Physics of Deep Learning Seminar

Generating Realistic Images using Deep Learning

Philip Dietrich,* Laurenz Hemmen,† Luca Tiede,‡ and Roland Zimmermann§
Georg-August-Universität Göttingen
(Dated: August 4, 2017)

Generative Models have shown huge improvements in recent years. Especially the field of Generative Adversarial Networks (GANs) have proven useful for many different problems. In this report we will compare two kinds of generative models, which are GANs and Variational Autoencoders (VAEs). We apply those methods to different data sets, to point out their differences and to see their capabilities and limits as well: We find that while VAEs are easier as well as faster to train, their results are in generell more blurry than the images generated by GANs. These on the other hand contain more details, which may realistic ones but often is just noise.

Keywords: generative adverserial network, variational autoencoder, image generation, generative models

I. INTRODUCTION

The research field of generative models is rapidly growing. New applications like creation of super-resolution images [2] or denoising of human speech [21] pop up constantly. But in the long term the goal is a much more abstract one: As these models learn to generate objects of a particular data set, they learn an efficient representation of a large data set with fewer parameters. Thus, some speculate, they could one day automatically recognize the essential features of a previously unknown data set [11]. In this paper however, we will focus on image data only.
In general, a generative model maps points from a usually Gaussian distribution to a distribution of generated images, which, in terms of the loss function, should be close to the true image distribution, from which the training data is sampled. The different models differ in exactly how the mapping from the Gaussian distribution to the image space happens.
Since generation problems have, compared to normal supervised learning, no concrete target vector, one had to find new methods for these tasks. In this paper, we want to compare two of those architectures, Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), on different data sets. Chapter II and III introduce the theory and architecture of the models to compare both on different data sets in chapter V.

II. VARIATIONAL AUTOENCODER

A. Introduction to Autoencoder

Autoencoders are certain neural network architectures, characterized by a hourglass shaped structure (see [22]). They are trained in an semi-supervised fashion.
One propagates images X through the network, and tries to reconstruct them at the other end as closely as possible.
Doing this, the network learns a compressed representation of the images in its middle, since most of the information of how the image should look like is encoded in just a few variables, e.g. the activations of the middle neurons. For obvious reasons, the left part of the network is called the encoder, and the right part is called the decoder (see fig. 1).

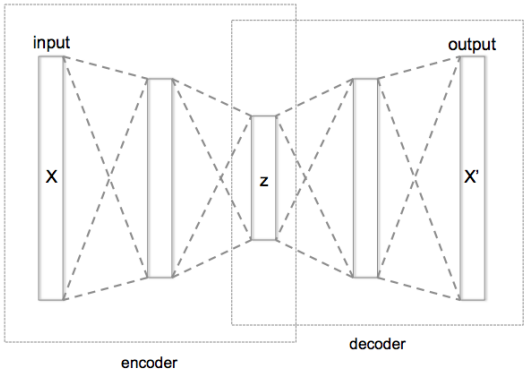


FIG. 1: Schematic structure of an *autoencoder*, modified according to [3].

Those variables in the middle are called *latent variables* z. They span the *latent space*.
Mathematically one can express the encoder as a probability distribution $P(z|X)$, and the decoder as $Q(X|z)$. It is reasonable to assume, that after training, there is a manifold in the latent variable space, that represents realistic looking images. For now, one *accesses* this manifold by encoding real images.

B. Sampling from latent variable space

If the task is to generate new images, one just has to sample points from this manifold, and decode them.

* philip.dietrich@stud.uni-goettingen.de
† laurenz.hemmen@stud.uni-goettingen.de
‡ lucaanthony.thiede@stud.uni-goettingen.de
§ roland.zimmermann@stud.uni-goettingen.de

Unfortunately, the shape of the manifold is unknown, and points laying not on the manifold, will just produce noise when propagated through the decoder.
Thus, the goal is to obtain the probability distribution $P(z)$.
One does this (see [7, 13, 14, 18]), by deriving a certain new loss function, which has a descriptive interpretation. Note that the form of the probability manifold depends on the encoder and decoder part. Therefore, given the right encoder and decoder, it can have arbitrary shapes. Starting with the *Kullback-Leibler (KL)* divergence between the real latent Variable space given some data $P(z|X)$ and the distribution the encoder produces given some data $Q(z|X)$

$$D_{KL}(Q(z|X)\,||\,P(z|X)) = \sum_z Q(z|X)\log\left(\frac{Q(z|X)}{P(z|X)}\right). \tag{1}$$

