Industrial revolution 2.0: using variational autoencoders to produce images of clothing

Adriaan Hilbers

December 2018

Abstract

This document describes the use of variational autoencoders as generative models for images of clothing, using the *Fashion MNIST* dataset. This work was done as part of the *Deep Learning* course at Imperial College London in Autumn 2018.

1 Introduction

In the last few decades, machine learning techniques have achieved considerable success in many tasks previously considered difficult for a computer. *Generative models* seek to produce new (digital) objects, such as images, without human assistance. They are typically *trained* on "real" data such as existing images or audio fragments to learn their structures, which are imitated to form new examples. Various generative frameworks exist, including generative adversarial neural networks, variational autoencoders and normalising flows [1].

In this report, we employ a variational autoencoder to produce new images of clothing. The dataset and model architecture are introduced in sections 2 and 3 respectively. Section 4 describes its training, while Section 5 describes its use to generate new images of clothing. Section 6 discusses the results and recommends possible extensions.

Code for this assignment, using Python and Tensorflow, is publicly available at `github.com/ahilbers/deeplearning/assignments/week_6`.

2 Dataset: Fashion MNIST

We use the *Fashion MNIST* dataset[1] to train our model. It contains 70,000 images of articles of clothing, classified into one of 10 categories. The data is split into 60,000 *training* images and 10,000 *testing* images. An overview of the images, by category, is shown in Figure 1.

The *Fashion MNIST* is a drop-in replacement for the traditional MNIST database of handwritten digits. However, it is more challenging for machine learning algorithms since the inter-category variations are smaller and the intra-category variations larger.

