Project – Auto-encoding Variational Bayes

Quentin Ravaux, Malo Adler

Group 56

Abstract. This paper aims at implementing and better understanding the concept of Variational Auto-Encoder, a particular model of auto-encoder which learns a distribution over the latent space. It allows among others to generate new samples once it has been trained. We compared this VAE to a more classical Vanilla Auto-Encoder, in terms of log-likelihood and in terms of visual results. It has appeared that the VAE was less efficient on a single task, but capable of performing several tasks. We have also analysed the efficiency of increasing the dimension of the latent space.

1 Introduction

This project is based on the article "Auto-Encoding Variational Bayes" written by Diederik P. Kingma and Max Welling. The main goal of this article is to build an algorithm (AEVB: Auto-Encoding Variational Bayes) capable of training an approximate posterior inference model such as the VAE (Variational Auto-Encoder) and to compare it to other algorithms such as Wake-Sleep algorithm or Monte Carlo Expectation Maximisation method. The model that we are going to use is of the following shape:

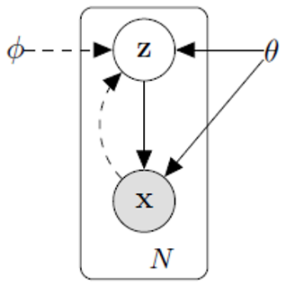

Figure 1: Model of the Variational Auto-Encoder

The goal is to create a system that can face 2 important problems. Indeed, the intractability of the marginal likelihood makes impossible its evaluation or differentiation, which forces us not to use EM algorithm. We can avoid this thanks to an approximator of the marginal likelihood: the approximator should be a neural network with a nonlinear hidden layer. The second problem is that the dataset is very large and this is too costly for algorithm like Monte Carlo EM. The objective is to get rid of these two problems and propose an algorithm that gives an efficient approximation of the parameters θ, the posterior inference of the latent value z given the observed x and the marginal inference of the variable x.

The basis of the AEVB consists in computing the marginal likelihood and to separate it into two terms:

$$\log p_\theta(x^{(i)}) = D_{KL}(q_\phi(z|x^{(i)}) \,||\, p_\theta(z|x^{(i)})) + \mathcal{L}(\theta, \phi; x^{(i)})$$

The first term corresponds to the Kullback-Leibler divergence between the true posterior and the approximated one. The second term corresponds to the variational lower bound on the marginal likelihood. We want to compute this second term and to maximize it in order to maximize the marginal likelihood. It holds that:

$$\mathcal{L}(\theta, \phi; x^{(i)}) = -D_{KL}(q_\phi(z|x^{(i)}) \,||\, p_\theta(z)) + \mathbb{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)]$$

The problem is that the gradient of the lower bound is really difficult to compute. And in the case of the VAE, the expectation leads to random variables in the neural network that makes the back propagation of the gradient impossible. In order to face this problem, we need an additional trick. The trick that is used is called reparametrization trick and consists in expressing the random variable $z \hookrightarrow q_\phi(z|x)$ thanks to a deteministic variable $g_\phi(\epsilon, x)$. In our case, $z = \mu + \sigma\epsilon$, with $\epsilon \hookrightarrow \mathcal{N}(0, I)$. Thanks to this trick, the variable z can be computed with deterministic coefficients and there is no problem with the back-propagation of the gradient in the neural network for the VAE.

Now we can compute the Variational Lower Bound thanks to this estimator (in the case of a Gaussian VAE):

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

where $z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$ and $\epsilon^{(l)} \hookrightarrow \mathcal{N}(0, I)$.
We want to maximize the lower bound so we make a gradient ascent on the estimator. Besides, the second term of the estimator depends of the type of output that we use (Bernoulli or Gaussian).

2 Methods

In order to compare the model and the methods presented in the article with known and more basic models, we re-implemented both a Variational Auto-encoder (VAE) and a Vanilla Auto-encoder. We applied both these models on the MNIST digits dataset. The goal for the Vanilla Auto-encoder is to learn a (coded) representation of the input data in a latent space (whose dimension is much less important than the one of the input space) and to learn to decode this representation. This encoder-decoder structure is learned with a neural network, which minimizes the reconstruction loss between the decoded version and the original version. We chose for this neural network a MLP structure of 1 hidden layer for the encoder made of 500 neurons (to keep a structure as similar as for the VAE as possible), and 1 hidden layer for the decoder. The input of the encoder is of size 28*28 (the size of the images of the MNIST dataset), and its output is of size the dimension of the latent space, which can be chosen. The structure of the decoder is the reverse structure of the encoder.

