. Both FID and 5-NN accuracy are calculated with the activations before the softmax layer (2048-dim) of InceptionV3 [23]. Lower FID scores are better and 5-NN accuracies that are close to 50% are better. All models are run five times with different random seeds, and we report the mean and standard deviations.

For flat models, we experiment with 4, 8, 16, and 32 generators, which for the hierarchical model translates to trees of depth 2, 3, 4, and 5. We also report the parameter count of each model; these do not include the shared deconvolution block used in all models.

Our experimental results on the five data sets are shown in Figure 9. We see that in terms of the FID score, both of our proposed MoG and HMoG outperform other approaches. We also see that MADGAN and MGAN perform worse than the baseline FC; only on MNIST, MADGAN performs better than the baseline. This might suggest that forcing discriminator to classify generators may not always work, which is the idea behind MADGAN and MGAN. On the other hand, MEGAN seems to perform on par with the baseline, sometimes even better. Note that unlike MADGAN and MGAN, MEGAN uses a gating function to select among its generators. This hints at the importance of training different generators in different input regions and combining them based on the input, instead of relying on the discriminator to force multiple generators to different modes.

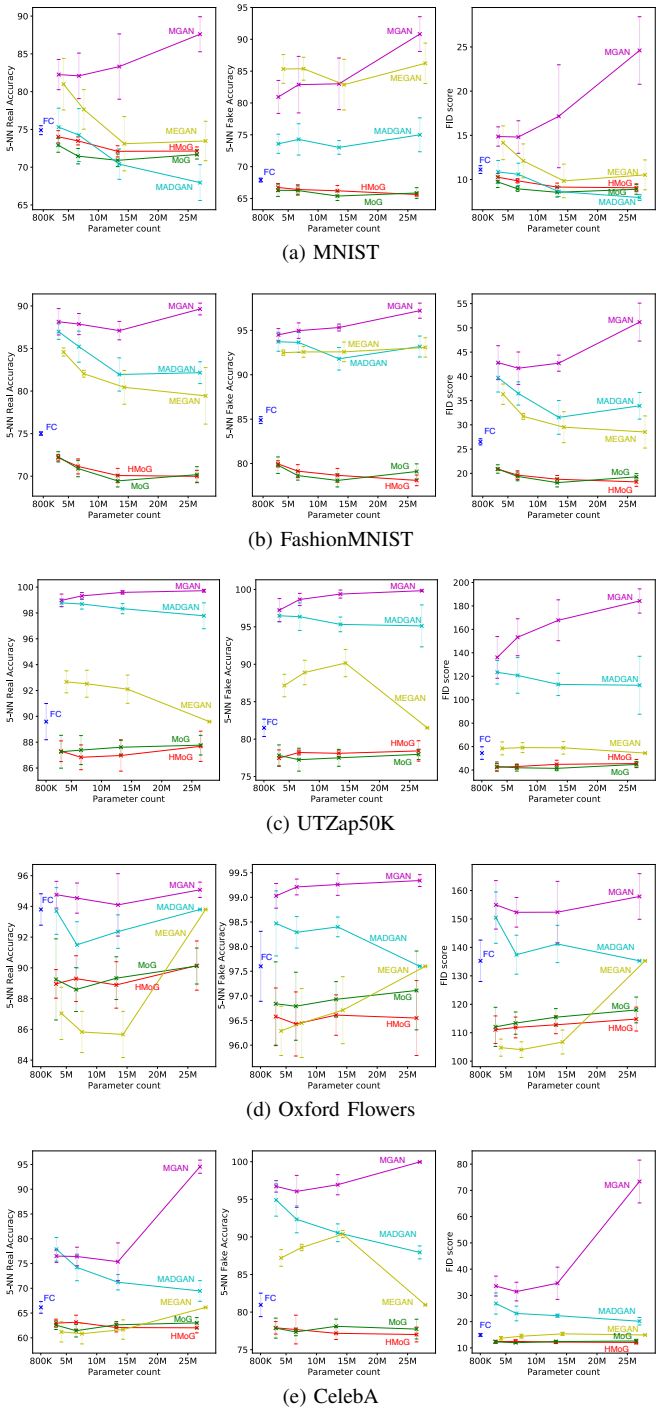
If we compare our mixture of experts formulation (MoG) with MEGAN, we see that our model gets better results in terms of FID scores and 5-NN accuracies. As opposed to MEGAN, our mixture of generators is a soft cooperative one. The input to the gating model is only the latent z, which also reduces the number of parameters significantly.

Some samples generated from HMoG with depth four are shown in Figure 10. For the sampling procedure, we randomly draw $\{z_i\}$ and disregard the least likely 25% percent to get rid of possible outliers [2]. A visual inspection of these also show that HMoG is able to generate realistic and diverse samples on all data sets.

C. Interpretability

Because both MoG and HMoG use a soft combination, we can check whether there is any correlation between the outputs of the local generators. For the flat MoG, the probability that a local model is used is given by the softmax gating; for the HMoG, it is the product of all the (binary) gatings on the path to the root. We calculate the correlation between these probabilities for pairs of local models for the case of 16 generators (or a tree of depth 4) on the CelebA data set. The 16×16 correlation matrices for both are shown color-coded in Figure 11.