[1] https://github.com/zalandoresearch/fashion-mnist

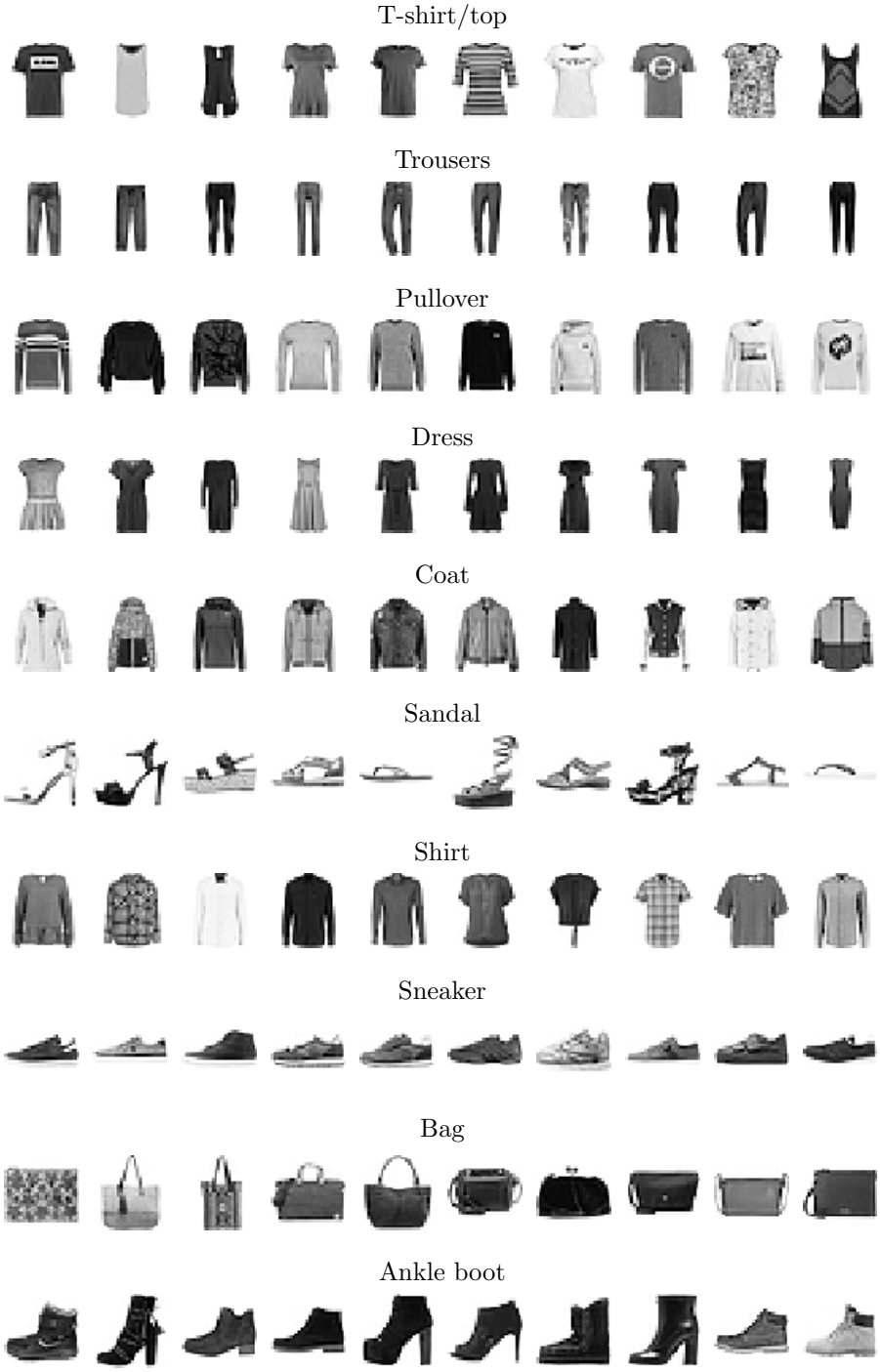

Figure 1: Samples images from the *Fashion MNIST* dataset.

3 Variational autoencoder: setup

In this section we introduce the variational autoencoder's architecture and how it is trained. It is inspired by [1, 4, 5, 6], and consists of two parts: an *encoder*, mapping each image into the *latent space*, and a *decoder*. A high level overview is as follows:

- Represent an image by $\boldsymbol{x}^{(i)}$, the length 784 vector of its pixel brightnesses.
- Encode this image into a mean vector $\boldsymbol{\mu}^{(i)}$ and variance vector $\boldsymbol{\sigma}^{(i)}$ with dimension n_z. Typically, this dimension is much smaller than that of $\boldsymbol{x}^{(i)}$.
- Sample a vector $\boldsymbol{z}^{(i)}$ from $\mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)})$ with diagional covariance matrix. In reality, a *reparametrisation trick* (see e.g. [1]) is performed to ensure the model is trainable, but this does not change the results of a feedforward run through the autoencoder). The density of $\boldsymbol{z}^{(i)}$ given $\boldsymbol{x}^{(i)}$ is denoted by $q_\phi(\boldsymbol{z}^{(i)}|\boldsymbol{x}^{(i)})$, where ϕ represents all parameters in the encoder.
- Decode $\boldsymbol{z}^{(i)}$ back into image space using a decoder $p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i)})$, where θ represents all parameters in the decoder.

Figure 2 provides a graphical overview of the encoder and decoder with all the neural network layers. The encoder and decoder are symmetric, so that the number of dimensions after e.g. the first convolutional layer in the encoder is equal to that before the last one in the decoder. The trainable parameters are ϕ (encoder) and θ (decoder).

In a traditional autoencoder, the loss function is some reconstruction error between $p_\sigma(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i)})$ and the input $\boldsymbol{x}^{(i)}$. In a variational autoencoder, we want to generate new images. This is usally done by decoding specific values of $\boldsymbol{z}$ from the latent space. For this reason, we need to ensure that the latent space is "meaningful", in the sense that a $\boldsymbol{z}$ value halfway between a shoe and trousers is some pictorial intermediate of the two.