After some calculations one arrives at the VAE loss function

$$\begin{aligned}\mathcal{L}(X,z) &= -\log(P(X)) + D_{KL}(Q(z|X)\,||\,P(z|X)) \\ &= -E[\log(P(X|z))] + D_{KL}(Q(z|X)\,||\,P(z)),\end{aligned} \tag{2}$$

where $E(\cdot)$ is the expected value.

The loss function can be interpreted as follows:
$P(X)$ is the probability distribution we want to model (which gives a high probability for data that represent an actual image, and low probability for noise) and is therefore fixed.
$D_{KL}(Q(z|X)\,||\,P(z|X))$ is the error arising when modeling $P(X)$. Thus, by minimizing the loss $\mathcal{L}(X,z)$ in (2), one obtains a lower bound for the real probability distribution.
The term $E[\log P(X|z)]$ represents the maximum likelihood estimation. Maximizing just this term would be a normal autoencoder.
The second term is the key for using an VAE to generate images: The KL divergence describes how different two probability distributions are (a value of 0 means that they are identical). By maximizing the right side of the equation, one can impose a form on the probability distribution of the latent variables, that is easy to handle and one can sample from.

This approach of imposing the form of a distribution by design and treating it as an optimization problem, in order to find a lower bound of the probability distribution is called *Variational Inference*, hence the name *Variational Autoencoder*. Naturally, the choice is the Normal distribution $\mathcal{N}(0,1)$, because it is easy to sample from, and the KL-divergence can be calculated analytically. Thus, after some calculations and substituting $\log\Sigma(X)$ with $\Sigma(X)$ (since it is numerically easier to calculate), one obtains

$$\begin{aligned}&D_{KL}(\mathcal{N}(\nu(X),\Sigma(X))\,||\,\mathcal{N}(0,1)) = \\ &\frac{1}{2}\sum_k \exp(\Sigma(X)) + \mu^2(X) - 1 - \Sigma(X).\end{aligned} \tag{3}$$

Concluding, the VAE loss function is

$$\begin{aligned}\mathcal{L} = -E[\log(P(X|z))] + \frac{1}{2}\sum_k \exp(\Sigma(X)) \\ + \mu^2(X) - 1 - \Sigma(X).\end{aligned} \tag{4}$$

One wants to learn the model with standard backpropagation. Sampling from a probability distribution however, is not differentiable. To solve this problem, the *reparametrization trick* can be used [13].

It uses, that even if the sampling itself is not differentiable, the parameters of the probability distribution (here the mean and variance) are [13].
To achieve this, one splits the network in the middle into two nodes which each have their own dense layer before it, one of which represents the mean, and the other the variance of the sampling layer.

C. Conditioning the model

The theory of conditional VAE are very similar to the plain VAE, just that each of the probability distributions is conditioned under an extra parameter c.
Practically one can do that for example, by concatenating one hot encoded class labels to the latent variable space.

D. Architecture

The basic architecture is from [5]. After various tries with different modifications, the results were achieved with the following one. The left side (the encoding part) is built by four convolutional layers with a ReLU activation function. The decoder uses three convolutional transposed layers and one convolutional layer with the same activation function. For the MNIST data set only two latent variables were used, while for the rest a number of 300 yielded acceptable results.

III. GENERATIVE ADVERSARIAL NETWORKS

The basic principle of GANs is rooted in game theory: In addition to the generator network G, we train a discriminator network D, which acts as a simple classifier, distinguishing between real images X and generated images $G(z)$, where z is a latent vector sampled from a gaussian.

