Unsupervised Deep Learning

Marco Ballarin 1228022
marco.ballarin.6@studenti.unipd.it

Abstract

In this report we will review several tasks related to unsupervised learning, focusing on the models called autoencoders. We will optimize their architecture and employ them for different tasks, like image generation, denoising and transfer learning. We will finally implement a more advanced type of autoencoder, the Variational Autoencoder, and analyze its performances. We will use the MNIST dataset.

1 Introduction

In this report we will tackle different problems using the unsupervised deep learning framework. This means that we will have datasets D formed of couples $\{(x_i)\}_{i=1,\dots,N}$, where x_i are the samples. We will use the MNIST dataset, an exhaustive dataset containing handwritten digits from 0 to 9 using 28×28 images. We will use Pytorch as programming framework, and in particular Pytorch Lightning [1].

The report will be divided into four main Sections:

1. Introduction, where we introduce our work;
2. Autoencoder, where through an encoder and a decoder we generate sample images;
3. Denoiser, where we introduce some noise on the images to reconstruct, creating a network able to eliminate that noise;
4. Fine tuning, where we train a supervised classificator, starting from the trained encoder;
5. Variational Autoencoder, where we improve our approach using more advanced techniques;
6. Encoded space analysis, where we analyze the space of the encoded images and how it varies from the autoencoder to the variational autoencoder;
7. Appendix, where we will report images.

2 Autencoder

2.1 Methods

We will use a convolutional autoencoder, i.e. an autoencoder which uses convolutional layers, which are particularly useful in the case of images. An autoencoder is formed by two main structures:

- An Encoder, which takes the input and transforms it in a limited series of parameters $\{\xi_i\}_{i=1,\dots,N}$ which captures the characteristics of that input, and we call the space where these parameters lives, i.e. $\mathbb{R}^N$, encoded space. Its architecture is shown in Figure 1;
- A Decoder, which has the same structure of the encoder but mirrored. It transforms points in the encoded space back into samples of the same type of the input. Its architecture is shown in Figure 2.

To avoid vanishing problems in the gradient we will use a rectified linear unit (ReLU) as activation function. We will implement a regularization procedure, to better generalize outside of the training set. We choose to use the L2 regularization, which consists of adding a penalty in the loss proportional to the 2-norm of the parameters, i.e. weights and biases. In pytorch this method is enabled not in the loss function but in the optimizer, using the `weight_decay` keyword.

We will try two different optimizers:

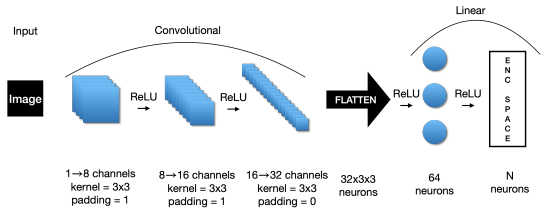

Figure 1: Architecture of the Encoder structure of the Autoencoder.

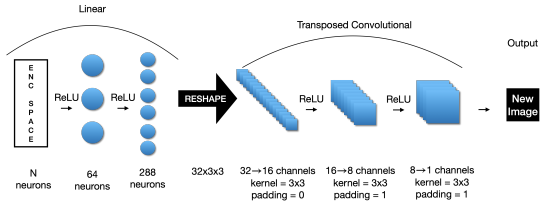

Figure 2: Architecture of the Decoder structure of the Autoencoder.

1. The stochastic gradient descent with momentum, which is the really basic algorithm for optimization. The addition of the momentum, i.e. the memory of previous steps, really helps to go out of local minima and speeds up convergence;

2. the Adam algorithm, adaptive moment estimator, a more advanced algorithm that uses estimations of first and second moments of the gradient to adapt the learning rate of each weight of the neural network.

To optimize the hyperparameters of the network we will use the Optuna framework[2], which consists of a python library which performs a bayesian optimization of the hyperparameters. Bayesian optimization basically infere the next best parameters based on the previous tests. It is a more advanced and better approach then grid/random search, since it "learns" from previous tests, while the grid/random search are completely memoryless. Indeed, we already shown of being able of implementing a cross validation setup in Homework 1, and since it has been proved that the Cross Validation is an approximation of bayesian optimization[3] we will not implement it in this homework. Furthermore, the MNIST dataset is quite exhaustive with respect to its statistical variability, and a cross validation technique is not necessary to avoid overfitting. We will also make use of an early stopping, i.e. we will prune trials that do not improve their performances. The hyperparameters that we will optimize are:

- The weight of the L2 regularization;
- The optimizer, between the Adam and the SGD with momentum;
- The learning rate of the optimizer;
- The encoded space dimension;

We will indeed keep the dimension of the encoded space small, since we want to compress as much as possible the informations in the image.

