engineering proceedings

Proceeding Paper

Data Generation with Variational Autoencoders and Generative Adversarial Networks [†]

Daniil Devyatkin * and Ivan Trenev

V.A. Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, 65 Profsoyuznaya Street, 117997 Moscow, Russia
* Correspondence: devyatkin.for.other@bk.ru; Tel.: +7-495-334-8910; Fax: +7-499-234-64-26

† Presented at the 15th International Conference "Intelligent Systems" (INTELS'22), Moscow, Russia, 14–16 December 2022.

Abstract: The paper considers the problem of modelling the distribution of data with noise in the input data. In this paper, we consider encoders and decoders, which solve the problem of modelling data distribution. The improvement of variational autoencoders (VAEs) is discussed. Practical implementation is performed using the Python programming language and the Keras framework. Generative adversarial networks (GANs) and VAEs with noisy data are demonstrated.

Keywords: machine learning; deep learning; autoencoders; generative adversarial network; MNIST

Citation: Devyatkin, D.; Trenev, I. Data Generation with Variational Autoencoders and Generative Adversarial Networks. *Eng. Proc.* **2023**, *33*, 37. https://doi.org/10.3390/engproc2023033037

Academic Editors: Askhat Diveev, Ivan Zelinka, Arutun Avetisyan and Alexander Ilin

Published: 20 June 2023

1. Introduction to Variational Autoencoders

An autoencoder is a special architecture of a neural network, applying unsupervised learning using the backpropagation method. In other words, it is a neural network that has been trained to copy its input to output. The network consists of two parts: encoding functions $z = E(x, \theta_E)$ and decoding function $\hat{x} = D(z, \theta_D)$ [1]. Figure 1 demonstrates an example of an autoencoder with three hidden states. In the learning process, the autoencoder tries to learn the identical function $\hat{x} = D(E(x))$ by minimizing some loss function $L(\hat{x}, x)$. The solution to minimizing the factor can be consider as

$$[\theta_E, \theta_D] = arg\,min\, L(\hat{x}, x), \tag{1}$$

where θ_E and θ_D are the weights of the encoder and decoder, respectively.

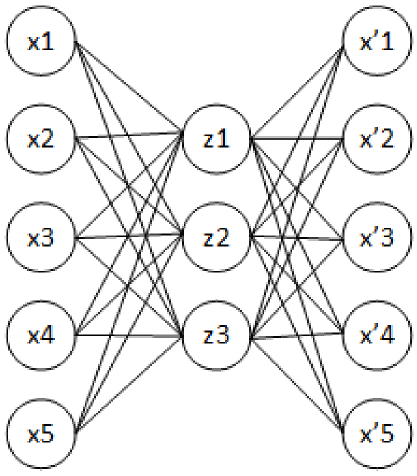

Figure 1. Autoencoder with three latent states.

There are many types of autoencoders: downscaling, where the dimension of z is less than of the input one; sparse, where a penalty is added to the loss function for large values of the latent representation z, etc. The main idea of autoencoder theory is the conjecture on the concentration of data in the neighbourhood of some low-dimensional manifold. Any autoencoder training procedure involves a compromise between two goals:

1. Training the latent representation z of the training sample x such it can be accurately reconstructed from z using the decoder. However, there is one critical factor: x must be chosen from the training dataset. This means that the autoencoders must not reconstruct $x$which are not possible with respect to the probability function.
2. Satisfaction of the constraints or regularizing penalty.

These two requirements force the latent representation to capture information concerning the structure of the distribution which generates the data. If we consider an image as an input, then it is obvious that not all pixels carry useful information, most of them are noise. For example, if you search for some object in the image, then the informative pixels are only those that form the desired object. Furthermore, if we consider the image of an object in a certain neighbourhood, then it can also be considered an object.

The main problem with these models is predicting a sample from the original sample based on its hidden representation; however, this is inconvenient since there is no knowledge of how much confidence was predicted. In other words, each sample has a hidden representation that can be depicted in some space, but we do not know if neighbourhood of this point is still an object similar to the predicted one. The solution to this problem is not to predict a sample, but some distribution of the latent variables.

