THE DEEP GENERATIVE DECODER: USING MAP ESTIMATES OF REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

A deep generative model is characterized by a representation space, its distribution, and a neural network mapping the representation to a distribution over vectors in feature space. Common methods such as variational autoencoders (VAEs) apply variational inference for training the neural network, but optimizing these models is often non-trivial. The encoder adds to the complexity of the model and introduces an amortization gap and the quality of the variational approximation is usually unknown. Additionally, the balance of the loss terms of the objective function heavily influences performance. Therefore, we argue that it is worthwhile to investigate a much simpler approximation which finds representations and their distribution by maximizing the model likelihood via back-propagation. In this approach, there is no encoder, and we therefore call it a Deep Generative Decoder (DGD). Using the CIFAR10 data set, we show that the DGD is easier and faster to optimize than the VAE, achieves more consistent low reconstruction errors of test data, and alleviates the problem of balancing the reconstruction and distribution loss terms. Although the model in its simple form cannot compete with state-of-the-art image generation approaches, it obtains better image generation scores than the variational approach on the CIFAR10 data. We demonstrate on MNIST data how the use of a Gaussian mixture with priors can lead to a clear separation of classes in a 2D representation space, and how the DGD can be used with labels to obtain a supervised representation.

1 INTRODUCTION

Recently, there has been a lot of focus on (deep) generative neural networks, such as variational autoencoders (VAEs) (Kingma & Welling, 2013) and generative adversarial networks (GANs) (Goodfellow et al., 2014). These models consist of a distribution over representations (z) and a neural network decoder, which maps from representation space to sample space (x) directly or gives a distribution $P(x|z)$ in sample space. In many of these models, the task of the decoder is to transform an arbitrary distribution over representation space (normally a Gaussian) into a proper distribution over sample space. The models differ in their objective functions and how they are trained, and there are several different variants (Tschannen et al., 2018; Bond-Taylor et al., 2021).

VAEs use maximum likelihood estimation of network parameters. The inference, however, is intractable due to an integral over the latent space (representations) and is therefore done by an approximate approach maximizing a variational lower bound of the likelihood (the "ELBO"). Firstly, the approximation is not always accurate (see e.g. Cremer et al. (2018)). Secondly, VAEs require an encoder, adding a whole new set of modelling choices to be made. Thirdly, the model estimation is more complicated than that of regular neural networks, requiring the "reparametrization trick" and choice of parameters are often found to be difficult. This complexity also makes these models less accessible for new-comers in the field.

A much simpler approach is to maximize the likelihood over representations of the training data and parameters at the same time (Han et al., 2017; Bojanowski et al., 2018). In this approach, the representations are treated like the other model parameters and since gradients can be derived exactly, the model can be trained using standard gradient descent. It is probably not possible to decide theoretically whether the approximate variational approach to optimizing the likelihood (VAE) is better than the exact approach of optimizing an approximation of the likelihood. The answer may

well depend on the learning problem. Even if the variational approach turns out to be best, we find it worthwhile to investigate the simpler approach.

Here we present a Bayesian formulation with maximum a posteriori (MAP) estimation of both weights *and* representations, which is the same basic idea as alternating back-propagation (Han et al., 2017). The model consists of a decoder (or generator), such as a feed-forward neural network, and a distribution over representations, which may or may not have learnable parameters. Estimating the representations of the training samples as well as the parameters of the decoder is straightforward and it can be done by gradient descent as in a non-probabilistic formulation (Schuster & Krogh, 2021). The basic formulation of our model contains no encoder, but we discuss how to add an encoder after the model has been trained and discuss the close analogy to VAEs.

Our emphasis is on a model with a parametrized distribution over representations, which can add structure to representations. This is motivated by the intuition that representations are meaningful and are likely to cluster samples with different properties. The special case of a fixed distribution over representations corresponds to the original VAE (Kingma & Welling, 2013). The distribution of VAEs can also be learned and expanded to Gaussian mixtures, which can lead to improved generalization and clustering as shown in Dilokthanakul et al. (2017) and Guo et al. (2020). However, the inference is still intractable and poses the same challenges for learning. Modelling of the probability distribution as a mixture of Gaussians has also been applied instead of classic random sample generation to generative adversarial networks (GANs), which usually loose in performance when the data set is highly diverse (Najar et al., 2019; Ben-Yosef & Weinshall, 2018).

The proposed model is tested on the CIFAR10 (Krizhevsky et al., 2009) data and compared to VAEs. We show that the DGD is easier and faster to optimize than the VAE, achieves more consistent low reconstruction errors of test data, and alleviates the problem of balancing the reconstruction and distribution loss terms. Additionally, we test a Gaussian mixture model (GMM) with priors on the MNIST (Deng, 2012) data and also show how the model can be used in a supervised manner with labels to learn label-specific mixture components.

2 THE MODEL

Consider a model with observed variable x and continuous latent variable z, which we also refer to as a representation. Usually, the x-space (or sample space) would be of a higher dimension than the z-space (or representation space), so the model would give a low-dimensional representation of data. A neural network (the decoder) with parameters θ defines a conditional distribution in sample space, $P(x|z,\theta)$. In the simplest cases the neural network would give the mean vector $f_\theta(z)$ of a univariate Gaussian with a small fixed variance for continuous outputs, or parameters of independent Bernoulli distributions for the discrete case.

We assume a distribution over representation space, $P(z|\phi)$, with parameters ϕ. The parameters may be fixed, in which case the distribution is not trained, making the model similar to a standard VAE with a fixed Gaussian distribution over representations. It can also have adjustable parameters, such as a mixture of univariate Gaussians with trainable means and mixture coefficients, but essentially any parametrized distribution can be used. When introducing priors over parameters, the joint probability of everything becomes

$$P(x,z,\phi,\theta) = P(x,z|\phi,\theta)P(\phi)P(\theta) = P(x|z,\theta)P(z|\phi)P(\theta)P(\phi). \tag{1}$$

Again, $P(x|z,\theta)$ represents the decoder neural network, $P(z|\phi)$ the distribution over representation space and we add priors over the decoder parameters (neural network weights), $P(\theta)$, and over the parameters in the distribution over representation space, $P(\phi)$.

