CS-433 FINAL PROJECT

Anonymous authors
Paper under double-blind review

ABSTRACT

Reproducibility of results obtained in research papers is an important part of the research process, as it allows to support the conclusions of the paper, or otherwise find potential inconsistencies in the results. The paper we have chosen to reproduce is called *MAE : Mutual Posterior-Divergence Regularization for Variational Autoencoders*, and aims to improve the performances of Variational Autoencoders (VAE) by constraining their latent variables as to ensure they do contain useful information. VAEs are one of two very powerful generative models and can be useful in many different fields, which is why we decided to study this paper.

1 INTRODUCTION

Our research history.

The paper *MAE : Mutual Posterior-Divergence Regularization for Variational Autoencoders* is based on research on Variational Autoencoders (VAEs) and tries to mitigate performance issues araising when the latent representation becomes uninformative, collapsing the model to an unconditional generative model. We had to read multiple papers about previous research on VAEs, as well as different Neural Network models used in the studied paper. We will here introduce those concepts and talk about the need for a better model than simple VAE as described in the first paper introducing the model (**?**).

1.1 VARIATIONAL AUTOENCODERS

We learned about VAE, how it works

what's an autoencoder

An autoencoder is a generative model and can be decomposed into two parts : an *encoder* and a *decoder*, which both can be many kinds of networks. The encoder generates a so-called *latent representation* of the original data received as input, which the decoder will then use to try and recover the original information. The goal is then to train the encoder to generate the most informative latent variables, and the decoder to reconstruct the original data as closely as possible. Such a model can be very useful for many reasons : an efficient latent representation can be useful for analysis and understanding of some natural process or for data representation tasks, and when used along with the decoder, it can be used to generate realistic data from small representations.

For the rest of our discussion, let x be an observable random variable which is assumed to be generated from a hidden continuous random variable z, following a conditional distribution $p_\theta(x|z)$.

Kingma and Welling in their paper wish to design an autoencoder that works well even on latent variables with intractable posterior distributions $p_\theta(x|z)$, i.e. that cannot be evaluated or differenciated. To this end, they introduce what they call a *recognition model* $q_\phi(z|x)$, an approximation to the true posterior $p_\theta(z|x)$. They also call $q_\phi(z|x)$ the *encoder* and consequently the conditional distribution $p_\theta(x|z)$ the *decoder*.

In fact, they point out that the marginal likelihood $\log p_\theta(x)$ can be lower bound by the following :

$$\log p_\theta(x) \geq \mathcal{L}(\theta, \phi; x) = -D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) + \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] \tag{1}$$

which they explain has too high a variance to be practical when estimated using a standard Monte Carlo gradient estimator, and where D_{KL} is the KL divergence of two distributions. To solve this

problem, they introduce a new lower bound that takes advantage of a *parameterization trick*, which consists in rewriting the distribution of z as a function of a noise variable ε :

$$z \sim q_\phi(z|x) = g_\phi(\varepsilon|x) \tag{2}$$

where ε follows an appropriate distribution $p(\varepsilon)$. This makes it possible, after replacing the lower bound in 10, to apply a Monte Carlo estimator on the second term, yielding :

$$\tilde{\mathcal{L}}(\theta, \phi; x) = -D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x, g_\phi(\varepsilon_l, x)) \tag{3}$$

where L is some sample size and ε_l is a sample from the distribution of ε.

1.2 AUTOENCODING CAPABILITIES OF VAEs

Read about VLAE, which explains well why VAE is not enough. Talk about KL-varnishing at some point

As a followup reading, cited in our paper of interest about MAEs, we looked at *Variational Lossy Autoencoders* (**?**), where we found good insight as to why VAEs on their own are not always sufficient for autoencoding.

They note that when the decoder is made too expressive, it is easy for the model to set the approximate posterior $q_\phi(z|x)$ to equal the prior $p_\theta(z)$ and hence avoid any cost from the regularizing term $D_{KL}(p_\phi(z|x)||p_\theta(z))$ in 3. Although this is explained by "optimization challenges" of VAE in most research , they present another observation that, even on a perfectly optimized VAE, ignoring the latent variables should still result in optimum results in most instances of VAE with intractable true posterior and powerful enough decoders.

1.3 CONVNET

Finally, as most experiments in our paper of interest make use of Neural networks for the VAE encoder and decoder, we describe here in essence what they are. Note that we focus on Convolutional Neural Networks (ConvNet) rather than Residual Neural Networks (ResNet), even though the latter is also used in the paper. We unfortunately have had technical and timing constraints, and Resnet being a fairly big model, we decided to focus on having good results on ConvNet alone.

Regular Neural Nets A Neural Network is composed of a sequence of hidden layers, each consisting of neurons which connect to all neurons of the previous layer. The first layer takes a single vector as input, and the last layer outputs the result. For example, in a classification problem, it outputs a score for each class.