As noted above, z is a vector of hidden variables that defines an object x from the sample [2]. Let $P(x)$ be a distribution function of the initial data, $P(z)$ denotes the latent factor distribution density, and $P(x \mid z)$ is the image probability distribution for given latent factors. Then the data generation process can be expressed as [3]:

$$P(x) = \int\limits_z P(x \mid z) P(z)\, dz.$$

Let $P(x \mid z)$ be the sum of some generating function $f(z)$ and noise ϵ. We parametrize the generating function $f(z, \theta)$ by the vector θ in some space Θ, where $f : Z \times \Theta \to X$. Then the generation process takes the following structure:

$$P(x, \theta) = \int\limits_z [f(z, \theta) + \epsilon] P(z)\, dz. \tag{2}$$

If we assume that the optimization is being carried out according to the L_2 metric, then the noise is normally distributed $\epsilon \sim \mathcal{N}(0, \sigma^2 E)$, and we obtain

$$P(x \mid z, \theta) = \mathcal{N}(x \mid f(z, \theta),\, \sigma^2 E),$$

where $f(z, \theta)$ is modelled by a neural network, E is the identity matrix, and σ^2 is a positive scalar.

Next, the parameters θ must be found to guarantee the maximum likelihood estimation in order to maximize the probability of occurrence for objects of the sample $P(x)$. In practice, for most z, the probability $P(x \mid z)$ tends to zero, and therefore, it contributes almost nothing to the estimate of $P(x)$. The key idea of the variational autoencoders (VAEs) is to try and find values of z which lead to x, enabling the calculation of the probability estimate $P(x)$ on x. To this, introduce a new function, $Q(z \mid x)$, must be introduced which can take values of x and construct a distribution for z that leads to x. The main hypothesis is the cardinality of a set with "good" z is much smaller than the cardinality of the set for all z. This distribution Q can be trained to assign high-probability values to those z that are highly likely to generate x. However, instead of maximizing (2), we need to maximize $\mathbb{E}_{z \sim Q}[\, P(x \mid z)\,]$.

We then write the Kullback–Leibler divergence between the $Q(z \mid x)$ and $P(x \mid z)$ distributions as follows

$$D_{\mathrm{KL}}[\, Q(z \mid x) \parallel P(z \mid x)\,] = \mathbb{E}_{z \sim Q}[\log Q(z \mid x) - \log P(z \mid x)],$$

where $P(z \mid x)$ is the real probability distribution of hidden factors for a given x. Next, by applying the Bayes formula to $P(z \mid x)$, we obtain

$$D_{\mathrm{KL}}[\, Q(z \mid x) \parallel P(z \mid x)\,] = \mathbb{E}_{z \sim Q}[\log Q(z \mid x) - \log P(x \mid z) - \log P(z)] + \log P(x). \quad (3)$$

In the expression in (3), $\log P(x)$ does not depend on z, so it leaves the mathematical expectation. Next, let us obtain one more Kullback–Leibler divergence

$$D_{\mathrm{KL}}[\, Q(z \mid x) \parallel P(z \mid x)\,] = D_{\mathrm{KL}}[\, Q(z \mid x) \parallel \log P(z)\,] - \mathbb{E}_{z \sim Q}[\log P(x \mid z)] + \log P(x). \quad (4)$$

The expression in (4) holds true for any $Q(z \mid x)$ and $P(z \mid x)$. By applying permutations to the terms of the equation, one can obtain the following expression

$$\log P(x) - D_{\mathrm{KL}}[\, Q(z \mid x) \parallel P(z \mid x)\,] = \mathbb{E}_{z \sim Q}[\log P(x \mid z)] - D_{\mathrm{KL}}[\, Q(z \mid x) \parallel \log P(z)\,]. \quad (5)$$

In Formula (5), the $Q(z \mid x)$ is the encoder and $P(z \mid x)$ is the decoder, and these distributions are modelled by a neural network; therefore,

$$Q(z \mid x) = Q(z \mid x, \theta_E), \;\; P(z \mid x) = P(z \mid x, \theta_D),$$

where θ_E and θ_D are the weights from Formula (1). The goal of training a VAE is to maximize $P(x)$. The right-hand side of (5) can be optimized by gradient methods. On the right, the first term denotes the quality of prediction x by the decoder from the values of the hidden variables z, and the second term is the Kullback–Leibner divergence between $P(z)$ and $Q(z \mid x)$. Let us predict Q as a normal distribution with the following parameters,

$$Q(z \mid x, \theta_E) = \mathcal{N}(\, \mu(x, \theta_E), \, \Sigma(x, \theta_E)),$$

where $\mu(x, \theta_E)$ is the mathematical expectation, and $\Sigma(x, \theta_E)$ is the covariance matrix. That is, the encoder for each x predicts two values: the mathematical expectation μ and the covariance matrix Σ. In other words, the encoder predicts some normal distribution. It is necessary for the distribution of $Q(z \mid x)$ to be similar to a normal distribution, that is, the Kullback–Leibner divergence tends to 0, and the quality of the data generated by the decoder is maximized. This means that two loss functions are used in the implementation of the model

$$D_{\mathrm{KL}}[\, Q(z \mid x, \theta_E) \parallel \mathcal{N}(0, \, E)\,],$$
$$||x - f(z)||,$$

where x is the true data sample, and $f(z) = D(E(x))$ is the predicted data by the VAE.

Here, random values $z \sim Q(z \,|x\,, \theta_E)$ are taken and passed to the decoder. It is impossible to propagate errors through random values directly, so the reparametrization trick is used. This method is based on the following formula:

$$\mathcal{N}(\mu(x\,, \theta_E), \, \Sigma(x\,, \theta_E)) = \mu(x\,, \theta_E) + \Sigma(x\,, \theta_E) \cdot \mathcal{N}(0, \, E),$$

where $\mu(x\,, \theta_E)$ is the expected value, $\Sigma(x\,, \theta_E)$ is the covariance matrix, and $\mathcal{N}(0, \, E)$ is a standard normal multivariate distribution (visualization in the Figure 2).

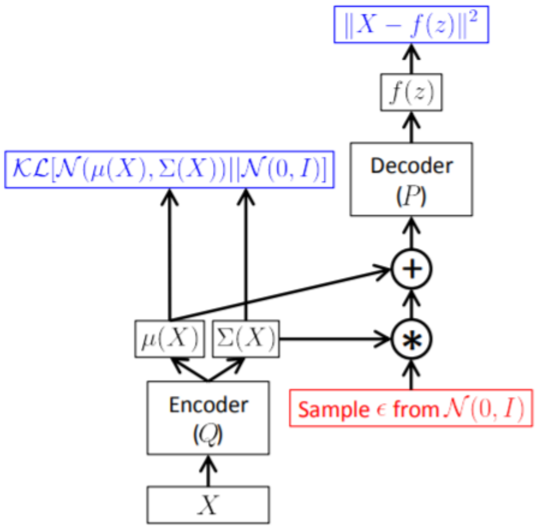
$\|X - f(z)\|^2$
$f(z)$
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$
Decoder (P)
$+$
$\mu(X)$
$\Sigma(X)$
$*$
Encoder (Q)
Sample ϵ from $\mathcal{N}(0, I)$
X

Figure 2. Block diagram of a variational encoder (VAE) architecture.

2. Generative Adversarial Networks

VAEs compare the original and generated objects based on the mean squares error and binary cross entropy if the labels are given as the input: this is a poor comparison. This disadvantage manifests to a lesser extent in another approach, generative adversarial networks (GANs). The overall goal of a GAN is to synthesize new data that has the same distribution as the training set.

There are two neural networks in the GAN model—generator and discriminator [4]. These models are trained in turn. After the model weights are initialized, the generator generates images, initially this is noise. After several iterations of training the generator, the discriminator starts working and the generator stops training. This network distinguishes differences between real images and images synthesized by the generator, and predicts whether the image is original or generated (0 if false, 1 if true). Networks have been trained to learn to solve their problems. Two networks play an adversarial game, where the generator learns to obtain its output and fool the discriminator. Discriminator detection becomes better at detecting synthesized images. The generator creates random numbers from a given distribution $P(z)$ and generates objects $X_p = G(z, \theta_g)$ from them, used as the input of the second network. The discriminator receives the objects from the training sample X_s and objects created by the generator X_p as the input; subsequently, it learns to predict the probability whether that particular object is real, giving the scalar $D(z, \theta_d)$. Let the generator be represented as a mapping $G(z, \theta_g)$, where G is a differentiable function, and θ_g are generator parameters. The discriminative model $D(z, \theta_d)$ is presented, the model predicts if the input is real from the training set or synthesized by the generator, where θ_d are the discriminator parameters [5].