If the distribution over latent space is fixed, we would obtain an even simpler model where the parameters ϕ can be left out, $P(x,z,\theta) = P(x|z,\theta)P(z)P(\theta)$. This is essentially identical to the likelihood in a standard VAE, except for the prior over θ.

The idea in the following is to treat representations z for the training data as model parameters. In Han et al. (2017) it is called "explaining away inference". It is similar to what we do in manifold learning or non-probabilistic representation learning.

2.1 MODEL ESTIMATION

Given some data for estimation (the training set), we need to deal with the unknowns. Ideally, we might want to integrate over all parameters z, ϕ, and θ to obtain their posterior mean estimates, or even better, calculate their full posterior distribution. This is a big computational challenge for the neural network parameters (θ) and therefore we normally use the maximum instead, the MAP or maximum likelihood estimate. The latent variable z, however, is typically integrated over. This leads to all the inconveniences of an intractable partition function and thus one has to use approximation or sampling. We choose to use MAP estimation for all the unknowns, including z. Although it may not be as good as integrating over z, it may still be as good or better than an approximation of the integral. Additionally, this approach turns out to be very simple and tractable.

For a data set $X = \{x^1, x^2, \ldots, x^N\}$ with N samples we thus want to find representations $Z = \{z^1, z^2, \ldots, z^N\}$ and parameters ϕ, θ that maximize $P(Z, \phi, \theta|X)$:

$$\underset{Z,\phi,\theta}{\operatorname{argmax}} P(Z, \phi, \theta|X) = \underset{Z,\phi,\theta}{\operatorname{argmax}} P(X, Z, \phi, \theta)/P(X) = \underset{Z,\phi,\theta}{\operatorname{argmax}} P(X, Z, \phi, \theta). \tag{2}$$

If we assume independence among training samples, the log of this joint probability can be obtained from (1), with k indexing the samples

$$\log P(X, Z, \phi, \theta) = \sum_k \big(\log P(x^k|z^k, \theta) + \log P(z^k|\phi)\big) + \log P(\phi) + \log p(\theta). \tag{3}$$

This is the quantity we want to maximize with respect to *both* the representations for all samples and the parameters of the decoder.

The maximization can be done via gradient descent starting from random parameters ϕ, θ as usual *and* randomly initialized representations. We can use automated differentiation when implementing the model. However, it is worth noting that the gradient wrt the z^ks in the first term of (3) becomes a back-propagated error just as in a gradient wrt a weight. If the representations are constrained, e.g. $0 \leq z \leq 1$, one can reparametrize $z = g(h)$ and instead do gradient descent in h.

The training may proceed like this: Initially, a random representation (an m-dimensional vector) is generated for each training sample and the parameters of the decoder and representation distribution are initialized. Then gradient descent is performed on all parameters ϕ, θ, Z. To spell it out, these steps in the gradient descent are repeated: 1) One step of back-propagation for the decoder (θ), 2) One step of gradient descent on the representations using the back-propagated errors from step 1 and the gradient with respect to $P(z|\phi)$, and 3) One step of gradient descent on the representation distribution parameters, ϕ (this step is not used if the distribution over latent space is fixed). One can use mini-batches and other tricks from neural network training.

The optimization can make use of alternative methods. In Han et al. (2017) the approach called "alternating back-propagation" is suggested to use Langevin dynamics as an alternative for the representations, where representations are sampled several times starting from the current value. We have found that the simple gradient descent is fast, straight-forward and easy to implement using automated gradient calculation available in modern neural network libraries. It is almost identical to the non-probabilistic case (Schuster & Krogh, 2021). The main difference is that we have gradients from $P(z|\phi)$ and may estimate the parameters ϕ for the distribution over z. If we use a Gaussian mixture in representation space, it may be possible to perform more efficient optimization, but because we want to be able to use any form of distribution, we have not pursued this further.

2.2 ADDING AN ENCODER

There is no encoder in the model, and it strictly speaking is not needed. For a new sample one can run the MAP optimization described above with all other parameters fixed aiming to find the most likely representation. We have used it in many of the experiments in this paper, suggesting that it often returns a good solution. However, complex models are likely to get stuck in local maxima and thus will not always find the optimal solution. Since we are using a random start point for z, the procedure is non-deterministic.

It may be desirable to have an encoder to avoid this non-deterministic iterative optimization. There are (at least) two approaches to this that use the trained (and fixed) generative model to generate training samples for the encoder.

The simplest is to train a deterministic encoder neural network to predict an approximation to the MAP estimate of the representation of an input. For training, we can generate (x, z) samples from the generative model, by first sampling z from $P(z)$ and then x from $P(x|z)$. The prediction of the encoder can also function as a method to provide a good starting point for MAP optimization by gradient descent. This approach is in the simplistic spirit of this work.

A more stringent approach is to train an encoder to estimate the posterior $P(z|x)$ using a probabilistic neural network that gives parameters for a distribution in representation space. It could for instance output the parameters for a Gaussian given an input x. For training, we can generate examples as above. This type of encoder giving a conditional distribution over representations is analogous to the original formulation of the VAE (Kingma & Welling, 2013). In the VAE, however, encoder and decoder are trained simultaneously with the decoder providing the variational distribution, which is sometimes mistakenly assumed to be equal to $P(z|x)$. Both approaches share the risk that the output distribution of the encoder is a bad match for the true posterior and to get a better estimate of the posterior, it can instead be used as a starting point for importance sampling (Burda et al., 2016).

For both types of encoders, it is possible to generate infinitely many (x, z) samples from the generative model. In fact, online training can be used, in which a new set of input/output samples are generated for each mini-batch. The encoder can therefore – in principle – be of very high complexity and be trained to arbitrary precision. Note, however, that we are sampling from $P(x)$ as given by the decoder and therefore an encoder of high complexity is likely to perform poorly given out-of-distribution (OOD) samples (Schuster & Krogh, 2021).