This is usually done by enforcing that the distribution of (noisy) $\boldsymbol{z}^{(i)}$ values throughout the training dataset is close to a unit normal. This can be achieved by adding a penalisation term to the loss function. A convenient choice is the Kullback-Leibler divergence between the distribution of $\boldsymbol{z}^{(i)}$ values and $\mathcal{N}(0, 1)$, since this term can be expressed analytically. The loss function we use to train is hence:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{N_i} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}) \tag{1}$$

where the index i corresponds to the the (training) images and

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}^{(i)}) = \frac{1}{2} \sum_{j=1}^{n_z} \left(1 + \log\left((\sigma_j^{(i)})^2\right) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)}) \tag{2}$$

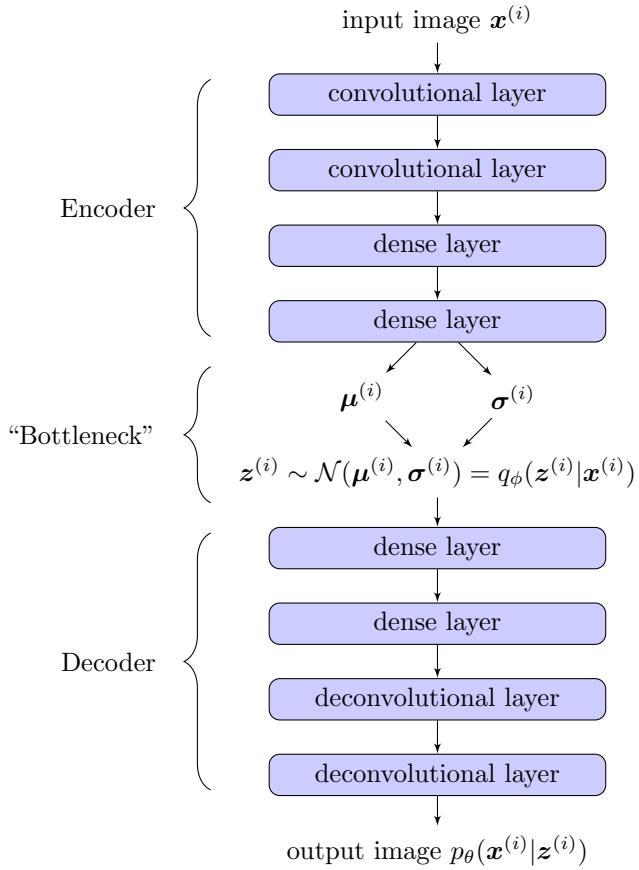

Figure 2: Variational autoencoder architecture. The input image $\boldsymbol{x}^{(i)}$ (a length 784 vector) is "encoded" into values of μ and θ, each with dimension n_z. A noisy value of $\boldsymbol{z}^{(i)}$ (with dimension n_z) is drawn from a normal distribution with these parameters (using the reparametrisation trick to ensure the model is trainable). $\boldsymbol{z}^{(i)}$ is subsequently "decoded" back into a vector of length 784.

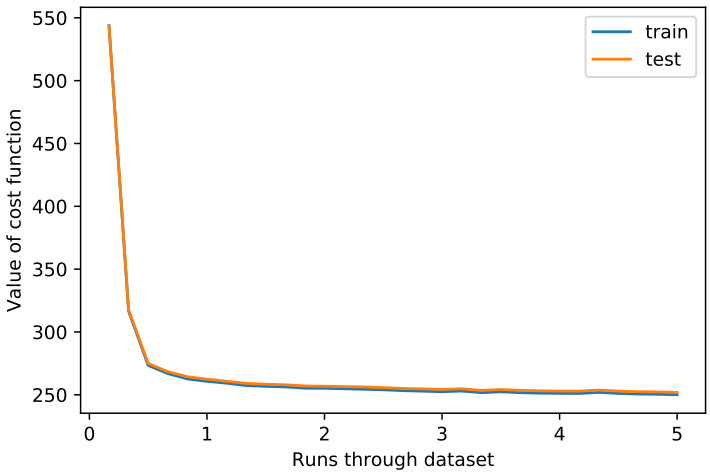

Figure 3: Value of loss function as model is trained.

with $\mu_j^{(i)}$ the jth component of $\boldsymbol{\mu}^{(i)}$, $\sigma_j^{(i)}$ defined the same way and $\boldsymbol{z}^{(i,l)}$ one of L samples randomly drawn from $\mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)})$. The first term is an analytic expression for the Kullback-Leibler divergence of the $\boldsymbol{z}$ space encoded by q_ϕ and the second measures the reconstruction error. A traditional autoencoder would only include the second term. Note that the encoder parameters ϕ are implicitly present in the the second term through $\boldsymbol{z}^{(i,l)}$.