We see that with the flat MoG, correlations are randomly scattered. In HMoG however, we see that the correlations are gathered around the diagonal; we can see spectral squares of



(a) MNIST

(b) FashionMNIST

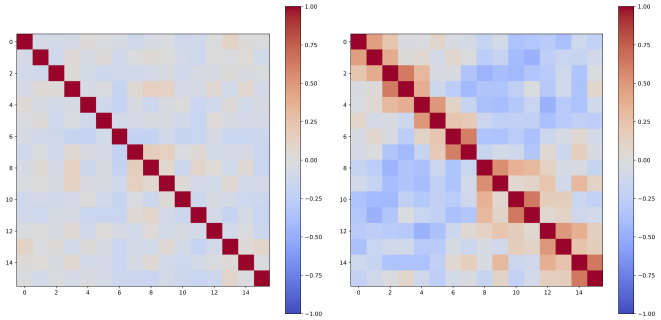
(c) UTZap50K

(d) Oxford Flowers

(e) CelebA

Fig. 9. FID scores and 5-NN accuracies of all tested models on the five data sets for different number of generators. These are average and one standard deviation error bars of five runs. The x-axis is the parameter count (not including B_s used in all) and the y-axis represents 5-NN real accuracy, 5-NN fake accuracy, or the FID score, respectively. Lower FID and 5-NN scores closer to 50% are better.

Fig. 10. Samples generated by the hierarchical mixture of generators (HMoG) with depth four (and 16 generators) on the five data sets.



(a) MoG (b) HMoG

Fig. 11. Correlation matrices of gating values of generators for MoG and HMoG with 16 generators. With MoG, there is no apparent correlation; with HMoG, we see that generators that are closer in the tree (in the same subtree) are used together, which imply a semantic correlation.

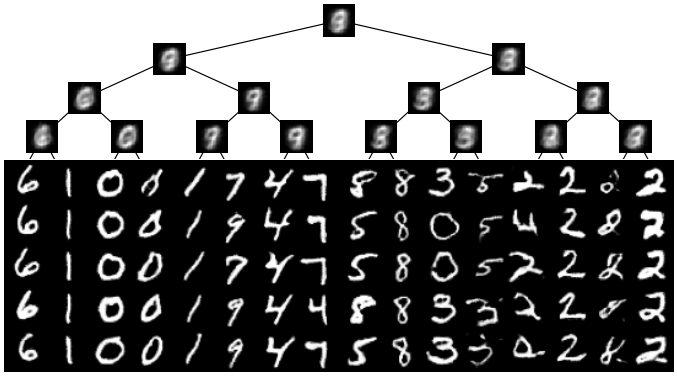


Fig. 12. On MNIST using HMoG, the average response of each internal node in the tree hierarchy is shown. For each leaf, five random samples are shown that have the highest probability of being generated in there.

sizes 2×2 and 4×4 corresponding to subtrees, which is an implication that generators that have the same ancestor on the second or the third level of the tree are frequently used together indicating that they learn semantically correlated samples.

In Figures 12 and 13, the average responses of decision nodes in the tree are visualized by taking the weighted average of the generated samples on MNIST and CelebA respectively. For a given node, weights are found by multiplying the gating probabilities along the path to the node. At the bottom of the tree under each leaf, we show five random samples generated from the corresponding generator. To find these, we sample random 10,000 z vectors and select the top five most likely for each generator. Here, the most likely point for a generator is the point which maximizes the probability that the corresponding leaf is chosen. We see the data set mean at the top root, and as we go down the tree the blurriness decreases and each node becomes more specialized to a specific region of $p(x)$. We see in Figure 12 that digits that are similar in shape are generated by leaves that are nearby in the tree. For CelebA too, as we see in Figure 13, we see that the examples are distributed over the leaves in terms of similarity in orientation, color, or background.

We believe that this interpretability is the advantage of the HMoG model over the MoG model, as well as other approaches that train a flat set of generators. As in soft hierarchical clustering, the division at each level, which may be interpreted as an architectural inductive bias, lets us view the data in different levels of granularity and understand the decisive features of the data through a divide-and-conquer type of approach.

V. Conclusions

We propose the hierarchical mixture of generators, HMoG, and a special case, MoG, which is a flat mixture of generators. There are GAN variants in the literature that also combine multiple generators but they are limited in the way they force the generators to different modes. Our formulation is the first to our knowledge that learns a cooperative mixture of generators, either organized in a flat manner or hierarchically.