2.3 Comparison to the Variational Autoencoder

We have already highlighted the close similarity to the VAE, which uses so-called amortized variational inference. The "amortization gap" between the variational distribution and the actual posterior of z (Cremer et al., 2018) is determined by the encoder. If the encoder is very expressive, this gap can be small, but at the danger of over-fitting the encoder. If the encoder is less complex the approximation error may be more severe. The encoder can instead be used to give a start point for sampling, to limit the error (Kim et al., 2018; Krishnan et al., 2018; Marino et al., 2018).

The Generative Decoder is not amortized and avoids any additional error from the encoder. This comes at the cost of another type of approximation error related to using the MAP estimate rather than integrating over the latent variable. It is possible to use the MAP estimate as a starting point for sampling, just as in the VAE in which case the two models would become very similar: they just use different methods for initializing the sampling.

One problem with the generative decoder stems from the scaling invariance of the representations relative to the weights of the first layer. If the representation space is unbounded, the representations and the first layer weights can be scaled oppositely ($\vec{z} \cdot \vec{w} = (\vec{z}a) \cdot (\vec{w}/a)$ for any constant a) without affecting the result. If the distribution over representation space is a fixed Gaussian, one can maximize the probability of representations by scaling representations to very small norm, so they are highly probable according to $P(z)$. This does not happen in the VAE, because of the Kullback-Leibler (KL) divergence term. In the DGD, this can be mitigated by choice of priors or through explicit normalization of representations to have the same variance as the distribution. If the distribution is learned, this may be less of a problem as it will adjust to the actual distribution of representations.

2.4 Extensions of the model

The advantage of the Bayesian approach is that it is possible to add priors to the parameters. If a mixture of Gaussians is used in representation space, it is natural to use a Dirichlet prior on the mixture coefficients to encourage the model to have some weight on all of them for instance. For the means, one would perhaps not like them to cluster together close to zero, and one could have a prior that encourages such behaviour. In some experiments below we used a "soft ball prior", which is flat in an m-dimensional ball and tails off with the function $r(z) = \frac{1}{1+e^{a(|z|/b-1)}}$, where b is the radius, a is a hardness and $|z|$ is the Euclidean norm of z. For the standard deviation (σ_k) of the mixture components we optimized the parameter $\log(1/\sigma_k^2)$ and used a Gaussian prior on it.

The simplicity of the model makes it easy to extend. One extension we tested is to use a form of supervised learning of the representations. Assume we use a mixture distribution on representations and that the training data have class labels (such as numbers from 0 to 9 for handwritten digits). Then we can label the components by simply assigning one or more mixture components to each class. The probability of a representation is then conditioned on the class label and we obtain a model separating labels in representation space.

3 TESTS COMPARING THE DGD TO THE VAE

3.1 FASTER MODEL SEARCH WITH GENERATIVE DECODERS

We begin our experimental analysis of the proposed approach by comparing our simplest generative decoder (DGD) to basic implementations of the VAE for the CIFAR10 data set. In the simplest setup, both DGD and VAE have a single fixed Gaussian with mean 0 and standard deviation 1 as $P(z)$. We started the comparison by searching for well-performing models in terms of test image reconstruction. We defined search spaces for both methods (see A.2), trained models in these search spaces for 50 epochs on a subset of CIFAR10 (see A.1 for details) and reported reconstruction losses on a CIFAR10 test subset. Importantly, we also included a beta factor from βVAEs, since the difficulty of balancing reconstruction loss and KL divergence is well known (Higgins et al., 2017). The resulting learning curves are shown in Figure 1 A. Even though many VAE test reconstruction loss curves converge faster, the generative decoder on average achieves lower test reconstruction losses after 50 epochs (Figure 1 B). Thus, even though best models from both DGD and VAE perform equally in terms of image reconstruction on the test subset, the DGD seems to present a more stable design and it also has a smaller search space (one degree of freedom less due to β).

3.2 NO TRADEOFF BETWEEN RECONSTRUCTION AND PRIOR LOSS

Both the DGD and the VAE aim to learn a latent representation of the data whose values are normally distributed. We therefore evaluated latent spaces based on 3 measures for the 4 most diverse out of the 10 best models for each method. We split the VAE models into VanillaVAE and βVAE. These 12 models were trained on the full CIFAR10 train set for 100 epochs and tested on the full test set. Hyperparameters and architectures for each model can be found in A.4 table 2.

Firstly, we tested normality of all representation values with the Shapiro Wilk test. The more likely the null hypothesis, the closer the test statistic is to 1. Figure 1D shows that the DGD obtains an almost perfect fit to the Gaussian, the VanillaVAE obtains an almost equally good fit, whereas the βVAE, which is shifting the reconstruction – prior loss balance towards reconstruction loss, obtains non-normally distributed representations. The DGD manages to achieve the best reconstruction and highest probability for the null hypothesis of normally-distributed representations.

Secondly, we investigated how well the representations fitted the standard Gaussian of the prior. We therefore calculated the ratios of how much of the standard Gaussian's $N(0, 1)$ space is used (b in $bG = L \cap G$) and how much of the latent space is covered by $N(0, 1)$ (a in $aL = L \cap G$), see Figure 1D. Values were rounded to 2 decimal points for this purpose. The only method achieving near-perfect overlap with the standard Gaussian is the VanillaVAE, with both ratios near 1. While DGD representations do not venture outside the standard Gaussian space, they are distributed according to a Gaussian with smaller scale and only cover 20% of the standard Gaussian space, which can lead to out-of-distribution representations when sampling from $N(0, 1)$ for image generation. This is most likely due to the scale invariance of the representations and the first layer of weights. Another explanation for this low-variance space could be the large latent dimensions of the best performing models, giving the representation "too much space" allowing the objective function to maximize the posterior probability by reducing the variance of the Gaussian. We investigated this relation and show in supplementary Figure 5 that the latent space's standard deviation decreases with increasing latent dimension for different data sets. This may also cause the slower convergence of the models in Figure 1 A.