On the contrary, the goal for the VAE is to learn a probabilistic representation of the input data through a probability distribution in the latent space, and to learn to decode this representation. In our case, we worked with a Gaussian distribution in the latent space, and the structure of the neural network is the following. The encoder is composed of 1 hidden layer, its input is of size 28*28 and its output is of size twice the dimension of the latent space (the first part of the output will be the mean of the distribution, the second part will be the vector of the diagonal elements of the covariance matrix of the distribution). Before going into the decoder, the coded data must be reparametrized (because if we sample from this distribution, this probabilistic dependency on the parameters of the model will make the backpropagation of the network impossible): we therefore sample from a known generic distribution (here, a centered standard Gaussian distribution), and apply a deterministic transformation, using the parameters of the model, which allows us to bypass this backpropagation problem. The input of the decoder is then this generated sample, and the decoder in itself is composed of 1 hidden layer (composed of 500 neurons, as advised in the article for this dataset).

For the training of the VAE, we have used an Adam optimizer with a learning rate of 0.001 and a batch size of 100 samples. Wa have used the same method for the Vanilla Auto-Encoder. In order to compare the two different networks, we only used the second term of the loss function for the VAE, which corresponds to the reconstruction loss (that we expressed as the binary cross-entropy).

3 Results

The results of the testing of our trained models are given below. For several dimensions of the latent space ($N_z \in \{3, 5, 10, 20, 200\}$, as in the article), we have plotted the log-likelihood for both the VAE and the Vanilla model (for the VAE it is only a part of the loss that we consider to make the gradient ascent, since this loss is composed of another term, which is the Kullback-Leibler divergence). These plots are shown on Figure 7. The models have each been trained on 40 epochs, with a learning rate of 0.001 and a batch size of 100. It is clear here, given these figures, that the Vanilla Auto-encoder gives better results for the reconstruction part. It is however not that surprising, since the VAE is more general: it is able to perform other tasks at which the Vanilla AE is not good, see section Discussion).

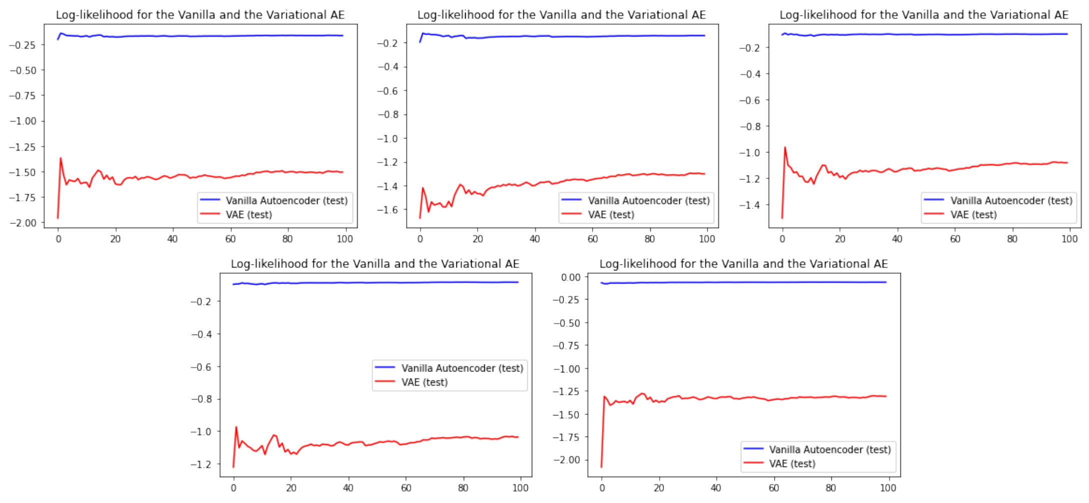

Figure 2: Log likelihoods for the VAE and the Vanilla AE for the test set. The latent space dimension changes with each image: its values are respectively $N_z = 3, 5, 10, 20, 200$

For each of these latent space dimensions, we have also represented the decoded version of several input images (see Figures 3 to 6). The first line shows several samples of the input images. The second one shows the output of the Vanilla Auto-encoder when the model is given this image as input, and the third one shows the output of the VAE when it is given the first image as input. As expected, the Vanilla AE is overall much better visually for the reconstruction.