4 Training

To train the model, we use Tensorflow's inbuilt *Adam* optimiser, along with the training/testing split provided by the dataset. We train in batches of size 100, up to five full runs through the training dataset (epochs). The training and testing errors are shown in Figure 3, with the expected results. Since the value of the cost/loss function is not easily intepretable, we also plot 10 reconstructions as a function of the number of epochs of training. The original images are shown in the last row as comparison. Note that the autoencoder does well in reconstructing the overall shape of the images, but blurs them considerably and, for this reason, fails to capture more detailed features such as the designs on t-shirts. This is often seen when minimising a loss function without extra adjustments, and is a problem common in (variational) autoencoders [3, 4].

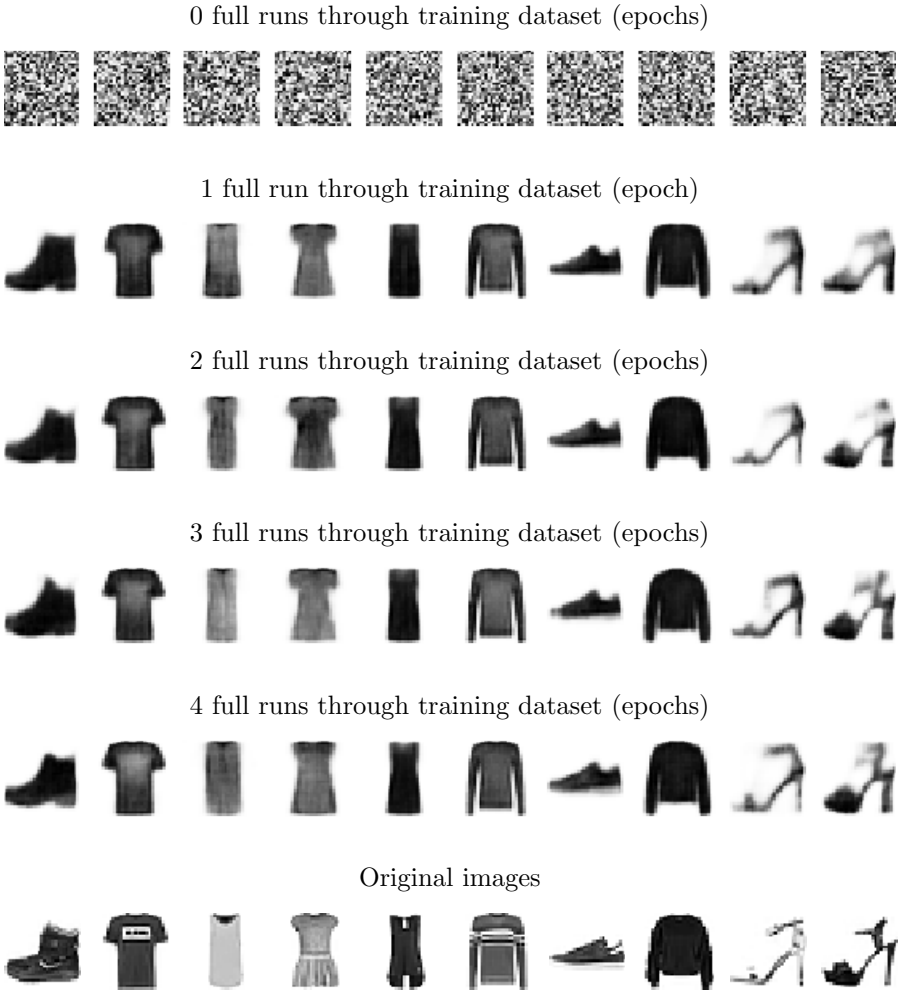

Figure 4: Autoencoder outputs for the first 10 images of the dataset after n full runs through training dataset (epochs), from $n = 0$ (top) to $n = 4$ (bottom). The last row shows the corresponding original images.