Thirdly, we compared the conservation of CIFAR10 class distances in representation space. We calculated euclidean distances between class means for representation and sample space and report the Pearson correlation coefficients between class distances of original data and representations. Here, correlations are highest for the DGDs, closely followed by the VanillaVAEs. The results are

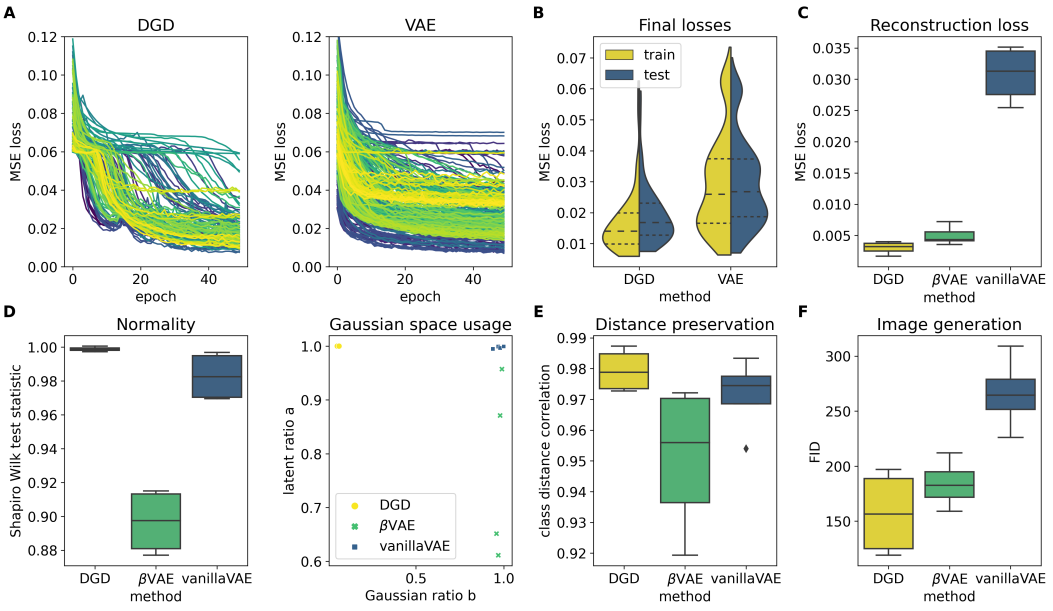

Figure 1: **Comparison of DGD and VAE methods in their simplest setups on CIFAR10. A** Model search test learning curves of single Gaussian DGD (left) and VAE (right) trained on a CIFAR10 subset (100 samples per class) for 50 epochs. Colors indicate individual models. **B** Violin plot of final test losses from models in A. Color-split shows train and test losses. Dashed lines indicate mean and standard deviations. **C-F** Reports on 4 out of 10 best models from A for each method. VAEs are split into VanillaVAE and βVAE. Model information and results are shown in supplementaries A.4. **C** Final test reconstruction losses (MSE). **D** Latent space evaluation relative to the prior Gaussian distribution. The left presents the normality statistics from the Shapiro Wilk test. Values close to 1 indicate no rejection of the null hypothesis. The right plots the ratio of latent space values overlapping with values from a standard Gaussian (a in $aL = L \cap G$) against the ratio of Gaussian space covered by the unique latent values (b in $bG = L \cap G$). L represents latent space, G Gaussian space. **E** Distance preservation in latent space presented as correlation coefficients between class-wise euclidean distances of latent class means and input data means. **F** FID scores as image generation performance metric.

shown in Figure 1E. Altogether, we see that while balancing the objectives of VAEs is not trivial and can result in a reconstruction-prior tradeoff, DGDs easily achieve good reconstruction and normally-distributed representations at the same time.

Finally, we calculated FID scores (introduced by Heusel et al. (2018) and implemented for Pytorch in Seitzer (2020)) for image generation by the models, Figure 1F. All the models selected for reconstruction performance fall short when it comes to image generation with FID scores all exceeding 114 and not reaching commonly reported scores of just above 100 for VAEs (e.g. in Bond-Taylor et al. (2021); Parmar et al. (2020)). This is likely due to the large latent dimensions of these models, insufficient training time and the insufficient coverage of the sampling space for the DGD. On average, however, the DGD still achieves lower FID scores compared to both VAE implementations.

Altogether, this shows that it is easier to find a well-performing simple DGD (in terms of image reconstruction) than a VAE and that the simple DGD at least converges much faster towards both good image reconstruction and a Gaussian latent space.

3.3 Improved Image Generation

Since the image generation capabilities of the models optimized for test reconstruction are anything but optimal, we set out to compare image generation performance of the DGD and VAE based on the well-known and reliable DCGAN architecture (Radford et al., 2016). Implementation and

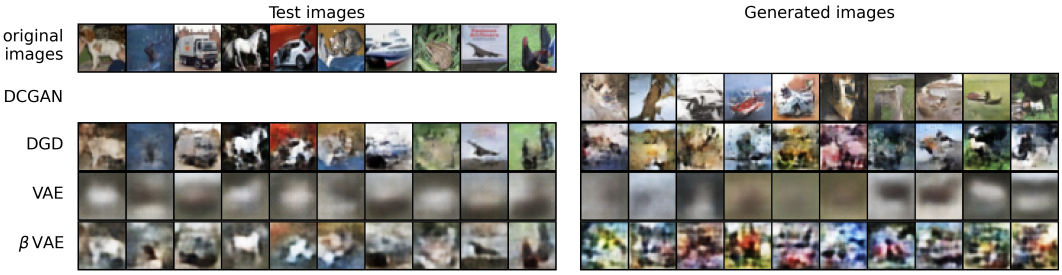

Figure 2: **Test image reconstructions and generated images from DCGAN, DGD and VAE.** Each image class is represented by its first sample in the CIFAR10 test split. All models were trained for 500 epochs and saved every 50 epochs. The best stages were selected based on test reconstruction loss and quality of generated images. These stages are indicated in table 1. Generated images were achieved by randomly sampling from a standard Gaussian and fed into the decoder/generator modules.