Convolutional Neural Nets An issue with Regular Neural Networks is the fact that each neuron must be connect to all neurons of the previous layer, which makes the model impractically big for large input data. Convolution Neural Networks (ConvNet) take advantage of the fact that, in inputs that represent images, localized information in the image can be enough for learning. Each layer organizes its neurons in a three dimensions (width, height, depth) and each neuron is not constrained to be fully connected to the previous layer. The first layer then usually has the same dimensions as the input images, and for the example of a classification problem with c classes, the output data will have dimension $1 \times 1 \times c$.

2 EXPERIMENTS

2.1 OUR JOURNEY TOWARDS MAE IMPLEMENTATION

As a first step, we wanted to reproduce the VAE model. In order to do this we used the pytorch framework. We generated an encoder decoder structure, with the encode and the decoder as one layer, and our scheme is the following:

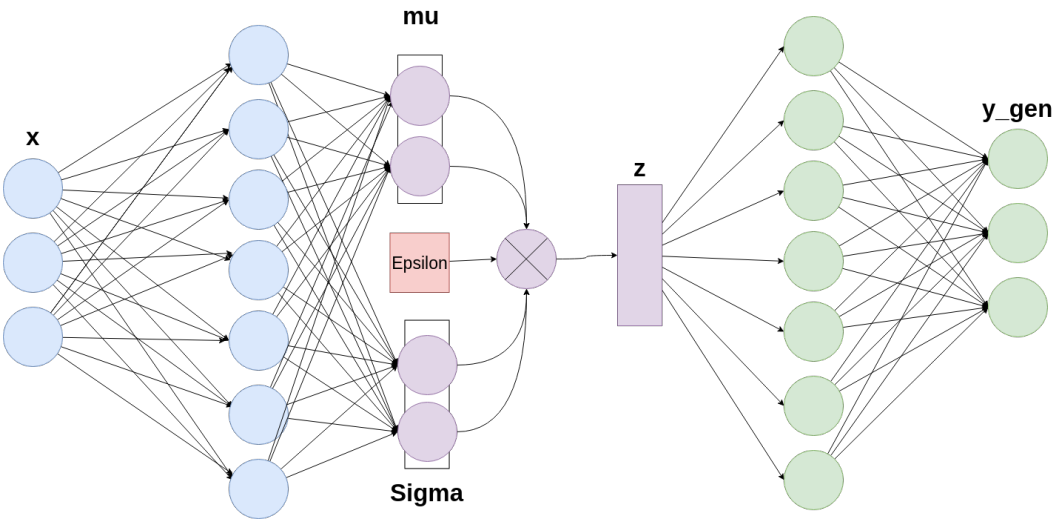


Figure 1: Our implementation, lower amount of neurons displayed

It is really close to a default implementation of some VAE. For us, the hidden layers (so the one between x and σ, μ) had to be bigger than the x dimensions, and we used 4x the dimension of x in out code. The mu and the sigma are the reparametrisation trick from the article, where in order to avoid sampling but still have some noise in the latent variable z, we use the following approximation:

$$z = \mu + \sigma \times \epsilon$$

Where epsilon is some random noise.

Then in order to train the model, we needed to have some loss, if we recall the one used in VAEs:

$$loss = -D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) + \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] \tag{4}$$

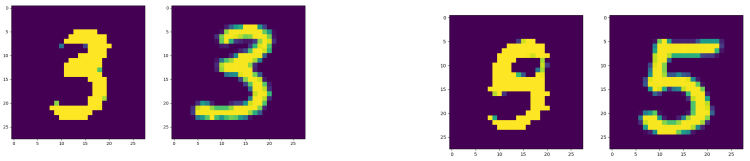
We can notice that it is in two parts, the D_{KL} one, which measures how the encoding distribution of z is similar to the real one, and the reconstruction loss, so how likely are our generated images knowing the distribution of z, which is generated by our encoder. For the first part of it, we applied the loss described in appendix B of the article, where they assume that the prior distribution of z is a standard gaussian ($N(0, I)$), and where our estimated distribution of z knowing x is a gaussian ($N(\mu, \sigma)$). We then get an estimation for D_{KL} using those settings as

$$-D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) = \frac{1}{2}\sum_{j=1}^{J}(1 + log(\sigma_j^2) - \mu^2 - \sigma^2) \tag{5}$$

For the reconstruction loss, we can just apply some known loss function to know how good is our model performing. We tried a few, and were quite surprised sometimes, since the MSE loss for example is performing really badly and we get almost all the time the same image. The one we finally used is the BCE_loss, since we have normalized images, with pixel values normalized, since it's the one that is performing the better, even if we were also surprised at first.

Finally, the last setting that we found important in order to get good results, is to use the sum over all individual batch element's loss, and not the mean over all individual batch element's loss, since this was improving the results a lot.

This allowed us to get those results, as an example, for the reconstructions we got from when feeding some images to our model4:



(a) (b)

Figure 2: Left: VAE reconstruction, right: original MNIST image

For the remaining of the samples showed, they are always taken from the test set, ie we trained the model on some train set, then, we getting the results or computing the metric on the results, it was always done on the test set.

Then, let's move forward to the MAE implementation, the goal of this project.

The MAE version of an autoencoder is really similar to the VAE one, they just change the loss function to a better performing one. The added two terms to the VAE loss, one encouraging the diversity of the learned model, the full explanation of why is not given here but in the article they start from a measure of diversity, and introduce those two losses that are in practice a better approximation of the diversity (expecially, they have less variance). The two added losses are then:

$$L_{diverse} = E_{X_1,X_2\sim P(X)}[\sum_{k=1}^{K} log(1 + exp(-KL(q_\phi(Z_k|X_1)||q_\phi(Z_k|X_2))))] \tag{6}$$

Where here ϕ is still the distribution of the encoder's parameters. This encourages the diversity of the parameters, since $L_{diverse} \geq 0$ and Ł$_{diverse} \to 0 \Leftrightarrow KL(q_\phi(Z_k|X_1)||q_\phi(Z_k|X_2)) \to \infty$, and if this happen the two distributins are indeed really different depending on the input datapoint. Here $Z = (Z_1, ..., Z_K)$ and hence Z_k is only one element in the latent variable, not taking this into account make the model behaves really badly.

The other loss term measures the smoothness of the latent distribution, since you ideally want them uniformly distributed over the latent space, for ease of use of the model, and fine grained generation. It is given as:

$$L_{smooth} = STD_{X_1,X_2\sim P(X)}[KL(q_\phi(Z_k|X_1)||q_\phi(Z_k|X_2))] \tag{7}$$

And here if the standard deviation is small, then the datapoints are more uniformly distributed (this comes from the fact that std is not a linear function, and can be seen in fig1 of MAE article).

We now have the following loss to minimize:

$$L_{MAE} = L_{ELBO} + \eta L_{diverse} + \gamma L_{smooth} \tag{8}$$

But how are $L_{diverse}$ *and* L_{smooth} computed in practice? Just by using a monte carlo estimate over the batch, ie using the following formulas, for example for $L_{diverse}$:

$$L_{diverse} = \frac{1}{M} \sum_{x_1 \neq x_2} \sum_{k=1}^{K} log(1 + exp(-KL(q_\phi(Z_k|X_1)||q_\phi(Z_k|X_2)))) \tag{9}$$

Where M is the batch size.

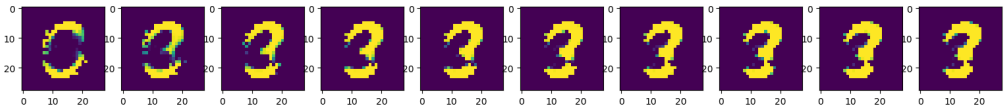


Figure 3: Our implementation, lower amount of neurons displayed

(a) (b)

Figure 4: Left: MAE reconstruction, right: original MNIST image

And the KL divergence term is derived from the following formula, assuming $q_\phi(Z_k|X_1)$ and $q_\phi(Z_k|X_2)$ follows normal distribution with there respective mu and σ[1] :

$$D_{KL}(N_0||N_1) = \frac{1}{2}(tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T\Sigma_1^{-1}(\mu_1 - \mu_0) - k + ln(\frac{\det \Sigma_1}{\det \Sigma_0})) \quad (10)$$

Which can be simplified in our case since we take the batch elements independently and hence are in one dimension, and wich can be implemented in python.

By using the same structure for our neural network as the one previously described for VAE, and by using $\eta = 1$ and $\gamma = 0.01$ we got the following samples, for example exploring the latent space in one direction:

Or by testing the reconstruction capabilities of our model:

Finally, we also implemented some convolution based models, for example for the CIFAR-10 dataset, but they were really long to train, and so we decided to stick with the MNIST dataset on simple networks.

2.2 Evaluating our experiments

Let's now see how we can quantitatively measure how well MAE performs against the VAE model. Our first attempt to do this was to plot the loss in both cases, which was indeed smaller in the MAE case, but sincec both of the losses were negative for some reason, we chose to explore some other direction. We decided to try to use the K-means algorithm on the latent space, since the MAE results seemed to be very good in comparison to other models. We implemented the k-means algorithm, but no matter what we tried, we did not get more than 0.2 for the accuracy accuracy on the latent variables. Unfortunatelly we didi not had time to investigate further on this, but it would be some nice future work on this project.

[1] hrefhttps://en.wikipedia.org/wiki/KullbackLeibler_divergence#cite_note-12Wikipedia