To quantify the performances of the autoencoder we will use the mean square error loss between the original image and the one reconstructed by the network.

2.2 Results

The hyperparameter search started from a very lucky point, as we can see from Figure 5. We can observe the reconstruction loss in Figure 6, and we see that the loss converges quickly to low values. On Figure 7 we can observe instead all the different combination of hyperparameters with their validation loss. It is really clear that the Adam is the best optimizer for this task. The regularization has a very low value, and the learning rate is really near the higher bound. This might suggest that a new search should have been implemented, increasing the higher bound on the learning rate. What is really interesting is, however, that the encoded space dimension $N = 9$ is the one performing better. This suggests that $N = 9$ is a sufficient dimension, and further enlargements are not needed.

We trained again a network using 100 epochs and we present in Figure 8 examples of the reconstructions on the test set, with a test loss of 0.0164.

3 Denoiser

3.1 Methods

The denoiser has the very same architecture of the autoencoder presented in Section 2. The only difference is that, before giving the images in input to the encoder we apply to it some random noise. Different types of noises can be used, but we implemented a Gaussian noise $\eta \sim \mathcal{N}(\mu, \sigma)$, which parameters are gaussianly distributed, i.e. $\mu \sim \mathcal{N}(0, 1)$, $\sigma \sim \mathcal{N}(0.5, 0.5)$. In this way the network is trained to reconstruct different variations of white noise. We used the same hyperparameters inferred in Section 2.

3.2 Results

The denoiser implemented as explained above is performing well, with a test loss of 0.03. As we can see in Figure 3 it is able to denoise the images in a coherent way, even though with some errors.

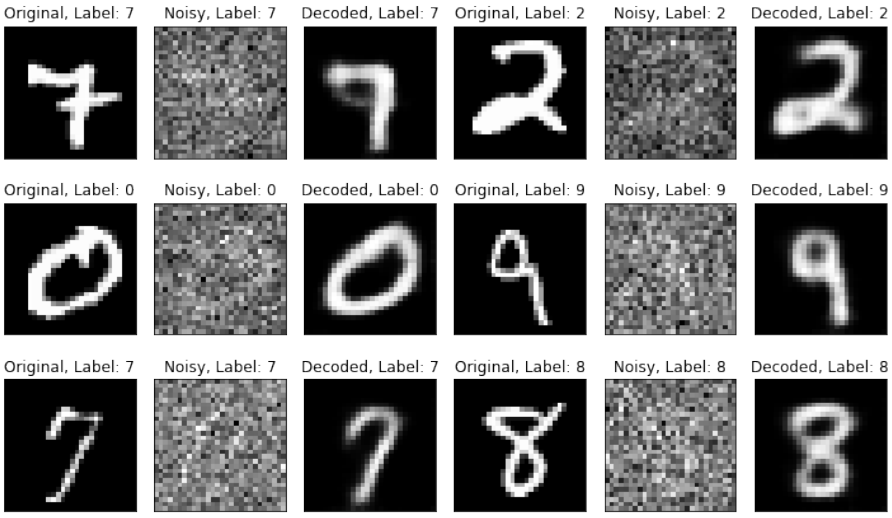

Figure 3: We present the original image, the noisy image and the denoised one for different samples of the test set. The results are really good, even though one of the digit is not denoised in the correct way, since it is almost impossible to understand which digit is represented in the noisy image for a human.

4 Fine Tuning

4.1 Methods

In this task we will use the prieviously trained encoder and apply the transfer learning technique. We use a pretrained network to speed-up the training phase of a classification task. We extract from the autoencoder the Encoder architecture, and then add two new linear layers to perform the classification. We freeze the parameters of the Encoder, i.e. we do not perform learning on that part of the network. We instead train the new layers, which consist of:

- An hidden layer with 64 neurons;
- The output layer with 10 neurons.

We use as activation function a ReLU and a negative log likelihood loss, which is optimal when we have the log-probabilities of the different input classes as output of the network, as in this case.