Table 1: Image generation on CIFAR10

Model	**Train epochs**	**Test MSE loss**	**FID**
DCGAN	50	NA	35.84
DGD	500	0.0052	90.77
VAE	500	0.0293	322.64
βVAE	100	0.0080	150.37

interpretation for DGD and VAE can be found in A.5. In Figure 2, we show CIFAR10 test image reconstructions next to randomly generated images from DCGAN, DGD, VAE and βVAE. We can see that while the VAE achieves better image reconstructions with decreasing β, generated images remain blurry. Our DGD model, however, generates sharper and more contrastive images, similar to those generated by DCGAN. This is reflected in the FID scores in table 1. Of course none of these models achieve recognizable images at this point and fall short compared to state-of-the-art image generators. We also concede that the VAEs reported here surely do not represent the best this method could do under these circumstances and that we are not experts in the field of VAEs. Nonetheless, our approach does offer an advantage compared to the VAE in terms of simplicity and basal performance, and may unite desirable features from both VAEs and GANs. Even though the fixed single standard Gaussian as a prior of the latent space is not sufficient to smoothly cluster representations and thus enable image generation of specific classes (see supplementary Figure 6), we aim at expanding this method to include GMMs for better structuring of the latent space.

4 Testing extensions of the model

4.1 Using a mixture distribution with priors

We tested the use of priors on a Gaussian mixture model with MNIST data in order to visualize a two-dimensional representation space. We used a soft ball prior on the means of the mixture components with a fixed radius of 10, a uniform Dirichlet prior on the mixture coefficients, and a Gaussian prior on $\log(1/\sigma^2)$. For the last prior influencing the variance of the means, we ended up using a mean of 2 and a standard deviation of 2 after testing combinations of 0, 1, and 2 for the means and 1, 2, and 3 for the standard deviations.

The architecture of the decoder was fixed with a two-dimensional representation fed into a first layer of 30 hidden units, followed by a second layer of hidden units. This is succeeded by two layers with 4x4 transposed convolutions with 64 and 32 channels, respectively, and ending in a 28x28 image. All activations are ReLU except the output layer, which uses a sigmoid. The images are binarized. The models were trained on a random sample of 5000 images from MNIST for 100 epochs.

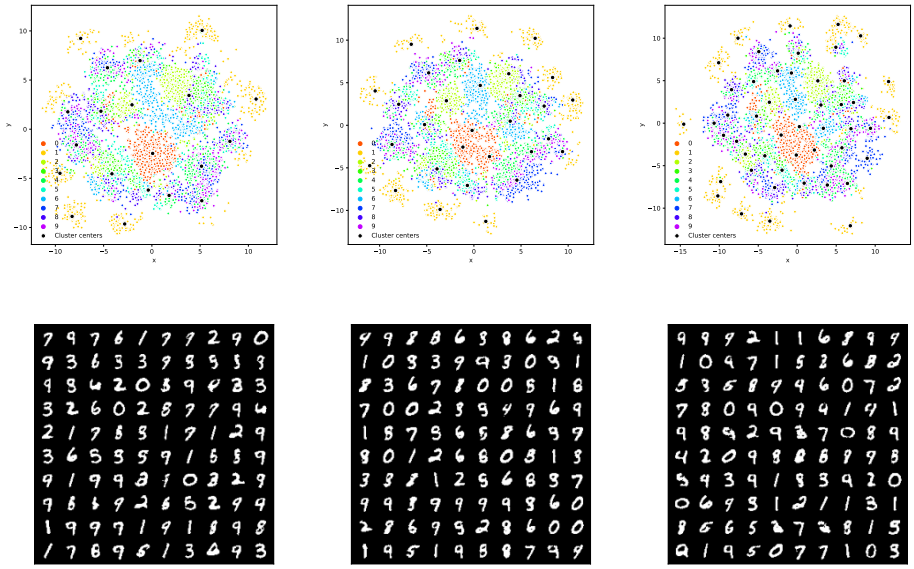

Figure 3: **Representation space and samples from models trained with 20, 30, and 50 mixture components.** The top row shows the two-dimensional representation space with the learned representations of the training samples and the means of the mixture components. Colors indicate the class (numbers from 0 to 9). The bottom row shows 100 randomly sampled images from each model. The models only differ in the number of components in the GMM.

Figure 3 shows the resulting 2D representation spaces for models with different number of Gaussian mixture components. The images are very stable showing reasonably clear clusters for the different numbers. Note the effect of the soft ball prior, which more or less confines the mixture components to a circle of radius 10.

4.2 Using labelled mixture components

Finally, a model with 10 labelled mixture components was trained on MNIST. A label, which in this case is a number from 0 to 9, was assigned to each component. During training the image labels were then used to choose the corresponding component. The model was otherwise identical to the one described above, and so were all training parameters.

In Figure 4 the resulting representation space is shown, along with images from the mixture component means and randomly sampled images. The model learned a more or less perfect clustering in 2D showing similar numbers close to each other (e.g. 3,8,5 in the upper left).

Although the results are perhaps not surprising, it shows the flexibility of the model and the ease of extending it. In our future research efforts, we will use these ideas to develop methods for partially labelled data and models that can obtain a similar clear separation of labels without supervision.

5 Conclusion

In this small study, we set out to investigate MAP estimation of representations as an alternative to the commonly used variational inference. As shown in previous work, the encoder is not needed to learn a representation of data and its mapping to sample space (Schuster & Krogh, 2021). Here, we showed that this is also the case for probabilistic representations. Compared to the VAE, the generative decoder provides a much simpler method, which in our tests ferforms as well or better than the VAE and which is much easier to implement. Even though the simple DGD presented here cannot match performance of the complex and highly advanced models based on GANs, Transformers and

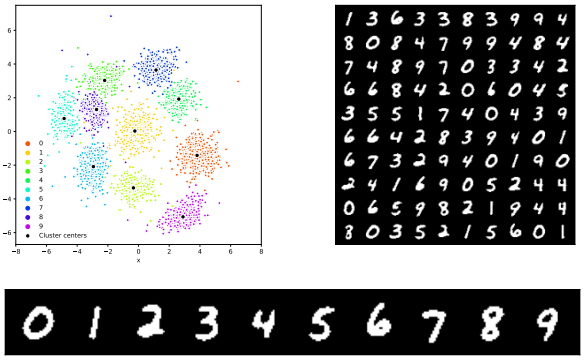

Figure 4: **A model trained with labelled mixture components.** The top shows the learned representations of the training data (left) and 100 randomly sampled images (right). The bottom shows the output images of the means of the mixture components. All images are binarized.