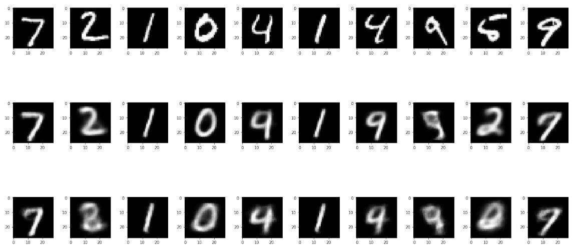
Figure 3: Examples of reconstructed images for the Vanilla AE (second line) and for the VAE (third line). The first line shows the original inputs, the latent space dimension is $N_z = 3$

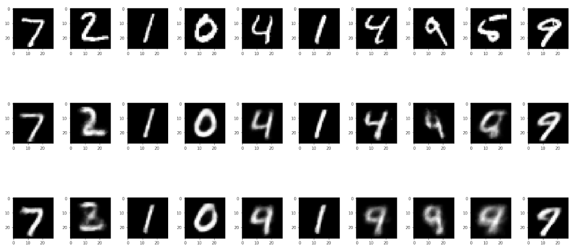
Figure 4: Examples of reconstructed images for the Vanilla AE (second line) and for the VAE (third line). The first line shows the original inputs, the latent space dimension is $N_z = 5$

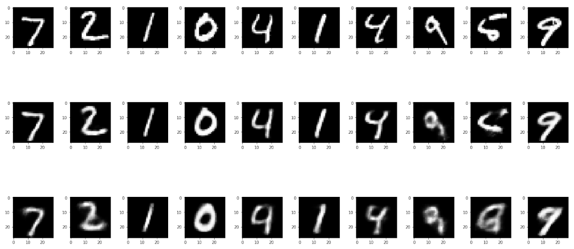
Figure 5: Examples of reconstructed images for the Vanilla AE (second line) and for the VAE (third line). The first line shows the original inputs, the latent space dimension is $N_z = 10$

To see the influence of the dimension of the latent space on the performances of the VAE, we have plotted the variational lower bound for model with these different latent space dimensions. The results are shown on Figure **??**. It is striking to see that, among the values tested, the values $N_z = 10$ and $N_z = 20$ give the best results in terms of variational lower bound. And in particular, increasing the dimension of the latent space

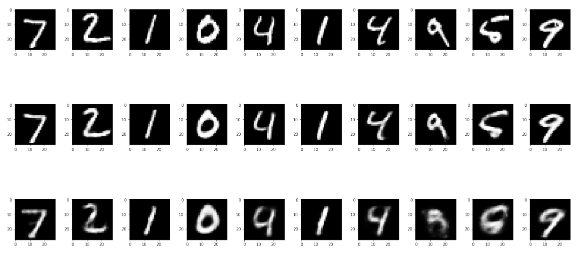

Figure 6: Examples of reconstructed images for the Vanilla AE (second line) and for the VAE (third line). The first line shows the original inputs, the latent space dimension is $N_z = 20$

a lot ($N_z = 200$), although it gives great results for the Vanilla AE, gives very poor results for the VAE.

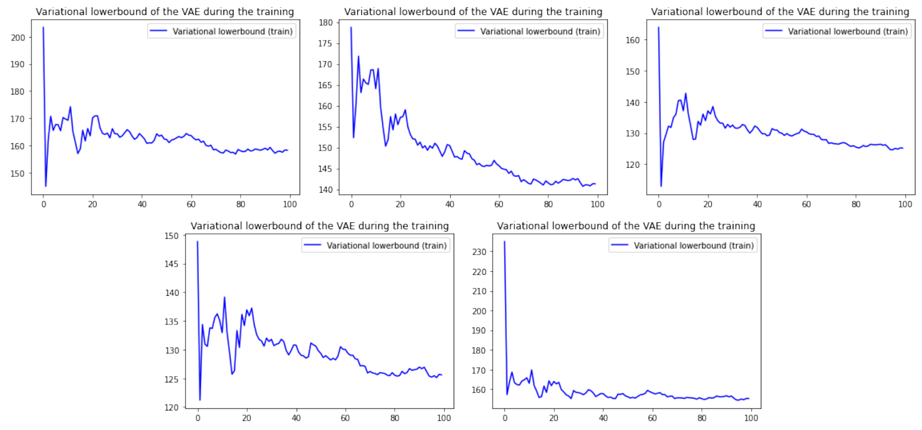

Figure 7: Average variational lowerbound for the VAE during the training. The latent space dimension changes with each image: its values are respectively $N_z = 3, 5, 10, 20, 200$

In the 2D case (for the latent space), it is possible to represent the latent space. This is shown in Figure 8.

4 Discussion

At the end of the article, they compare the AEVB algorithm with the wake-sleep and the Monte Carlo EM algorithm. They present two plots, one with 1000 samples in the train set and an other one with 50000 samples in the train set. The goal is to see the behavior of the algorithms with small data and large one. We can see that the wake sleep algorithm in both cases is under the AEVB with the same kind of convergence. However, the Monte Carlo algorithm converge as fast as the AEVB with 1000 samples in the train set but has a worse and slower convergence in the case of a 50000 samples train set. Indeed, the Monte Carlo EM uses a sampling loop per data point which is not a problem with a small dataset but is really problematic with a large dataset.