The training process consists of two alternate steps: First, the discriminator is trained on a batch of real and generated images to distinguish between the two. Secondly, the generator is trained on a new batch of generated images (twice as much, since the batch contains no fake images) by backpropagating through the combined network of discriminator and generator, using a target vector that labels all generated images as real. In other words, the discriminator is trained to maximize the log-likelihood that it categorizes an object correctly, whereas the generator wants to minimize the same quantity [20]:

$$\begin{aligned}\mathcal{L} &= E[\log P(S = real|X)] + E[\log P(S = fake|G(z)] \\ &= E[\log D(X)] + E[\log 1 - D(G(z))] \end{aligned} \tag{5}$$

One can say that both networks compete against each other: The discriminator learns to separate real from fake images, whereas the generator tries to "fool" the discriminator until both, generated and real, look the same for it.

A. Architecture

For the generator architecture two types of layers where used to transform the latent vector into the shape of the image space: First, simple upsampling layers were used, which copy the value of one pixel into multiple pixels in the next layer, thereby multiplying the size. Secondly, we used, convolutional transpose layers, which details are explained in [10].

We found that after some critical point, more layers destabilize the training process or lead to instant collapse of the GAN. A good rule of thumb, found in [11] states, that the number of model parameters should be of the same or lower order of magnitude than the number of pixel in the training data. This ensures that the information from the training data cannot be copied, but a generalization of it has to be learned.
For all types of GANs there were some general rules for the architecture, already presented in other work (e. g. [23]), which we used and partially confirmed (only partially, because some changes had no measurable effects for some datasets): Batch normalization in both the generator and the discriminator, improves training success. The best activation function for the generator is ReLU (except for the output, where tanh is used), whereas the discriminator works best with LeakyReLU.
Additionally, in [23] it is recommended to replace all pooling and upsampling with convolutions and transposed convolutions, respectively. We found that especially for the simple datasets like MNIST the upsampling led to good results with less training time. For the complex datasets on the other hand, the transposed convolutions proved to be faster converging and generated better images.
For all networks we chose ADAM [12] as our optimizer with the parameters found in [23].

B. Wasserstein GAN

Even with those tricks, training a GAN is a very unstable process: Whenever one of the competing networks becomes to strong the other one suffers from high gradients, which lead to an unstable learning process. Another problem of GANs is that it is impossible to quantify the realness of generates images using only the loss function. Wasserstein GANs (WGANs) overcome those two problems [1]. The main difference to usual GANs is the changed loss function: Instead of predicting probabilities, determining how likely one sample is real, in a WGAN the activation function is omitted in the last layer of the discriminator network (which therefore is called **critic** here). The loss-function is therefore changed to

$$\mathcal{L} = E[P(S = real|X) - P(S = fake|X)] \tag{6}$$

This function originates from the mathematical background of WGANs which is omitted here, but can be found in [1]. With this loss function gradients will neither vanish nor explode, which leads to a more stable training process. It also has the advantage, that it is no longer a problem for the critic network to become to strong. Thus, the critic will typically be trained more often, which directly leads to the second advantage of the WGAN. This originates from its mathematical background, which is trying to minimize the so called Wasserstein distance, which measures the similarity between the real data distribution and the generated one. This distance is (up to a constant factor) approximated by the loss function (for mathematical details see [1]). So one can easily determine convergence behaviour of the model.

C. Auxilary GAN

One way to improve stability of GANs and avoid mode collapse is to use class labels. There are several different ways to include class labels in the network architecture, as depicted by [20]. The one used in this project is called AC-GAN and can be derived from a usual GAN as follows:
First, the generator model will get a second input, which is the class label. This label will be turned into a vector of dimensions of the number of latent variables, so for every class there is another vector.
Furthermore, the discriminator gets the additional task of predicting the class label. It has been shown in multiple cases that giving the network a secondary task, the performance of the primary task improves, too (eg. [23], [25], [24]). The loss function (5) gets an additional component:

$$\mathcal{L}_{\text{class}} = E[\log P(C = c|X)] + E[\log P(C = c|G(z))] \tag{7}$$

The prediction of the class should be correct for both, the generator and the discriminator. Therefore, the

total loss to minimize becomes $\mathcal{L}_c + \mathcal{L}$ for the generator, and $\mathcal{L}_c - \mathcal{L}$ for the discriminator [20].

Together these two changes give more stability, with the welcome side effect, that, similar to the conditional GAN [19], one can predetermine the class one want to generate. The difference here is that the conditional GAN gives both the discriminator and the generator the class information as input, whereas the auxiliary GAN lets the discriminator predict the class label.