VAEs (Tseng et al., 2021; Park & Kim, 2021; Zhao et al., 2020; Jiang et al., 2021; Vahdat et al., 2021; Parmar et al., 2021; Xiao et al., 2021) in terms of image generation, we do show that it can – with little optimization – achieve much better generative results than an architecturally similar VAE on the CIFAR10 data set.

One may argue that it is possible to optimize VAEs further using for instance a different architecture of the encoder (we essentially used mirrored encoder and decoder). That is true of course, and our main aim has been to show that the DGD is just a lot easier to deal with, because you do not have to worry about the encoder.

We further show that when extending the Gaussian prior to a GMM trained on MNIST data, we can learn representations that separate the numbers quite well in a 2D representation space. By using priors on the parameters of the mixture model, it can be guided to, for example, have mixtures of low or high variance. Lastly, we have shown that a model can even be trained with labelled mixture components to learn ten very well-separated components for each number in MNIST.

In future works, we will be extending the model to partly-supervised learning. We also aim at trying to unite with ideas from manifold learning to obtain better representations. Finally, we are working on using the model with missing data, for which it has obvious advantages (Han et al., 2017).

REFERENCES

Matan Ben-Yosef and Daphna Weinshall. Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images, 2018.

Piotr Bojanowski, Armand Joulin, David Lopez-Pas, and Arthur Szlam. Optimizing the latent space of generative networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 600–609. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/bojanowski18a.html.

Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models, 2021.

Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.00519.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Con-*

ference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, pp. 1078–1086. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/cremer18a.html.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Nat Dilokthanakul, Pedro A. M. Mediano, Marta Garnelo, Matthew C. H. Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Chunsheng Guo, Jialuo Zhou, Huahua Chen, Na Ying, Jianwu Zhang, and Di Zhou. Variational autoencoder with optimizing gaussian mixture model priors. *IEEE Access*, 8:43992–44005, 2020. doi: 10.1109/ACCESS.2020.2977671.

Tian Han, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. Alternating back-propagation for generator network. In Satinder P. Singh and Shaul Markovitch (eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 1976–1984. AAAI Press, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14784.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

I. Higgins, L. Matthey, A. Pal, Christopher P. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

Yifan Jiang, Shiyu Chang, and Zhangyang Wang. Transgan: Two pure transformers can make one strong gan, and that can scale up, 2021.

Yoon Kim, Sam Wiseman, Andrew C. Miller, David A. Sontag, and Alexander M. Rush. Semi-amortized variational autoencoders. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2683–2692. PMLR, 2018. URL http://proceedings.mlr.press/v80/kim18e.html.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv*, 1312.6114 [stat.ML], December 2013.

Rahul G. Krishnan, Dawen Liang, and Matthew D. Hoffman. On the challenges of learning with inference networks on sparse, high-dimensional data. In Amos J. Storkey and Fernando Pérez-Cruz (eds.), *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, volume 84 of *Proceedings of Machine Learning Research*, pp. 143–151. PMLR, 2018. URL http://proceedings.mlr.press/v84/krishnan18a.html.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2009. URL http://www.cs.toronto.edu/~kriz/cifar.html.

Joseph Marino, Yisong Yue, and Stephan Mandt. Iterative amortized inference. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3400–3409. PMLR, 2018. URL http://proceedings.mlr.press/v80/marino18a.html.

Fatma Najar, Sami Bourouis, Nizar Bouguila, and Safya Belghith. A new hybrid discriminative/generative model using the full-covariance multivariate generalized gaussian mixture models. *Soft Computing*, 24(14):10611–10628, 2019. doi: 10.1007/s00500-019-04567-2.

Jeeseung Park and Younggeun Kim. Styleformer: Transformer based generative adversarial networks with style vector, 2021.

Gaurav Parmar, Dacheng Li, Kwonjoon Lee, and Zhuowen Tu. Dual contradistinctive generative autoencoder, 2020.

Gaurav Parmar, Dacheng Li, Kwonjoon Lee, and Zhuowen Tu. Dual contradistinctive generative autoencoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 823–832, June 2021.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

Viktoria Schuster and Anders Krogh. A manifold learning perspective on representation learning: Learning decoder and representations without an encoder, 2021.

Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.1.1.

Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning, 2018. Third workshop on Bayesian Deep Learning (NeurIPS 2018), Montreal, Canada.

Hung-Yu Tseng, Lu Jiang, Ce Liu, Ming-Hsuan Yang, and Weilong Yang. Regularizing generative adversarial networks under limited data, 2021.

Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space, 2021.

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

Zhisheng Xiao, Karsten Kreis, Jan Kautz, and Arash Vahdat. Vaebm: A symbiosis between variational autoencoders and energy-based models, 2021.

Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training, 2020.

A APPENDIX

A.1 DATA

In this work we made use of the publicly available natural image data sets MNIST (Deng, 2012), FashionMNIST (Xiao et al., 2017) and CIFAR10 (Krizhevsky et al., 2009). We used the data sets' implemented train-test splits and created class-balanced subsets for model search and supporting experiments. These subsets are created by taking every 30th and 50th image per class from FashionMNIST and CIFAR10, respectively.

A.2 SEARCH SPACE DESIGN

Architectures investigated in the search for well-performing models in terms of test reconstruction are taken from A.3. We set a random seed of 0 and used train mini-batch size 64 and test mini-batch size 16. Learning rates examined are $\{1e-2, 1e-3, 1e-4\}$ for the decoder and VAE and $\{1e-1, 1e-2, 1e-3\}$ for the representation. Capacity options are $\{8, 16, 32, 64\}$ and weight decays in the Adam optimizer are either 0 or $1e-5$. VAEs are investigated with an additional option of the β factor described in Higgins et al. (2017). The β factors applied (before normalization $b' = \frac{b \times latent_dimension}{input_dimension}$) are $\{None, 10, 1, 1e-1, 1e-2\}$. $None$ refers to the original VAE implementation without reconstruction-KL divergence balancing.