What is really important is that, even though the full network has about 26 300 parameters we only have to train 1 300 of them, speeding up the task.

4.2 Results

Even though 100 epochs were given as the maximum number of epochs, 49 were sufficient to achieve convergence. For this task we pick as test metrics the accuracy, i.e. the number of correctly classified samples over the number of total samples. We were able to achieve an accuracy of 0.94, which is quite impressive compared to the accuracy we achieved in the first homework, where we tuned all the hyperparameters and trained the whole network to perform the classification task, reaching an accuracy of 0.999. We can state that the transfer learning is a very powerful technique, since applying just a hidden layer and the output layer to a pre-trained encoder produced an high test accuracy.

5 Variational Autoencoders

5.1 Methods

The encoded space of a classical Autoencoder can be very irregular, and small perturbations can lead to very big changes, as we will see in Section 6. To eliminate this problem a new type of Autoencoder has been introduced, called Variational Autoencoder. We so make the mapping probabilistic: the encoder now gives as output a vector of means $\vec{\mu}$ and the covariances $\vec{\sigma}$ of the input distribution over the encoded space. To reduce the complexity we do not generate a $N \times N$ covariance matrix, but simply assume that it is diagonal, using only a vector. We then regularize the loss function l such that the resulting distribution is as close as possible to a multivariate gaussian using the Kulba-Leiber divergence. Calling x the original sample and $\hat{x}$ the reconstructed one:

$$l = MSE(x, \hat{x}) + \lambda KL(\mathcal{N}(\vec{\mu}, \vec{\sigma}), \mathcal{N}(\vec{0}, \vec{1})) = MSE(x, \hat{x}) + \frac{\lambda}{2} \sum_{i=1}^{N} \left(\sigma_i^2 + \mu_i^2 - 1 - \ln(\sigma_i^2)\right)$$

where with MSE we denote the mean square error and λ a parameter to control the importance of the "gaussian regularization".

There is however a problem: the sampling is a discrete process, along which we cannot backpropagate. We so introduce a trick that mimicks the sampling, i.e. if we have $\vec{z} \sim \mathcal{N}(\vec{\mu}, \vec{\sigma})$ we can write:

$$\vec{z} \simeq \vec{\mu} + \vec{\zeta}\vec{\sigma} \quad \text{with } \vec{\zeta} \sim \mathcal{N}(\vec{0}, \vec{1}) \tag{1}$$

We can now apply the backpropagation.

5.2 Results

The variational autoencoder seems to perform well. It was particularly important to tune the parameter λ, since it strongly affected the results. A too high λ simply resulted in a blurred image for every input, while a too low λ produced the same results as the classical autoencoder (as expected). We finally fixed it at $\lambda = 10^{-5}$. However, the variational autoencoder works correctly, with a test loss of 0.05. The encoded space seemed more regular, as we can observe in Figure 4, where we kept fixed all the components of the encoded samples but one.

6 Encoded Space Analysis

6.1 Methods

All the information of the input are reduced in the encoded space. We will now study this space, using dimensionality reduction tools, i.e.:

- Principal Component Analysis (PCA), a linear technique that investigates the dimensions with higher variability in the data;

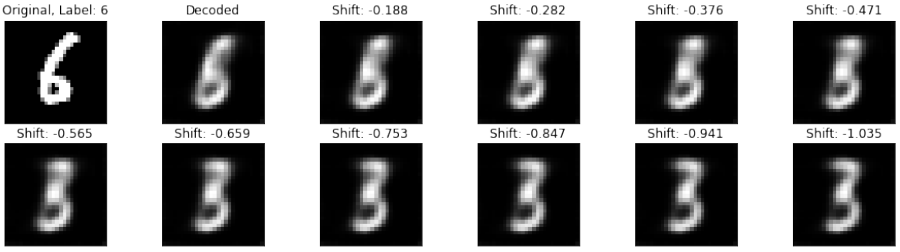

Figure 4: Smooth evolution of the digit 6 by varying the 0-th component of its encoded representation by 1/5 of its original value in each of the figures. The images should be read from left to right, starting from the top row. We notice that the digit changes smoothly from 6 to 3.

- t-distributed stochastic neighbor embedding (t-SNE), a non-linear technique. The algorithm tries to mantain the local distance between points, i.e. keeps points that were distant in the high-dimensional space distant in the reduced space.