IV. EXAMINING THE IMAGE GENERATION

To get a better understanding of the learned relations between the input and the generated output of our models, we developed certain live applications to illustrate those.
For examining the behaviour of the GANs an application has been created, in which one can modify the randomly chosen input one by one and observe, how the generated image changes. Doing this, one can identify specific roles of certain pixels, which can control visual aspects of the image, e.g. the rotation, the color and the sharpness.
As the VAEs can generate images using a low dimensional latent space (for the MNIST data set), another application has been developed in which one can see the latent variables of certain test images. By manually choosing latent variables which lay between points of different classes, one is able to observe a transformation from one class to another. This encourages the understanding of why images, generated by the unconditioned VAEs have a greater probability of being blurry compared to the results of the GANs.

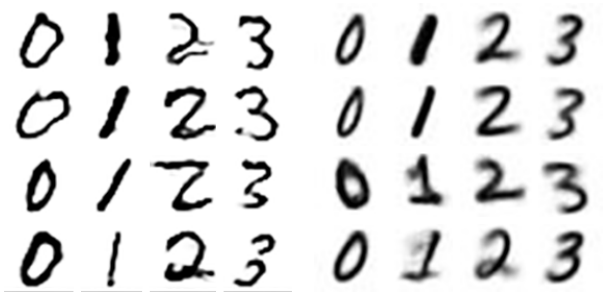
V. RESULTS

The described methods where applied to four different data sets, namely MNIST [16], CHAR74K [6], CIFAR10 [15] and CELEBA [17]. To reduce the required computational effort we resized to images to have a maximum size of 72×72 pixels. Implementations were done with Python and KERAS [4]. The code can be found at github.com/yypdyy95/GANs-vs.-VAEs.

A. Generated Samples

1. *MNIST*

As the MNIST data set has the least variance of the examined data sets, one expects the two models to generate realistic images. Therefore, this data set is the first one to be examined. Some exemplifying images, generated by a conditioned VAE and an auxiliary GAN, can be seen in figure 2. Both models generate images which

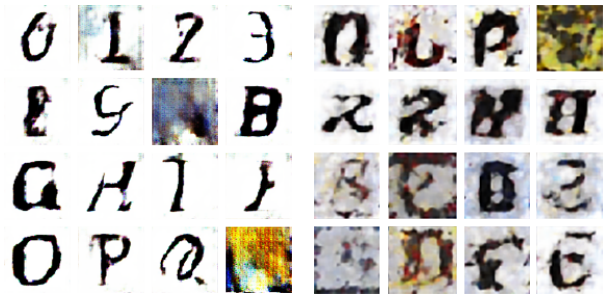


(a) Auxiliary GAN (b) VAE, conditioned

FIG. 2: Comparison of sampled images of the two models based on the MNIST data set.

can easily be recognized as digits. While the GAN generates sharper images, the VAE tends to smooth the edges of the digits.

2. *char74k*



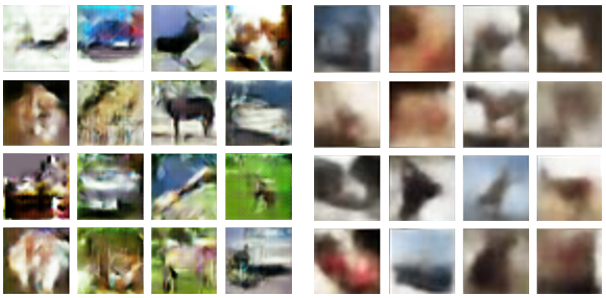
(a) Auxillary WGAN (b) VAE, not conditioned

FIG. 3: Comparison of sampled images of the two models based on the CHAR74K data set.

Following the MNIST data set, images from the CHAR74K data set have also been generated using an Auxillary WGAN and an unconditioned VAE. Examples are presented in figure 3. While the VAE offers images with more variation of style, the images of the GAN are clearer. All in all, they can be recognized as characters easier as the pictures of the VAE.

3. *CIFAR10*

The CIFAR10 data set has a great variance of motives and camera angles. Therefore, one expects this images to be harder to generate than the previous examples. This can be confirmed by looking at the resulting images in figure 4. The images of the VAE are once again blurry and no realistic objects can be recognized. The GAN generates images with sharper edges; nevertheless, most of the generated objects can not be uniquely identified.