An important advantage of the hierarchical model is its interpretability. Since it is a tree architecture, we can make a post-hoc analysis of the learned tree to gain insight about the data. At each level of the tree, nodes can be seen as clusters, or modes, in different levels of granularity, where as we go down

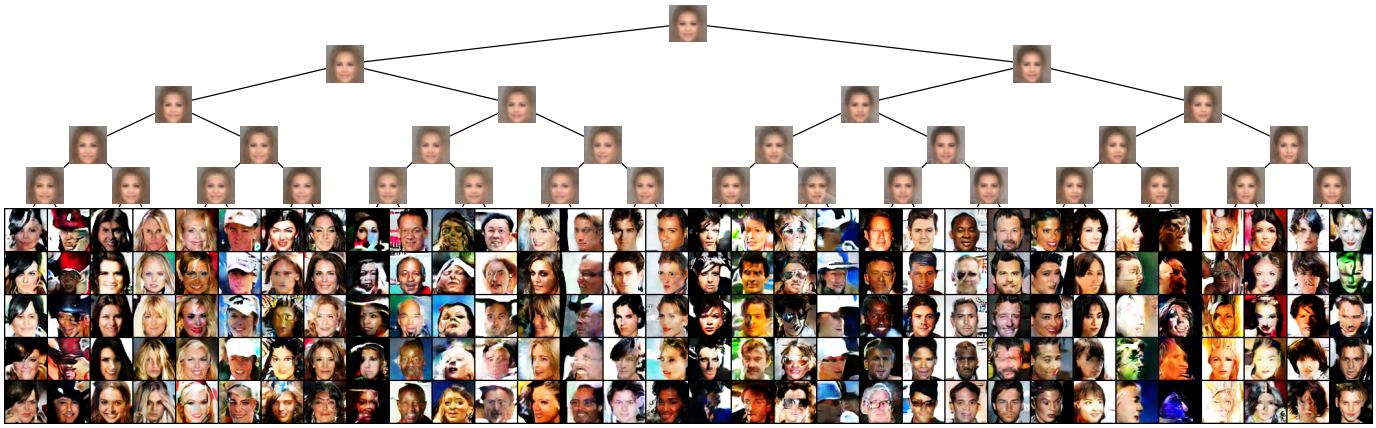


Fig. 13. On Celeb using HMoG, the average response of each internal node in the tree hierarchy is shown. For each leaf, five samples are shown that have the highest probability of being generated in there.

the tree, clusters get more local. At the same time, splits are soft and what the tree learns is a hierarchical soft clustering of the data. In the generative setting that we have here, the leaves are generators each responsible from generating one local cluster,

Our experimental results on five data sets show that the proposed models can generate samples that are realistic and diverse. Our proposed models have better FID score and 5-NN accuracy with lower variance when compared with other methods that incorporate multiple generators as well as the fully-connected standard GAN implementation.

ACKNOWLEDGEMENTS

This work is partially supported by Boğaziçi University Research Funds with Grant Number 18A01P7. The numerical calculations reported in this work were partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

REFERENCES

[1] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International Conference on Machine Learning 34*, 2017, pp. 214–223.

[2] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[3] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: an overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[4] A. Ghosh, V. Kulharia, V. P. Namboodiri, P. H. Torr, and P. K. Dokania, "Multi-agent diverse generative adversarial networks," in *IEEE Conference on Computer Vision and Pattern Recognition 31*, 2018, pp. 8513–8521.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Neural Information Processing Systems 27*, 2014, pp. 2672–2680.

[6] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Neural Information Processing Systems 30*, 2017, pp. 5767–5777.

[7] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a nash equilibrium," *arXiv preprint arXiv:1706.08500*, 2017.

[8] Q. Hoang, T. D. Nguyen, T. Le, and D. Phung, "MGAN: Training generative adversarial nets with multiple generators," in *International Conference on Learning Representations 6*, 2018.

[9] Y. Hong, U. Hwang, J. Yoo, and S. Yoon, "How generative adversarial networks and their variants work: an overview," *ACM Computing Surveys*, vol. 52, no. 1, p. 10, 2019.

[10] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[11] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Computation*, vol. 6, no. 2, pp. 181–214, 1994.

[12] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[13] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *IEEE Conference on Computer Vision and Pattern Recognition 32*, 2019, pp. 4401–4410.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[15] K. Kurach, M. Lucic, X. Zhai, M. Michalski, and S. Gelly, "The GAN landscape: Losses, architectures, regularization, and normalization," *arXiv preprint arXiv:1807.04720*, 2018.

[16] Y. LeCun, "The MNIST database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*, 1998.

[17] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *IEEE International Conference on Computer Vision 15*, 2015, pp. 3730–3738.

[18] D. Lopez-Paz and M. Oquab, "Revisiting classifier two-sample tests," *arXiv preprint arXiv:1610.06545*, 2016.

[19] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Indian Conference on Computer Vision, Graphics & Image Processing 6*. IEEE, 2008, pp. 722–729.

[20] D. K. Park, S. Yoo, H. Bahng, J. Choo, and N. Park, "MEGAN: mixture of experts of generative adversarial networks for multimodal image generation," *arXiv preprint arXiv:1805.02481*, 2018.

[21] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[22] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019.

[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *IEEE Conference on Computer Vision and Pattern Recognition 29*, 2016, pp. 2818–2826.

[24] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[25] A. Yu and K. Grauman, "Fine-grained visual comparisons with local learning," in *IEEE Conference on Computer Vision and Pattern Recognition 27*, 2014, pp. 192–199.