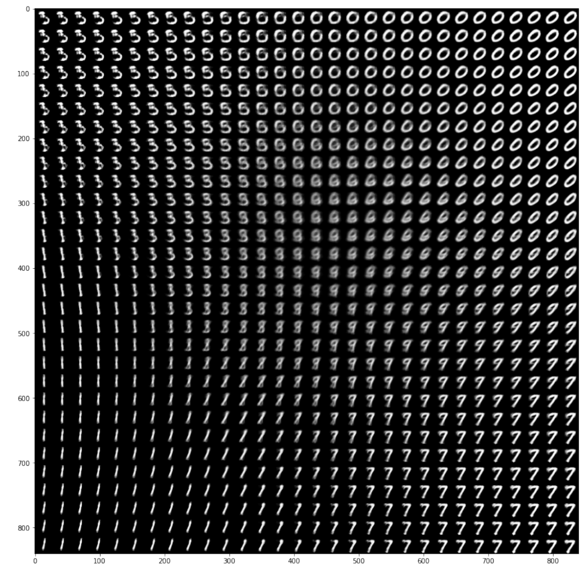
Figure 8: A representation of the latent space

We can argue the the plot presented in the algorithm uses the number of training examples as an abscissa and doesn't take the temporal complexity into account. For example, the sampling loop per data point of the Monte Carlo EM could be very long to compute.

We have seen in the Results section that for the reconstruction tasks, no matter the dimension of the latent space, the Vanilla AE that we trained was always better than the VAE. However, a major interest of the VAE is that it somehow learns the structure of the data in the latent space, contrarily to the Vanilla AE. As a result, the VAE is capable of generating new data, from random points of the latent space. This is what we show on Figure 9: we randomly sampled values for μ and σ, and then reconstructed the images corresponding to these latent variables. And the results for the VAE are quite satisfying: most of the images returned are indeed clearly visible and recognizable digits. On the other side, the Vanilla AE, which is very good only at reconstructing the data, loses this generative capacity: when doing the same thing with the Vanilla AE, the results are not conclusive at all (see Figure 10).

To learn more about the behaviors of the VAE and the Vanilla AE, we have considered the latent representation of a given digit, and we added some noise to this latent representation. We wanted to see how the VAE reacted to this noise, in comparison with the Vanilla AE. And the results are clear: the Vanilla AE (see Figure 12) does not give conclusive results: as the noise increases, the reconstruction gets worse. On the other side for the VAE, even when the noise increases, the reconstruction is not accurate but is still made of visible digits. It therefore seems that the Vanilla Auto-encoder is better than the VAE on the mere reconstruction of the data, but the VAE has the advantage of being capable to perform several tasks, including the generation of new samples.

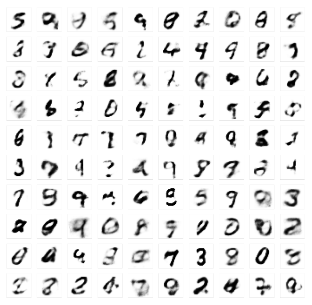
Figure 9: Generation of new data from random latent variables with the VAE

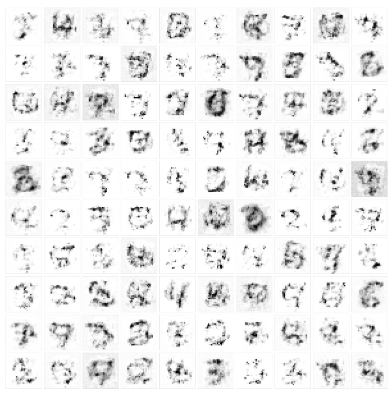
Figure 10: Generation of new data from random latent variables with the Vanilla AE

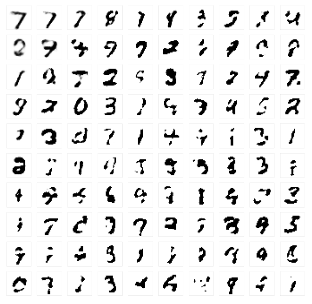
Figure 11: Generation of new data from latent variables with noise with the VAE

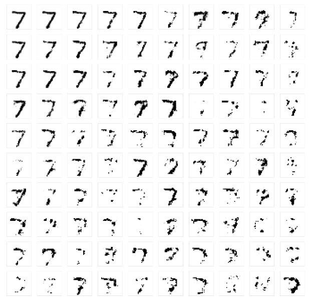
Figure 12: Generation of new data from latent variables with noise with the Vanilla AE