We will also put particular attention in the differences between the classical autencoder encoded space and the one of the variational autoencoder.

6.2 Results

I found this to be the most interesting task. We firstly analyzed the encoded space dimension of the classical autencoder using the PCA linear technique, as we can see from Figure 9. However, the different digits seemed not well divided. This behaviour was in contrast with the results obtained in Section 2. We so further study the problem using the t-SNE, and the results can be seen in Figure 10. In this case the points cloud are more separeted, helping us in the understanding the correct results obtained previously. This representation was, however, quite cumbersome, since interpreting the points clouds was not trivial. We so decided to study a new quantity, defined as follows. First, we reduce the dimensionality of the problem d using our tools to $d = 2$, and then we take the average and the standard deviation of the these coordinates for each digit, plotting ellipses which center is in the average, and have as axis the standard deviations. We see the results of this plot in Figure 11.

We then repeat the same analysis for the variational autencoder, as we can see from Figures 12, 13 and 14. By confronting the behaviour in the two cases we can observe that in the VAE the ellipses are nearer to each other, in particular when digits presents similar patterns. We can also observe that, with respect to the classical autencoder, in the VAE the ellipses are more similar to circles, following the constraint of a normal distribution $\mathcal{N}(\vec{0}, \vec{1})$, which has the same standard deviation along all the directions. We report in Table 1 the average absolute value of the difference between the standard deviation along the first encoded variable and the second one.

Autoencoder		Variational	
PCA	t-SNE	PCA	t-SNE
3.4	4.7	0.07	2.7

Table 1: Average absolute value of the difference between the standard deviation along the first encoded variable and the second one, looking at both the Autoencoder and the VAE, for PCA and t-SNE.

We can conclude that the VAE really has a more regular encoded space, and that the more informative way to do dimensionality reduction in this case is to use the t-SNE, i.e. to keep the distances along the procedure.

Finally, we report in Figure 15 some images generated sampling the encoded space with a gaussian distribution for the autoencoder, while in Figure 16 we show the images obtained from the VAE with the same procedure. Again, we can observe that the regularity of the encoded space of the VAE generate better images when we perform a totally random sample.

References

[1] WA Falcon. "PyTorch Lightning". In: *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* 3 (2019).

[2] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[3] Edwin Fong and CC Holmes. "On the marginal likelihood and cross-validation". In: *Biometrika* 107.2 (2020), pp. 489–496.

7 Appendix

We present here images that can be interesting but are not fundamental for the report.

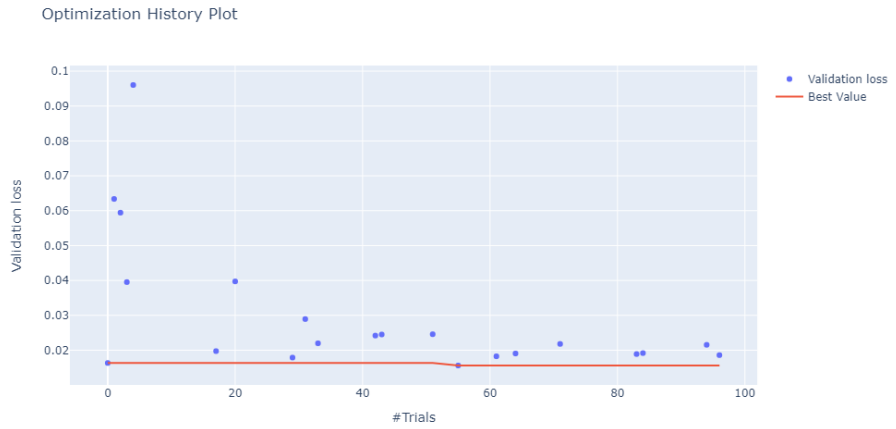

Figure 5: Optimization trend of the validation loss. On the y-axis we report the final validation loss, while on the x-axis the trial number. The red line represents the evolution of the best score.