Initialization from $N(0, 1)$ and updating of representations is performed as described in Schuster & Krogh (2021). Reconstruction losses are computed as mean squared error (MSE) losses. Prior losses are represented by the log of the posterior probability densities $log\left(P(x|z, \theta) + P(z|\phi)\right)$ for DGD (we will later refer to this loss as MAP loss) and the KL divergence for VAEs.

A.3 ARCHITECTURES

The architectures investigated in the model search and comparison between DGD and VAE reported in Figure 1 are described below. They and all other code is implemented in Python using Pytorch. The inspiration for these architectures derives from Radford et al. (2016) and were used similarly in Schuster & Krogh (2021). VAE architectures are based on popular community implementations of Kingma & Welling (2013).

DGD

The standard decoder used in the CIFAR10 experiment consists of 5 2D-transposed convolutional layers. After each of the first 4 transposed convolutional layers follow a batch normalization layer and a ReLU activation function. After the last layer, Sigmoid activation is applied. The channel sizes are defined by a base we call $capacity$ (c). Channel inputs are $\{c \times 4, c \times 2, c \times 2, c, c\}$ and the output channel size is 3. Kernels defined as *(kernel_size, stride, padding)* are $\{(4, 2, 0), 3 \times (4, 2, 1), (1, 1, 0)\}$. The decoder takes an input vector of size $c \times 4$, which is reshaped to $(c \times 4, 1, 1)$ before the first layer. The output is the image tensor of size $(3, 32, 32)$.

Decoder (shallow): This is a shortened version of the standard decoder described above. It lacks the first layer block (ConvTransposed2d, BatchNorm2d, ReLU) and thus takes as input a 1D $c \times 32$ vector which is reshaped to $(c \times 2, 4, 4)$.

Decoder (linear): This decoder builds upon the shallow decoder above. It adds a fully-connected linear layer at the very beginning.

VAE

Each VAE architecture consists of two equivalent encoders with different final activation functions (ReLU for latent means and Sigmoid for latent standard deviations (sd)), followed by a decoder whose input is generated via the reparametrization trick (Kingma & Welling, 2013) from latent means and sds. The decoder modules are equivalent to decoders from the DGD architectures. The encoder architecture of the VAE is the mirrored version of the here implemented decoder. The encoder outputs are flattened to two $c \times 4$ vectors and reparametrized to the single input vector of the decoder.

VAE (shallow): The shallow decoder reconstructs the input from mirrored encoders.

VAE (shallow, encoder linear): This is the shallow VAE with fully-connected linear layers between convolutional encoder and final encoder activation functions.

VAE (shallow, linear): In addition to the linear layers in the encoder, this model makes use of the linear decoder from above.

A.4 DGD-VAE COMPARISON

The following table explains chosen architectures and hyperparameters of the models reported in Figure 1C-F.

Table 2: Architecture and hyperparameters

Model ID	**Architecture**	**Latent dimension**	**Learning rates (decoder, representation)**	**Weight decay**
DGD 1	decoder (shallow): capacity 32	1024	$1e-3, 1e-2$	0
DGD 2	decoder (shallow): capacity 16	512	$1e-3, 1e-2$	$1e-5$
DGD 3	decoder: capacity 64	256	$1e-4, 1e-2$	$1e-5$
DGD 4	decoder (linear): capacity 64	200	$1e-3, 1e-2$	$1e-5$
βVAE 1	VAE (shallow): capacity 32, beta $1e-2$	1024	$1e-3$	0
βVAE 2	VAE (shallow): capacity 16, beta $1e-2$	512	$1e-2$	0
βVAE 3	VAE (shallow): capacity 8, beta $1e-2$	256	$1e-2$	0
βVAE 4	VAE (shallow, linear): capacity 16, beta $1e-2$	200	$1e-3$	$1e-5$
VanillaVAE 1	VAE (shallow): capacity 16	512	$1e-4$	0
VanillaVAE 2	VAE (shallow, linear): capacity 32	200	$1e-3$	$1e-5$
VanillaVAE 3	VAE: capacity 32	128	$1e-3$	0
VanillaVAE 4	VAE (shallow, encoder linear): capacity 16	64	$1e-3$	$1e-5$

Table 3: Experimental results

Model ID	**MSE loss**	**FID score**	**Shapiro Wilk test statistic**	**Intersection ratio a in** $aL = L \cap G$	**Intersection ratio b in** $bG = L \cap G$	**Class distance correlation**
DGD 1	0.0017	197.35	1.0005	1.0000	0.0515	0.9737
DGD 2	0.0028	186.48	0.9972	1.0000	0.0619	0.9873
DGD 3	0.0040	119.47	0.9991	1.0000	0.0515	0.9840
DGD 4	0.0037	126.54	0.9984	1.0000	0.0619	0.9728
βVAE 1	0.0036	175.55	0.9127	0.9573	0.9897	0.9421
βVAE 2	0.0044	189.00	0.8770	0.8711	0.9794	0.9697
βVAE 3	0.0056	212.00	0.9150	0.6514	0.9588	0.9721
βVAE 4	0.0072	159.15	0.8822	0.6113	0.9691	0.9193
VanillaVAE 1	0.0343	260.26	0.9968	0.9995	1.0000	0.9755
VanillaVAE 2	0.0254	225.58	0.9943	0.9993	0.9691	0.9993
VanillaVAE 3	0.0351	308.63	0.9705	0.9967	0.9794	0.9833
VanillaVAE 4	0.0283	269.27	0.9695	0.9948	0.9381	0.9734