Figure 3 shows the following: (Left) the training phase of the discriminator is shown: the gradient (red arrows) only flows from the loss function to the discriminator, where θ_d (green) is updated to reduce the loss function. The gradient from the right side of the loss function (object identification error) flows to the generator, only updating the θ_g generator weights (green) towards increasing the probability of the discriminator to make an error. During the training of the two models, it is necessary for the discriminator D to maximize the probability of correctly identifying objects using the training and generated samples, enabling the generator G to minimize $\log(1 - D(G(z)))$. In other words, it is necessary to reach the next criterion (see Figure 3)

$$\min_G \max_D V(D, G) = \underset{x \sim p_{data}}{E}[\log D(x)] + \underset{z \sim p_z}{E}[\log(1 - D(G(z)))].$$

Generally, the task of training the discriminator and generator is not to find the local or global minimum of a function, but to find an equilibrium point. In game theory, this point is called the Nash equilibrium point, where both players no longer benefit, although they follow the optimal strategy [6].

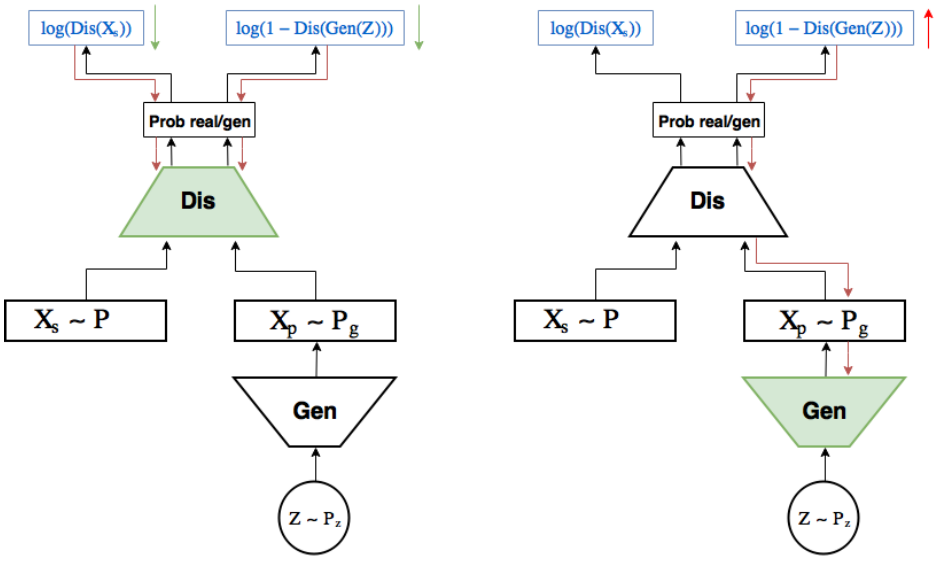

Figure 3. GAN training scheme.

3. Practical Implementation

The implementation of the described generative models is demonstrated using Python and the Keras framework [7–9]. In the first stage, data generation based on the VAE and GANs on the MNIST (Modified National Institute of Standards and Technology database) dataset is demonstrated [10]. MNIST is a well-known dataset for handwritten numbers.

The encoder architecture used two convolution layers and three fully connected layers. For the encoder, except the output, the activation function was ReLU and the output is sigmoid. The size of the hidden variables was 2. The decoder used a fully connected layer and three layers of transposed convolutions. Furthermore, the decoder output is sigmoid. Let us move on to the GAN architecture. The generator used a fully connected layer, a 2D convolution layer, and upsampling. The discriminator used a fully connected layer, 2D convolution, max pooling, and flattening [11].

Figure 4 shows an example of data generation. The image shows that the autoencoder performing as expected; however, the contours of the images are very blurry and some numbers are very similar (this is due to the similar hidden representation of these objects). Let us add noise to the data resulting in the following: neither the VAE nor the GAN are robust models. It is worth noting that the noise has a normal distribution with a mathematical expectation of 0 and a variance of 0.1. Figure 5 shows that in the presence of noise, data generation does not give the necessary results [12].

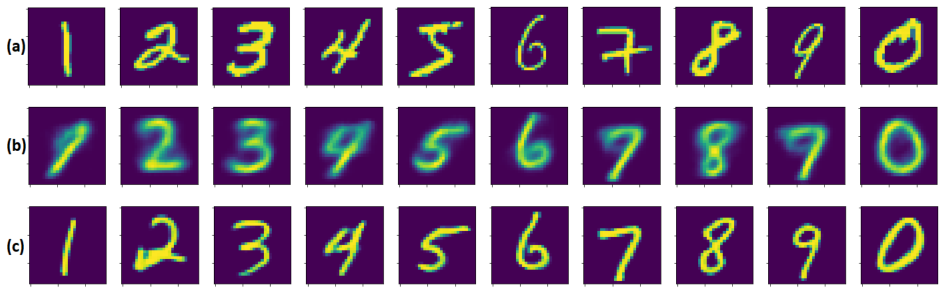

Figure 4. (**a**) Original MNIST dataset. (**b**) Synthesized images after autoencoder training. (**c**) Synthesized images by the GAN generator.

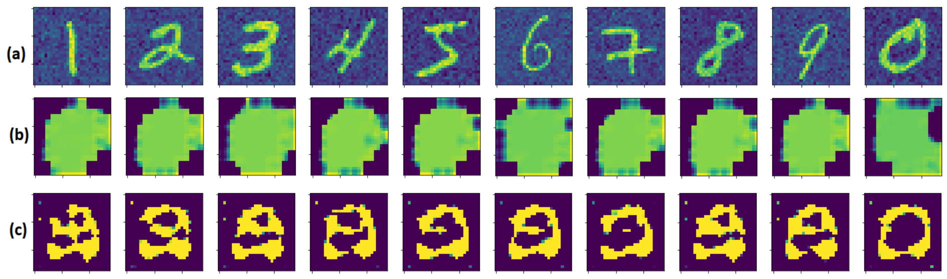

Figure 5. (**a**) Original MNIST dataset with noise. (**b**) Synthesized images after autoencoder training. (**c**) Synthesized images by the GAN generator.

4. Conclusions

This experiment shows that generative models demonstrate good results with pure data; however, if noise is added to the original sample, the results become unpredictable. To solve this problem, conditional generative models can be considered. In the case of GAN, one can consider their improvements: DCGAN and WGAN. DCGAN is a modification of the GAN algorithm based on convolutional neural networks (CNN). The task of finding a convenient representation of features on large volumes of unlabelled data is one of the most active areas of research, in particular, the representation of images and videos. One convenient way to find views can be this network. WGAN uses the Wasserstein loss metric inside the error function, allowing the discriminator to learn to identify repetitive outputs faster on which the generator stabilizes.

Author Contributions: Conceptualization, I.T.; methodology, I.T.; software, D.D.; validation, I.T.; visualization, D.D.; writing—original draft preparation, D.D.; writing—review and editing, D.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; The MIT Press: Cambridge, MA, USA, 2018; pp. 422–441.
2. Doersch, C. Tutorial on Variational Autoencoders. *arXiv* **2016**, arXiv:1606.05908
3. Ivchenko, G.I.; Medvedev, Y.I. *Introduction to Mathematical Statistics*; Publishing House LCI: Moscow, Russia, 2010; pp. 457–542.
4. Raschka, S. *Python Machine Learning*; Packt Publishing Ltd.: Birmingham, UK, 2020; pp. 513–551.
5. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; WardeFarley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. *arXiv* **2014**, arXiv:1406.2661.

6. ITMO University. Generative Adversarial Nets (GAN). Available online: https://neerc.ifmo.ru/wiki/index.php?title=Generative_Adversarial_Nets_(GAN) (accessed on 12 July 2022).
7. Python Software Foundation. The Python Standard Library. 2020. Available online: https://docs.python.org/3/library/ (accessed on 12 July 2022).
8. Keras. Simple. Flexible. Powerful. Keras: The Python Deep Learning API. Available online: https://keras.io/ (accessed on 12 July 2022).
9. Chollet, F. *Deep Learning with Python*; Simon & Schuster: New York, NY, USA, 2018; pp. 269–314.
10. MNIST Handwritten Digit Database. The Mnist Database of Handwritten Digit. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 14 July 2022).
11. SImonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2014. Available online: https://arxiv.org/abs/1409.1556 (accessed on 13 July 2022).
12. Mueller, A.; Guido, S. *Introduction to Machine Learning with Python*; O'Reilly Media Inc.: Sebastopol, CA, USA, 2016; pp. 180–221.