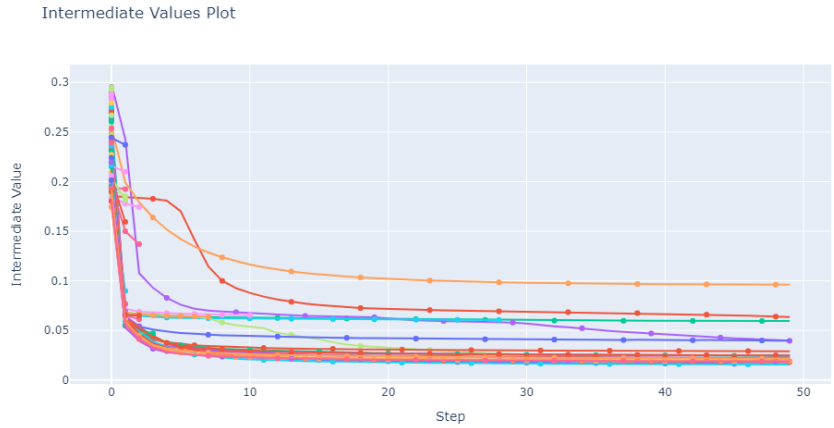

Figure 6: Evolution of the validation loss. While some trial is pruned at the beginning for a poor performance we can observe that the validation loss converges quite quickly.

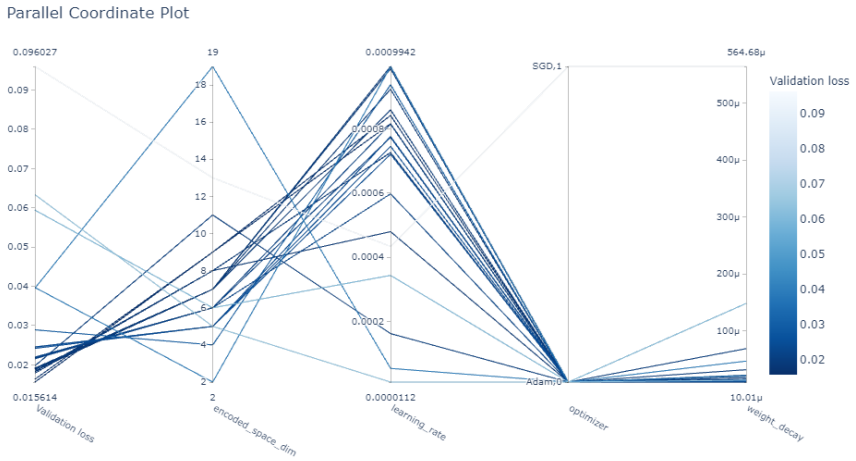

Figure 7: Visual representation of the used hyperparameters, where the color of the line connecting the parameters used in a single trial is related to the validation loss as shown in the colorbar. We can observe that the stochastic gradient descent is not performing well.

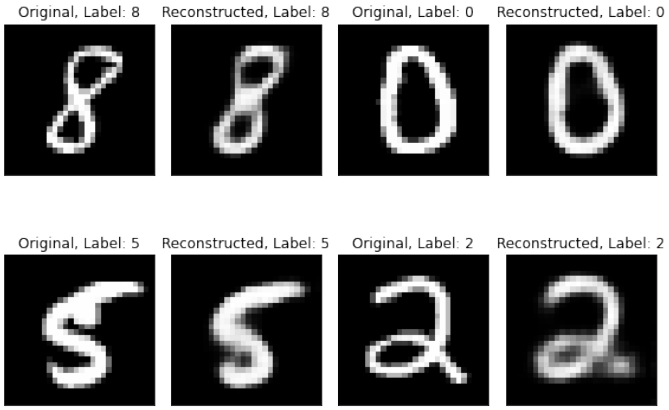

Figure 8: Reconstruction of the Autoencoder with optimized hyperparameters

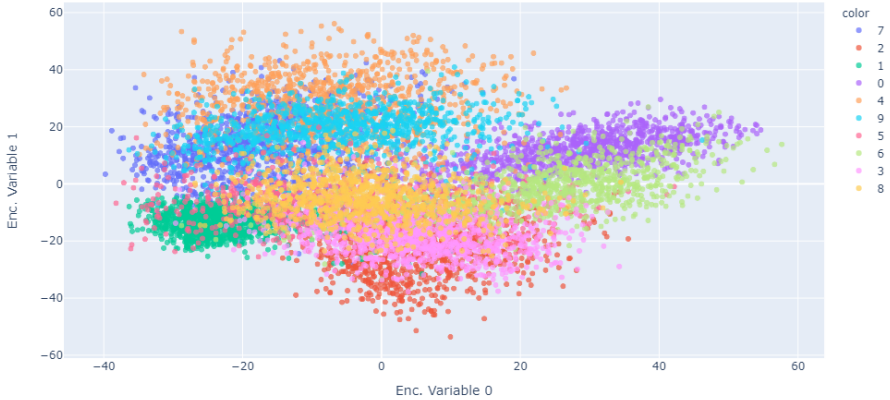

Figure 9: Encoded space with dimensionality reduced with PCA for the classical Autoencoder, using as points the test set.