A.5 DCGAN IMPLEMENTATION

The DCGAN model is implemented in Pytorch according to Radford et al. (2016) with some deviations. To the best of our knowledge, deviations are found in the input size and first layer (we kept the image size of 32x32x3 and performed a 1x1 convolution) and the negative slope of the LeakyReLU function (from 0.2 to 0.1). A VAE was designed based on the DCGAN discriminator with the last convolutional layer as two layers mapping to 100-dimensional vectors with sigmoid and ReLU activation. As the complexity of the model with channel sizes up to 1024 lead to strong over-fitting, the VAE is implemented without the first 1x1 convolution, thus reaching only maximum channel sizes of 512. Training was performed for 500 epochs for each model on CIFAR10 full train set. Mini-batch sizes are 512 and 128 for train and test set, respectively. Random seed is again set to 0 and all weights are initialized with default settings. DCGAN was trained with the Adam optimizer with a learning rate of $2e-4$, weight decay $1e-5$ and beta 0.5. DGD and VAE were trained with the Adam optimizer as well with weight decay $1e-5$, but with default beta 0.9 and decreasing learning rates. Learning rates for the DGD are set to $\{1e-3, 1e-4, 1e-5\}$ at epochs $\{0, 50, 200\}$. The learning rate of the VAEs is set to $1e-4$ and decreased to $1e-5$ at epoch 200. Losses are computed as BCE loss for DCGAN, $MSE + MAP$ loss for DGD, and $MSE + \beta KLD$ for the VAEs. The value for β was set to 0.00001.

A.6 SUPPORTING EXPERIMENTS

GAUSSIAN SPACE COVERAGE VS. LATENT DIMENSION

We investigated the relationship between latent dimension and latent space standard deviation based on subsets of FashionMNIST and CIFAR10. The CIFAR10 models used here are based on the linear decoder introduced in A.3 with capacity 24. The architecture for the FashionMNIST models is similar, adapted to the different output image of size $(1, 28, 28)$. The linear layer output is of size $(capacity \times 4, 3, 3)$ and the second transposed convolutional layer has padding 0. The capacity is here set to 16. Latent dimensions $\{2, 5, 10, 20, 50, 100, 200\}$ are investigated for both data sets, and models are trained in triplets with random seeds $\{0, 25, 872\}$. All models are trained with learning rates $1e-4$ (decoder) and $1e-2$ (representation), the Adam optimizer with weight decay $1e-5$ for 50 epochs. Batch sizes are 64 and 16 for train and test sets, respectively. Reported are the latent dimensionality and the standard deviation of all latent values (of the test representation) at the end of training.

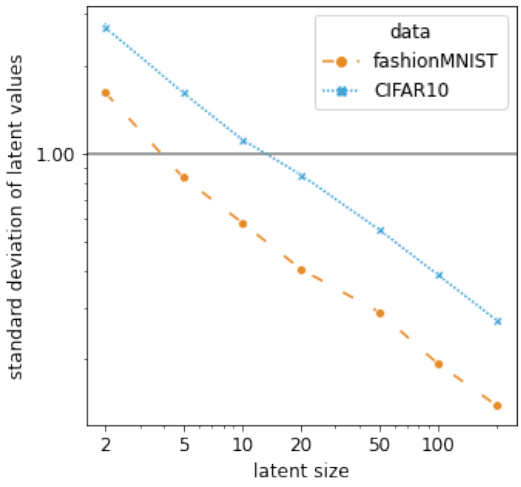

Figure 5: **Relationship between latent dimension and scale of latent space** Log-scaled standard deviations of triplet (different random seeds) models are plotted against the log-scaled dimensionality of the latent space.

CLASS-SPECIFIC CIFAR10 IMAGE GENERATION

We investigated whether recognizable, class-specific images could be generated from the DGD model presented in Figure 2. For this purpose, we calculated the mean representations per class along with their standard deviations, and sampled new data from resulting normal distributions. However, the generated images bear no significant systematic differences and are not recognizable as their assigned classes.

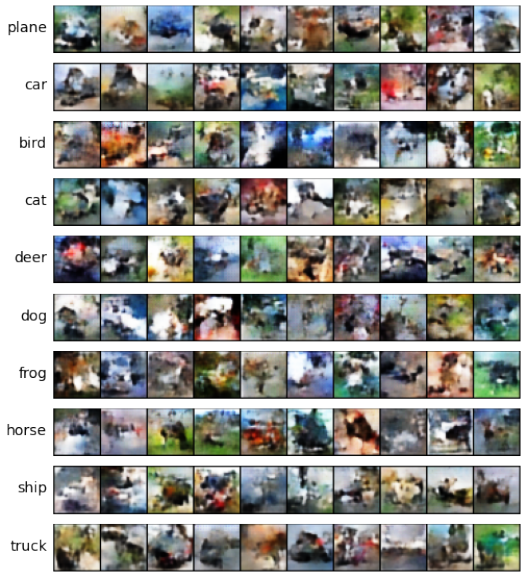

Figure 6: **Generated images from class-specific latent spaced of the DGD.**

A.7 MNIST EXPERIMENTS

The neural networks used for the MNIST experiments have a 2D representation space, one hidden layer with 30 units followed by two layers of 4x4 transposed convolutions with 64 and 32 channels, respectively. The output layer use a sigmoid and the other layers a ReLU activation. Images are thresholded and a cross entropy loss function is used. The training parameters are as follows:

Table 4: Training parameters for MNIST experiments

	Learning rate	**Momentum**	**Weight decay**	**Optimizer**
Decoder	0.001	NA	0.00001	Adam (Pytorch)
Representation	0.01	0.9	0	Stochastic gradient descent
GMM parameters	0.1	NA	0	Adam (Pytorch)

Priors on the GMM:
For the mixture weights we used Dirichlet prior with parameter 1 for all components.
For component means we used a soft ball prior with radius $b = 10.0$ and hardness of $a = 5.0$.
For the prior on the variances of the components, we used a Gaussian prior on $B = \log(1/\sigma^2)$. We tested combinations with mean 0, 1, and 2 and standard deviations 1, 2, and 3. From visual inspection of initial tests we selected mean 2 and standard deviation 2.

A random subset of 5000 images was used for training. Training was done for 100 epochs with a mini-batch size of 32.