(a) GAN, conditioned to different classes (b) VAE, not conditioned

FIG. 4: Comparison of sampled images of the two models based on the CIFAR10 data set.

4. *CelebA*

Lastly, the generative models have been used to generate portrait images of humans using the CelebA data set. Figure 5 shows exemplary portraits of the data set and generated images. Herex, an unconditioned GAN has been compared to the results of a conditioned VAE. The GAN produces again much sharper images than the VAE. Nevertheless, the faces produced by the VAE own a more natural appearance. Apart from the blurry earth-colored background, some VAE-generated images resemble realistic faces. In figure 5c and 5d the condition between male and female persons is demonstrated.

B. Observations

During the training processes of the GANs and the VAEs one can see that the models at first learn to produce the right shapes. The usage of the correct colors is learned later.

A similar phenomenon could be observed using the conditional models: First, all images produced simple, similar looking shapes, that over time developed into the different classes: When generating numbers with the AC-GAN, all numbers resembled ones first, that then slowly warped in the different numbers over the epochs.

VI. CONCLUSION

The main difference between the methods examined here, is their learning process. VAEs are minimizing a loss reproducing a certain image, and can therefore be considered as semi-supervised learning. GANs on the other hand are considered unsupervised learning, because they do not use labeled pixels. The most important difference, found in this work was the training time for the two methods. Mostly GANs took a lot longer to train (in terms of number of epochs, as well as in terms of run time). Because experiments were run on different hardware a quantitative comparison is not done here. Another advantage of VAEs is their stability. For GANs highly oscillating image quality could be observed in the course of training. Clear pictures turned into purely grey images within only a few epochs of training. Therefore the use of WGANs was considered and proved a lot more stable. The problem with VAEs is, that with increasing diversity of the data set generated images become more and more blurry and a lot of details get lost (see CIFAR10 results). With GANs this does not necessarily occur. Eventually we conclude, that for low-diversity data sets like MNIST or char74k, both methods give sufficiently realistic images. For more complex data sets this was not the case in this work, but prior work like [8, 9] shows, that it is possible to generate realistic images with both the techniques used here.
All in all, using VAEs one can achieve results in less time, but with decreased image quality compared to results of GANs.

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[2] Zhimin Chen and Yuguang Tong. Face super-resolution through wasserstein gans. *CoRR*, abs/1705.02438, 2017.

[3] Chervinskii. Schematic picture of an autoencoder architecture. `https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png`, 2015.

[4] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[5] Franois Chollet. Convolutional variational autoencoeder example. `https://github.com/fchollet/keras/blob/master/examples/variational_autoencoder_deconv.py`, 2016.

[6] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009.

[7] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

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

[9] Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vázquez, and Aaron C. Courville. Pixelvae: A latent variable model for natural images. *CoRR*, abs/1611.05013, 2016.

[10] Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generating images with recurrent adversarial networks. *CoRR*, abs/1602.05110, 2016.

(a) Ground truth (b) GAN, not conditioned (c) VAE, conditioned to generate males (d) VAE, conditioned to generate females

FIG. 5: Comparison of sampled images of the two models based on the CelebA data set.

[11] A. Karpathy, P. Abbeel, G. Brockman, V. Cheung P. Chen, R. Duan, I. Goodfellow, D. Kingma, J. Ho, R. Houthooft, T. Salimans, J. Schulman, I. Sutskerver, and W. Zaremba. Generative Models, June 2016.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[14] Augustus Kristiadi. Variational autoencoder. `http://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/`, 2016.

[15] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[16] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[17] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[18] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015.

[19] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[20] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*, 2016.

[21] Santiago Pascual, Antonio Bonafonte, and Joan Serrà. SEGAN: speech enhancement generative adversarial network. *CoRR*, abs/1703.09452, 2017.

[22] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2007.

[23] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[24] B. Ramsundar, S. Kearnes, P. Riley, D. Webster, D. Konerding, and V. Pande. Massively Multitask Networks for Drug Discovery. *ArXiv e-prints*, February 2015.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.