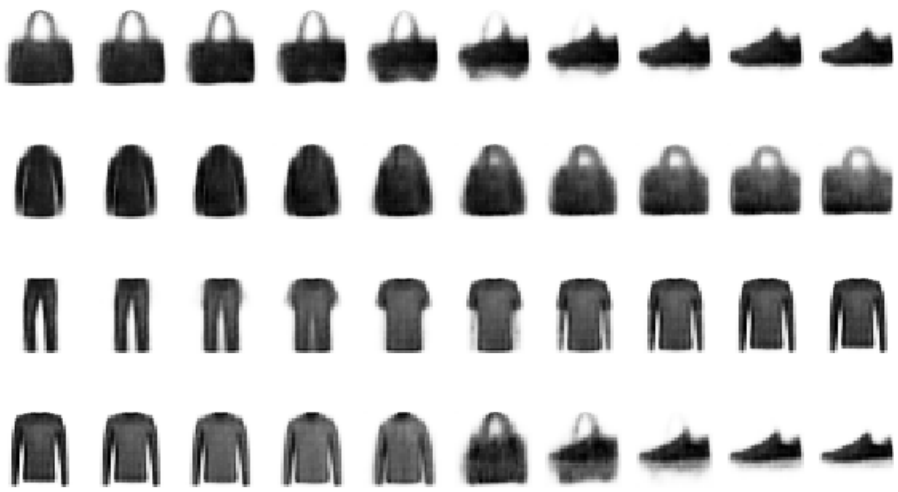

Figure 5: Interpolations in latent space between selected images.

5 Generating images

The variational autoencoder was chosen for its potential as a *generative* model, so we want not just to recreate available images, but also to create new ones. The idea is that the latent space in which $\boldsymbol{z}$ lives has some meaning, and we can create images with custom features by decoding suitable $\boldsymbol{z}$ values. In this section, we explore to what extent we have made this possible.

5.1 Interpolating

We start by interpolating images in latent space. Interpolations in the original 784-dimensional image are superpositions of pixel brighntesses and do not provide images resembling real clothing. We hope that interpolating in latent space returns a continous mapping from one image to another, where each intermediate image looks like an item of clothing.

Interpolations in latent space between various images are shown in Figure 5. The results are roughly what was expected: an image "morphs" continuously into another one. Interestingly, there are sometimes intermediate steps where the interpolation "passes through" another item. For example, in the last row of Figure 5, a pullover becomes a bag and only then a sneaker.

5.2 Adding custom features

Interpolations provide only a limited form of generative power; we would also like to create images with custom features. Suppose, for example, we want