Figure 10: Encoded space with dimensionality reduced with PCA for the classical Autoencoder, using as points the test set.

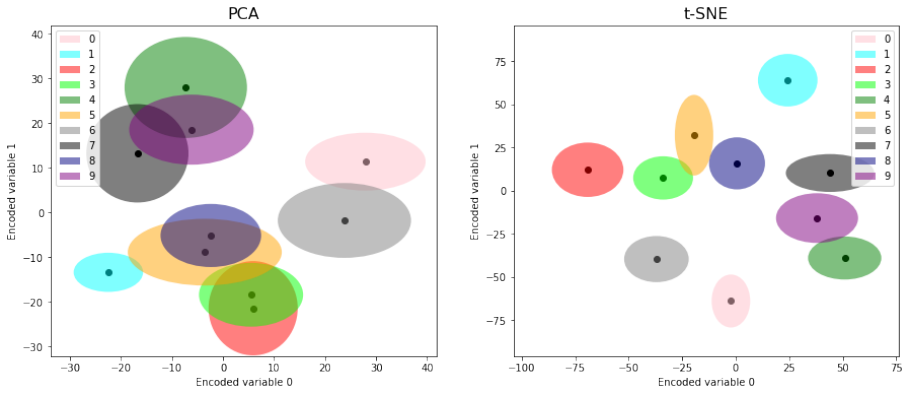

Figure 11: Encoded space with dimensionality reduction for the classical autoencoder, where the center of the ellipse is the average position of a given digit, while the ellipse represents its standard deviation along the two different coordinates.

Figure 12: Encoded space with dimensionality reduced with PCA for the Variational Autoencoder, using as points the test set.

Figure 13: Encoded space with dimensionality reduced with PCA for the Variational Autoencoder, using as points the test set.

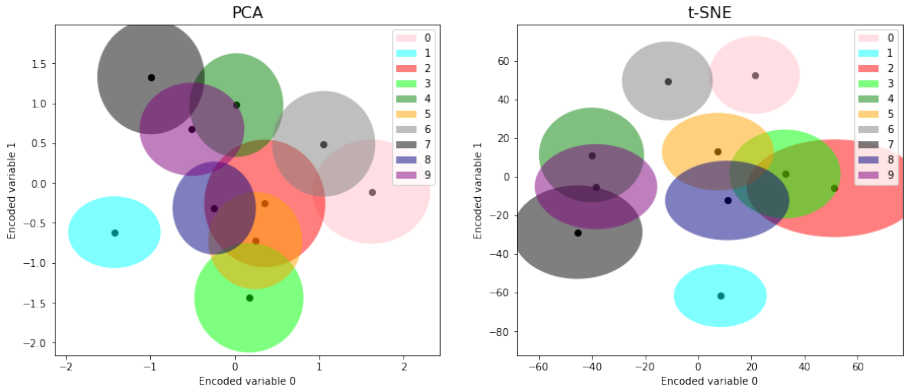

Figure 14: Encoded space with dimensionality reduction for the Variational autoencoder, where the center of the ellipse is the average position of a given digit, while the ellipse represents its standard deviation along the two different coordinates.

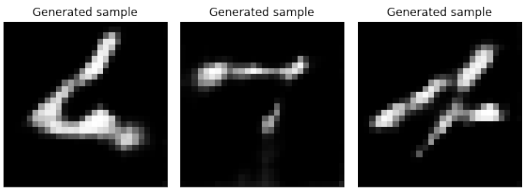

Figure 15: Images generated by sampling the classical autencoder using a normal ditribution.

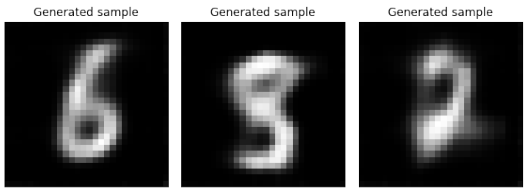

Figure 16: Images generated by sampling the variational autencoder using a normal ditribution.