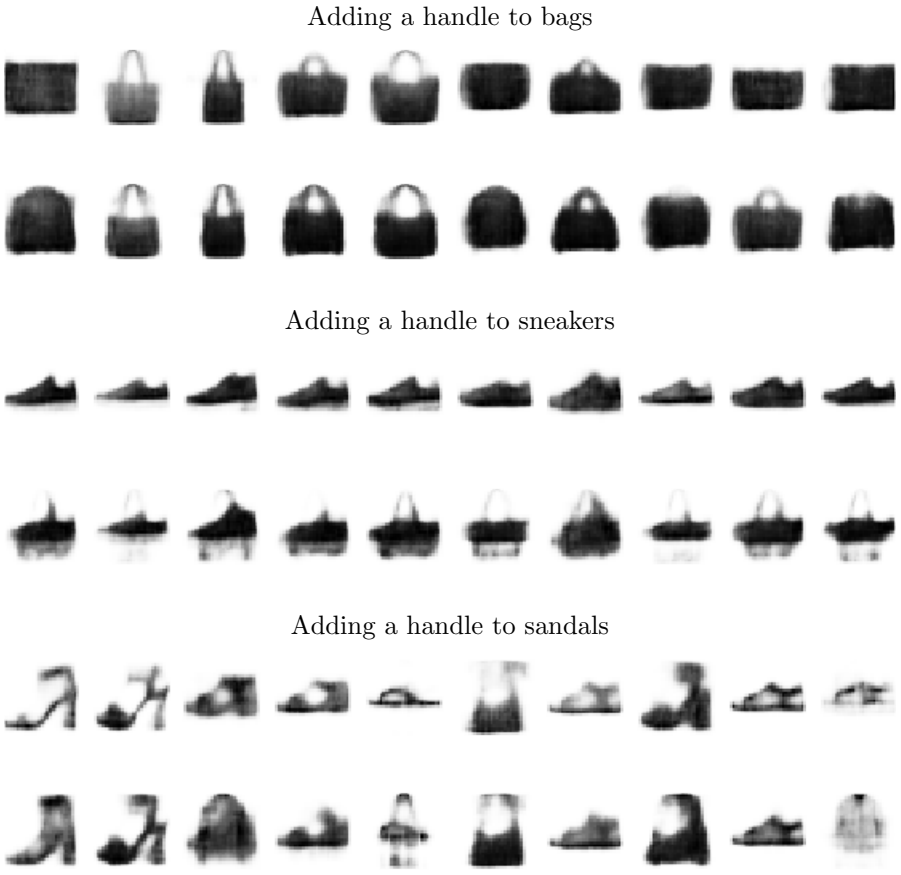

Figure 6: "Adding" a handle to various images in latent space. The unmodified pictures are shown above those with an "added" handle.

to generate an image of a sneaker with a handle added to it. We can do this via

$$\begin{aligned} \text{sneaker with handle} &= \text{sneaker} + \text{handle} && (3) \\ &= \text{sneaker} + (\text{bag with handle} - \text{bag without handle}) && (4) \end{aligned}$$

where all addition occurs in latent space.

We use this approach to "add" handles to various articles of clothing. The results are displayed in Figure 6. The first two rows are bags, with the original (autoencoded) images above and those with an added handle below. The approach does not work perfectly: it tends to add features that somewhat resemble a handle but would probably not be recognised as such by a human. For example, for the first bag (left top image of Figure 6), the bag gets a bulge where a handle would be, but there is no "loop". For bags already containing a handle, it tends to be come slightly thicker. The only bag for which a legitimate handle is added is the 9th (top row, second from right). However, it and the image below it are the images the difference was taken over to isolate the handle in the first place.

For sneakers (third and fourth row), the resultant images do appear to have a handle, but the sneaker shape has been destroyed, so that we are left with blob-like shapes instead of shoes. For sandals, it is very difficult to interpret the bottom row as images. However, interestingly, the last image (right bottom) looks like a coat, evidencing the complex structure in which $\boldsymbol{z}$ values live.

6 Discussion

Most interesting behavior is related to the structure of the latent space and this is the most obvious area of improvement. If we can make latent space "meaningful", we may have a powerful generative tool. However, the success of this may require more careful tuning. For example, the Kullback-Leibler divergence as a penalisation term can be viewed as somewhat simplistic, and a more sophisticated loss function may give more control over the distribution of $\boldsymbol{z}$ values. We can also study the latent space in more detail in the hopes of understanding its structure better.

Another drawback is the considerable blur on (re)constructed images. This blur remains even when the "obvious" improvements of more hidden layers, nodes or filters are applied, and is a feature of simple reconstruction error loss functions. Some papers (e.g. [3]) recommend adjustments to alleviate this issue.

References

[1] Webster, Kevin; Richemond, Pierre Harvey. *Deep Learning*. Imperial College London, October 2018. https://www.deeplearningmathematics.com.

[2] White, Tom. *Sampling Generative Networks*. 2016.

[3] Lamb, Alex; Dumoulin, Vincent; Courville, Aaron. *Discriminative Regularization for Generative Models*. 2016.

[4] Frans, Kevin. *Variational Autoencoders Explained*. 2016. http://kvfrans.com/variational-autoencoders-explained/.

[5] Altosaar, Jaan. *Tutorial - What is a Variational Autoencoder?*. 2017. https://jaan.io/what-is-variational-autoencoder-vae-tutorial/.

[6] Shafkaat, Irhum. *Intuitively Understanding Variational Autoencoders* 2018. https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf