Machine Learning, Advanced Course Project Work

VAE with a VampPrior

Kenza Bouzid
Tobias Höppe
Arthur Tondereau
Felix Köhler

Abstract

Modern deep-generative models combine artificial neural networks with probabilistic aspects to allow for rich generation or reconstruction of various kinds of data. The Variational Auto-Encoder (VAE) is one such deep-generative model which is widely discussed since the seminal paper of [3]. The used prior for the latent space is responsible for regularizing the extent of this hidden representation. Depending on the dataset a simple Gaussian prior, as used in the original paper, might lead to over-regularization or over-fitting. Recent advances in this field therefore fostered the idea of using different prior distributions. In [5] a flexible prior called the VampPrior (Variational Mixture of Posteriors prior) was introduced which can also be used in the hierarchical fashion with multiple stochastic latent layers. In a re-implementation of their work we try to confirm on the reported improved results for multiple artificial datasets. We were able to reproduce similar improvements in the marginal log likelihood for the VampPrior over the standard Gaussian prior. The visual results matched the ones reported in the original paper and we can also report similarly shaped trained pseudo-inputs. However, we cannot report a substantial improvement with using the hierarchical prior as we observed an opposite performance on the marginal log likelihood evaluation.

I. Introduction

Most Machine Learning techniques can be unified under a probabilistic point of view. For instance, least squares regression results from the Maximum Likelihood Estimate (MLE) of noise modelled by a normal distribution. Often, the probabilistic point of view has the advantage of higher explainability and additional uncertainty quantification.

Research in Machine Learning recently fostered the combination of procedural artificial neural networks with variational Bayesian approaches giving rise to so-called deep-generative models as an unsupervised learning task. These models can learn structure in the data to then later produce (i.e. generate) new unseen data. One way of training generative models consists of framing the problem as a supervised learning problem where the labels are the training data point themselves. *Generative Adversarial Networks (GANs)* [4] are an approach to generative modeling that trains two sub-models (the generator and the discriminator) against each other as if they were playing a zero-sum game until the discriminator model is fooled about the veracity of the images generated by the generator. In case of faces, for instance, deep generative models would be able to create new faces or morph existing ones. This has many, also ethical, implications which are right now discussed under the term of 'deep fakes'.

Another deep-generative model is the Variational Auto-Encoder (VAE) [3]. This model extends the procedural form of an Auto-Encoder consisting of an encoding and decoding network with mirrored architecture. It does so by making the input space X as well as the latent space Z probabilistic and by putting a prior on the latent space. By this we can regularize the extent of the latent space and improve the generative performance of the model.

Recently, an extension to this framework was proposed by an augmented prior called the VampPrior that can also be used in a hierarchical fashion [5]. The authors reported an improved performance in almost all scenarios with popular artificial datasets. In this work, we attempt to re-implement the essential parts of their work and try to confirm on their results.

We structure this article as follows. In section two an introduction and explanation to the main concepts of the original VAE work [3] and its VampPrior extension [5] is given. We follow with a description of the network architecture and hints on the implementation which we chose to conduct using *Tensorflow* and *Tensorflow Probability*. Section 4 explains the subset of experiments we selected to validate and will present the results. We finish the article with a discussion on our findings and evaluate the original work on the VAE and its VampPrior extension.

II. Background

When talking about generative models, we are generally interested in finding representations of the latent variables Z of given data such that we can maximize

$$p(X) = \int_{\mathcal{Z}} p(Z)p(X|Z)dZ. \tag{1}$$

But to solve this we have to define the latent space $\mathcal{Z}$ from which Z is drawn and we have to deal with the integral, which is often intractable. Variational Auto-encoders [3] cope with both of these problems.

A. Variational Auto-Encoders

We cannot compute the marginal likelihood (1) since the encoding posterior

$$p(Z|X) = \frac{p(X|Z)p(Z)}{p(X)} \tag{2}$$

is intractable. We therefore use a recognition model $q_\phi(Z|X)$ to approximate the posterior, which is then tractable. When using $Z \sim q_\phi(Z|X)$ we can compute $\mathbb{E}_{Z\sim q_\phi(Z|X)}[p_\theta(X|Z)]$, where $p_\theta(X|Z)$ is the parametrization of the generative model. In order to use this to optimize (1), we use the relationship between $\mathbb{E}_{Z\sim q_\phi(Z|X)}[p_\theta(X|Z)]$ and $p_\theta(X)$ by rewriting the KL divergence between the true encoder $p(Z|X)$ and $q_\phi(Z|X)$ using Bayes rule.

$$\begin{aligned} &D_{KL}[q_\phi(Z|X)||p(Z|X)] && (3)\\ &\quad= \mathbb{E}_{Z\sim q_\phi(Z|X)}[\log q_\phi(Z|X) - \log p(Z|X)] && (4)\\ &\quad= \mathbb{E}_{Z\sim q_\phi(Z|X)}[\log q_\phi(Z|X) - \log p_\theta(X|Z) && (5)\\ &\quad- \log p_\theta(Z)] + \log p_\theta(X) && (6)\end{aligned}$$

which can be rewritten as

$$\begin{aligned} &\log p_\theta(X) - D_{KL}[q_\phi(Z|X)||p(Z|X)] = && (7)\\ &\quad= \mathbb{E}_{Z\sim q_\phi(Z|X)}[\log p_\theta(X|Z)] && (8)\\ &\quad- D_{KL}[q_\phi(Z|X)||p_\theta(Z)] && (9)\end{aligned}$$

We can further conclude

$$\begin{aligned} &\log p_\theta(X) \geq \mathbb{E}_{Z\sim q_\phi(Z|X)}[\log p_\theta(X|Z)] && (10)\\ &\quad- D_{KL}[q_\phi(Z|X)||p_\theta(Z)] && (11)\\ &\Rightarrow \log p_\theta(X) \geq \mathcal{L}(\theta, \phi, X) && (12)\end{aligned}$$

where $\mathcal{L}(\theta, \phi, X)$ is the variational lower bound (or ELBO for Evidence Lower BOund). This variational lower bound consists of the reconstruction error $\mathbb{E}_{Z\sim q_\phi(Z|X)}[\log p_\theta(X|Z)]$ and the KL divergence of the approximate posterior and the prior, which acts as a regularizer. The encoder $q_\phi(Z|X)$ and the decoder $p_\theta(X|Z)$ are parameterized as Neural Networks and as already mentioned above, we are able to optimize the parameters (θ, ϕ) jointly.

When optimizing, the KL-divergence is usually tractable, but to compute $\mathbb{E}_{z\sim q_\phi(Z|X)}[\log p_\theta(X|Z)]$ we need to sample $z_i \sim q_\phi(Z|X)$ and approximate the loss of a datapoint by

$$\begin{aligned} &\hat{\mathcal{L}}(\theta, \phi, x) = \frac{1}{K}\sum_{k=1}^{K} \log p_\theta(x|z_k) && (13)\\ &\quad- D_{KL}[q_\phi(Z|x)||p_\theta(Z)] && (14)\end{aligned}$$

However, doing so introduces a problem during back-propagation, since stochastic gradient descent via back-propagation cannot handle stochastic layers within the Neural Network. To solve this issue, [3] introduced the reparametrization trick in this context. We assume that our prior has the form of a centered isotropic Gaussian $p_\theta(Z) = \mathcal{N}(Z|0, \mathbb{I})$ (later we will introduce other priors). Instead of sampling Z directly from $q_\phi(Z|X)$, the sampling is moved to an input layer. Given the parameters $\mu(X, \phi)$, $\sigma(X, \phi)$ and a sample $\epsilon \sim \mathcal{N}(0, \mathbb{I})$ we can set $Z \sim \mu(X, \phi) + \sigma(X, \phi)^{\frac{1}{2}} * \epsilon$. Thanks to this, the gradient can pass through the entire Network, and we are able to train the parameters for the encoder and decoder jointly.

B. VampPrior

As discussed above, the prior does work as a regularizer and can therefore have a strong influence on the performance of the model. For example, by using a simple prior as $p_\theta(Z) = \mathcal{N}(Z|0, \mathbb{I})$ we tend to over-regularize. In this section, we introduce a prior, which can help finding a good balance between over-regularizing and over-fitting.

To introduce an expression of the Variational lower bound for the entire data set, we can rewrite $\hat{\mathcal{L}}(\theta, \phi, x)$ in the expectation of the empirical distribution $\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N} \delta(x - x_i)$[1]. By doing this, we get

$$\mathbb{E}_{x\sim\hat{p}(x)}[\mathcal{L}(\theta, \phi, x)] \tag{15}$$
$$= \mathbb{E}_{x\sim\hat{p}(x)}[\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)]] \tag{16}$$
$$- \mathbb{E}_{x\sim\hat{p}(x)}[\mathbb{H}[\log q_\phi(z|x)] \tag{17}$$
$$- \mathbb{E}_{z\sim q(z)}[-\log p_\theta(z)] \tag{18}$$

where $q(z) = \frac{1}{N}\sum_{i=1}^{N} q_\phi(Z|x_i)$, which can be derived from

$$\mathbb{E}_{x\sim\hat{p}(x)}[\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(z)]] \tag{19}$$
$$= \int_{\mathcal{Z}} \frac{1}{N}\sum_{i=1}^{N} q_\phi(Z|x_i) \log p_\theta(Z) dZ \tag{20}$$
$$= \mathbb{E}_{Z\sim q(Z)}[\log p_\theta(Z)] \tag{21}$$

The equation above shows, that our loss consists of three components. The reconstruction error of the input (16), the entropy $\mathbb{H}$ of our estimated posterior (17) (here we can see, that the loss pushes the posterior to have a large variance) and the expectation of the prior under the aggregated posterior (18). If we want to maximize this expression above according to our prior we can use a Lagrange multiplier β and get

$$p_\theta(Z) = \underset{p_\theta(Z)}{\text{argmax}}\Big(- \mathbb{E}_{Z\sim q_\theta(Z)}[-\log p_\theta(Z)] \tag{22}$$
$$+ \beta\left(\int p_\theta(Z)\,\mathrm{d}Z - 1\right)\Big) \tag{23}$$

This can be solved analytically and we get the following optimal prior

$$p_\theta(Z) = \frac{1}{N}\sum_{i=1}^{N} q_\phi(Z|x_i). \tag{24}$$

However, when using this prior the model tends to over-fit and is therefore not optimal in a practical sense. To find a good trade-off between over-regularization and over-fitting a mixture of variational posteriors with pseudo-inputs is proposed in [5].

$$p(Z) = \frac{1}{K}\sum_{k=1}^{K} q_\phi(Z|u_k) \tag{25}$$

The two main advantages of this prior are its multi-modality, which prevents it from over-regularizing and on the other hand the use of only K pseudo-inputs, which avoids over-fitting. There are two ways for initializing and possibly training pseudo-inputs. We can initialize them randomly, meaning that at first they will consist only of noise, and then train them as part of the model (more on that in the next section). The pseudo-inputs can also be chosen as K random data points from the dataset, which will lead to a different, slightly more regularized prior.

C. Hierarchical Prior

Another extension to the VAE are multiple stochastic units. But when using these, the model often suffers from the inactive stochastic latent variable problem. The deeper stochastic layers might obtain less information from the real data, since the generative process and the variational process are opposite to each other. This can lead to an over-regularized layer which will have a noticeable number of inactive units.

To prevent this, a new two layered model (e.g two latent variables $z1, z2$ is proposed in [5]). The joint distribution of the encoder given x will be

$$q_{\phi,\psi}(Z_1, Z_2|X) = q_\psi(Z_1|Z_2, X)q_\phi(Z_2|X). \tag{26}$$

Whereas the decoder generates a picture according to the following distribution

$$p_\theta(X|Z_1, Z_2)p_\lambda(Z_1|Z_2)p(Z_2) \tag{27}$$

The distributions of Z_1 and Z_2 will be trained jointly. The prior of Z_2 can be any prior discussed in this paper. For the distribution of Z_1, which depends on Z_2, we chose a Gaussian normal with trainable parameters $\mu(\lambda, Z_2)$ and $\sigma(\lambda, Z_2)$

$$p_\lambda(Z_1|Z_2) = \mathcal{N}(Z_1|\mu(\lambda, Z_2), \sigma(\lambda, Z_2)) \tag{28}$$

[1]The empirical distribution is a common way to represent our given dataset $\mathcal{D} = \{x^{[1]}, \dots x^{[N]}\}$. Since we know that we observed $x^{[n]}$ (which is just a sample of a e.g. 28×28 multivariate space in case of *MNIST*), we can express this by a peak (concentrated probability density mass) using a dirac delta distribution $\delta(x)$. Then our full dataset is given as the mixture distribution of N dirac peaks with each of them being equally likely.

The posteriors are then trained given x and x, z_2, respectively.

$$q_\phi(Z_1|Z_2, X) = \mathcal{N}(Z_1|\mu(\phi, X, Z_2), \sigma(\phi, X, Z_2)) \quad (29)$$

$$q_\psi(Z_2|X) = \mathcal{N}(Z_2|\mu(\psi, X), \sigma(\psi, X)) \quad (30)$$

D. Test Marginal Log-Likelihood

A common Bayesian quantity for model comparison is the marginal log likelihood of our observed data X.

$$\log p(X) = \log \int_Z p(X|Z)p(Z)\,\mathrm{d}Z \quad (31)$$

This equals the logarithm of the expectation over $p(Z)$ of $p(X|Z)$. We can use Jensen's inequality to move the logarithm into the expectation.

$$\log p(X) = \log \mathbb{E}_{p(X)}\left[p(X|Z)\right] \quad (32)$$

$$\geq \mathbb{E}_{p(X)}\left[\log p(X|Z)\right] \quad (33)$$

The importance sampling trick applied to (31) results in the expectation over the probabilistic encoder

$$\log p(X) = \log \int_Z p(X|Z)p(Z)\frac{q(Z|X)}{q(Z|X)}\,\mathrm{d}Z \quad (34)$$

$$\geq \mathbb{E}_{q(Z|X)}\left[\log \frac{p(X|Z)p(Z)}{q(Z|X)}\right] \quad (35)$$

We can approximate the expectation operator by sampling from the probabilistic encoder. If we are then given a data point $x^{[n]}$ (e.g. an image), the marginal log likelihood becomes

$$\log p(X = x^{[n]}) \geq \frac{1}{L}\sum_{l=1}^{L}\Big[\log p(X = x^{[n]}|Z = z^{[l]}) + \log p(Z = z^{[l]}) - \log q(Z = z^{[l]}|X = x^{[n]})\Big] \quad (36)$$

with $z^{[l]} \propto q(Z|X = x^{[n]})$. Note that $\log p(Z) - \log p(Z|X = x^{[n]})$ is the negative Kullback-Leibler divergence between the encoder and the prior which in the case of a Gaussian prior can be computed analytically. In this case, no sampling of Z is required. Consequentially, $\log p(X =^{[n]}|Z)$ is the reconstruction error.

If we have a (test) dataset $\mathcal{D} = \{x^{[1]}, \ldots, x^{[N_{\text{(test)}}]}\}$, the average marginal log-likelihood is given as

$$\text{MLL} = \frac{1}{N_{\text{(test)}}}\sum_{n=1}^{N_{\text{(test)}}} \log p(X = x^{[n]}) \quad (37)$$

Note that for the implementation the logsumexp operator can be used to avoid Jensen's inequality. Assume Y is a random variable we want to take the log-expectation of

$$\log \mathbb{E}[f(Y)] = \log \mathbb{E}[\exp(\log(f(Y)))] \quad (38)$$

$$= \log\left(\frac{1}{L}\sum_{l=1}^{L}\exp\Big(\log\Big(f(y^{[l]})\Big)\Big)\right) \quad (39)$$

Define $\xi^{[l]} := \log f(y^{[l]})$, then

$$\log \mathbb{E}[f(Y)] = \text{logsumexp}(\{\xi^{[l]}\}_{l=1}^{L}) - \log(L) \quad (40)$$

A marginal log likelihood calculated using this method is not a lower bound (since Jensen's inequality is not used) but a true estimate.

III. Method

A. Gated Dense Layers

In the original papers [3], [5], the authors made use of gated dense layers instead of regular dense fully connected layers. Define a fully connected layer as

$$\Phi_{\text{dense}}(y) = \rho(\mathbf{W}y + b) \quad (41)$$

with a weight matrix $\mathbf{W}$, a bias vector b and a non-linear activation function $\rho(\cdot)$, which we selected to be the sigmoid function. Then the gated dense layer is described by two independent layers of the same size with weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ as well as bias vectors b_1 and b_2. Its result is calculated as the component wise multiplication ($\odot$) of the two layers' results with only one of them being activated

$$\Phi_{\text{gated\_dense}}(y) = (\rho(\mathbf{W}_1 y + b_1)) \odot (\mathbf{W}_2 y + b_2) \quad (42)$$

B. Integrating the pseudo-inputs

The pseudo-inputs are a key concept of the VampPrior VAE. They are of the same shape as the inputs of the networks (say images of 28x28 pixels in the case of the *MNIST* dataset). In order to include them in the model we input them to the encoder, the VampPrior then being the mixture of all the distributions it outputs. In case of trained pseudo-inputs (as opposed to pseudo-inputs taken from the data) they are created by training a single layer, that outputs images which can then be fed into the encoder.

Basically, it transforms a one-hot representation of the pseudo-inputs into the actual images. Examples of trained pseudo-inputs are shown in 8.

C. Network Architecture

The theory on Variational Auto-Encoders implies certain design decisions:

- We want to be able to decompose the model into an encoder and a decoder, so that we can separate the decoder to generate new images once the model is trained.
- The intermediate representation is in a latent space, described by a probabilistic layer. Its size is set to 40 for all datasets tested.

The output layer of the model produces not an image as in a normal Auto Encoder, but a juxtaposition of distributions. Since the input images of the chosen datasets are binary for each pixel we output a Bernoulli distribution. The output of the network can then be evaluated by taking a sample, the mean, or the mode of each distribution.

Below is an illustration of the model with the described features.

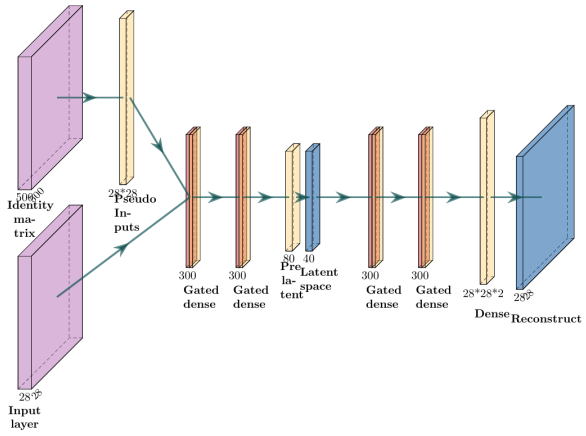


Fig. 1: Architecture of the VAE with pseudo-inputs. Purple denotes an input layer, yellow a dense non-activated layer, red a dense activated layer and blue a stochastic layer.

D. Hierarchical architecture

As stated in the previous section we implement a hierarchical VAE. The variational part (encoder) of this two-layered model returns two outputs corresponding to the two latent variables Z_1 and Z_2. First, we create the posterior $q_\phi(Z_2|x)$. After passing the input x through two gated dense layers and passing Z_2 through one respectively, we concatenate these outputs in order to join x to Z_2 and create the conditional posterior $q_\psi(Z_1|Z_2, x)$. The two latent variables Z_1 and Z_2 are then passed (after the reparameterization) trick into the generative part (decoder) which creates the prior $p_\lambda(Z_1|Z_2)$ by passing Z_2 through two gated dense layers. Finally the reconstructed images are produced by joining Z_1 and Z_2. The process of generating new images is described in section II-C.

The detailed architecture is presented below.

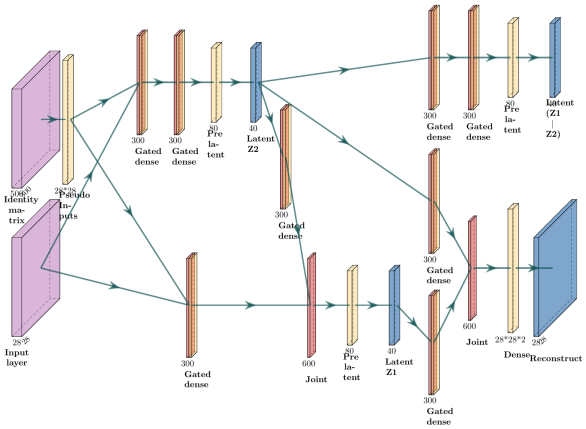


Fig. 2: Network architecture of the Hierarchical VAE (2 levels) with pseudo-inputs (colors are explained in Figure 2).

E. Implementation details

It is noteworthy to explain our choice of deep-learning framework to implement the Variational Auto-Encoder. We hereby used *TensorFlow* and *TensorFlow Probability*. The latter allows for probabilistic layers in a *Keras* sequential or functional model. This naturally translates the idea of the probabilistic encoder and decoder to be just networks that calculate the distributions' parameters.

TensorFlow Probability implements the reparameterization trick and allows for efficient batched querying and sampling from distributions.

IV. Results

In this section we present our results based on the subset of experiments we conducted.

A. Experiments conducted

As presented earlier, the very last layer of our neural network is always related to the reconstruction of our original data shape. When speaking in terms of images with only one color channel (a gray-scale channel) this can be either an Independent Bernoulli layer for black-and-white images or an Independent Normal layer for gray-scaled images. We followed the original paper and implemented both approaches. However, for brevity, we will resort to only use binary image datasets. We hereby reduced the experiments to the standard *MNIST* set of handwritten digits, the *Caltech 101 Silhouettes* set and the *Omniglot* set. For confirming on the VampPrior's claimed performance, we found it sufficient to only investigate on the non-convolutional architectures. This results in the selection of the vanilla VAE and the hierarchical VAE, both with the standard Gaussian prior, the VampPrior using training samples (Vamp Data) and the VampPrior with trained pseudo-inputs (Vamp Generated). The generated and reconstructed images in this reports only show the mean value outputs in the images which corresponds to the theta parameter of the Bernoulli distribution in the case of binary images. This is mainly to give a view as unbiased as possible on the results, as opposed as the use of samples.

B. Model Training and Hyper-Parameters

We follow the original paper's choice of the used Hyper-Parameters. Based on this, we select the *Adam* optimizer [2] emphasized with gradient clipping in order to avoid vanishing gradients. A slower learning rate of 1.0×10^{-4} has been used for the more complicated datasets (*Omniglot* and *Caltech 101 Silhouette*) and a faster learning rate of 5.0×10^{-4} for the simpler *MNIST* set. We trained with a batch-size of 100 for up to 1000 epochs. An early-stopping with patience of 50 epochs and minimum absolute change threshold of 1.0×10^{-3} was used to prevent over-fitting by monitoring the validation loss on the test split. The

number of pseudo-inputs in case of the VampPrior was 1000 for the *Omniglot* set and 500 for every other set. We used a default weight of 1.0 for the KL divergence for all the training. All the weights of the layers have been initialized according to *He Normal (He-et-al) initialisation* [1] by drawing them from a distribution with zero mean and a variance equals to:

$$Var(w_i) = \frac{2}{fan\_in} \tag{43}$$

where fan_in is the number of incoming neurons. This helps breaking the symmetry among different units and leads to better performance. Training was performed on different *Nvidia Tesla* GPUs using *Google Colab*.

C. Quantitative Results

As mentioned in the previous section our metric for Bayesian model comparison is the marginal log-likelihood. With this quantity we can assess the performance of image reconstruction and image generation. The expectation approximation was performed with a sample size of $L = 5000$ in all datasets but the *Omniglot* set where we used only 2000 samples. We report the evaluated metrics in Table I.

For the vanilla VAE model we obtain similar results to the original paper in case of the *MNIST* and *Caltech* dataset. Although those values differ by ≈ 2 a similar picture can be drawn in that the VampPrior improves our result significantly. Note, that we are comparing log likelihood values in terms of the natural logarithm. Hence, every four digits of difference corresponds to one order of magnitude in the decimal system (making it ten times more likely). We explain our slightly worse results by a potentially different weight initialization and by our choice of implementation using *TensorFlow Probability* instead of *PyTorch*.

In contrast to the two smaller datasets, the method performed substantially worse on the*Omniglot* than in the original paper's reporting with differences of over 1100 in log-space. We can not explain this particular results since its visual quality is not as bad as this value might indicate (see next subsection). The higher resolution of these images together with a more diverse set of symbols increased the training complexity, explaining why we had to reduce the learning rate to achieve convergent behavior.

In case of the HVAE performance, we can not report similar values to the original article. In case of the *MNIST* and the *Caltech* dataset, our results are on average 20 values in log-space worse. Consequently, they are also worse than the results from the vanilla VAE not favoring the use of hierarchical networks.

More insight on the distribution of the marginal log likelihood values for the MNIST dataset is given with the histograms of Figure 3 and 4. Although the mean of this distribution (which we report in the mentioned Table I) does not match the reported values of the original article, the distributions' shape do coincide. By this we can report that all of our models can at least explain the data.

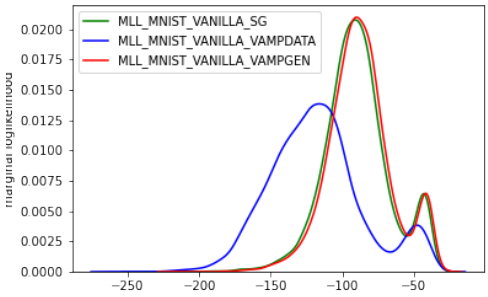


Fig. 3: Histogram Kernel-Density Estimate for the Marginal Log Likelihoods of the test split of the *MNIST* dataset on the vanilla VAE.

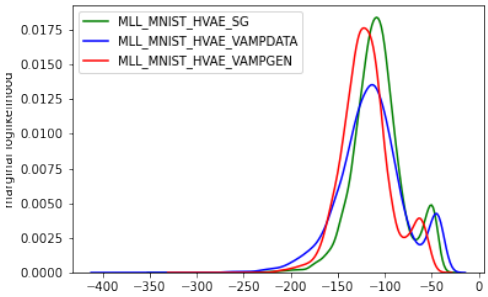


Fig. 4: Histogram Kernel-Density Estimate for the Marginal Log Likelihoods of the test split of the *MNIST* dataset on the hierarchical VAE.

TABLE I: Marginal log likelihood per model and dataset.

Architecture	VAE			HVAE		
Prior	Gaussian	Vamp Data	Vamp Generated	Gaussian	Vamp Data	Vamp Generated
MNIST	−89.59	−118.12	**−87.42**	−107.15	−116.49	**−104.30**
OMNIGLOT	**−1138.9**	−1145.8	−1153.11	**−1555.96**	−2166.31	−1637.72
CALTECH 101	−119.31	**−115.45**	−116.98	**−140.65**	−184.63	−146.07

D. Qualitative Results

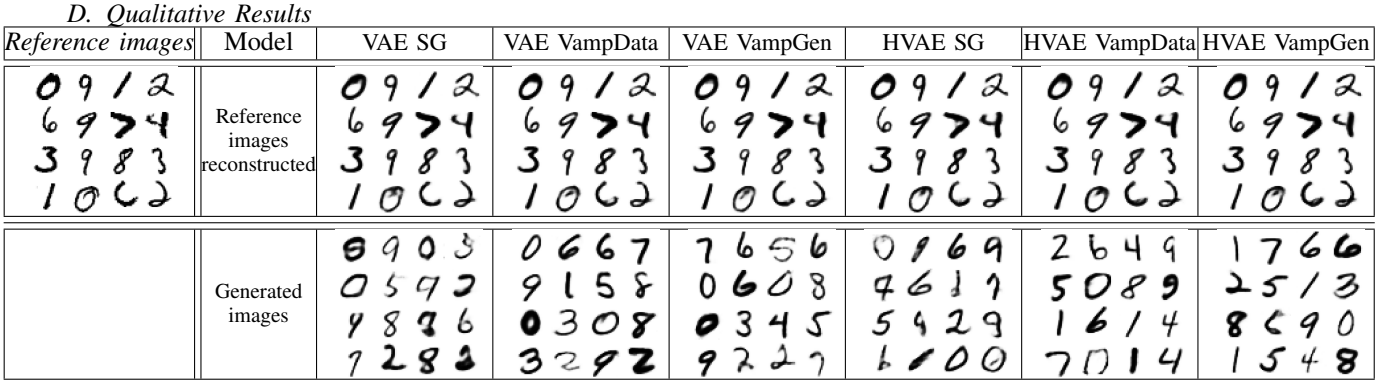


Fig. 5: Results for the *MNIST* dataset.

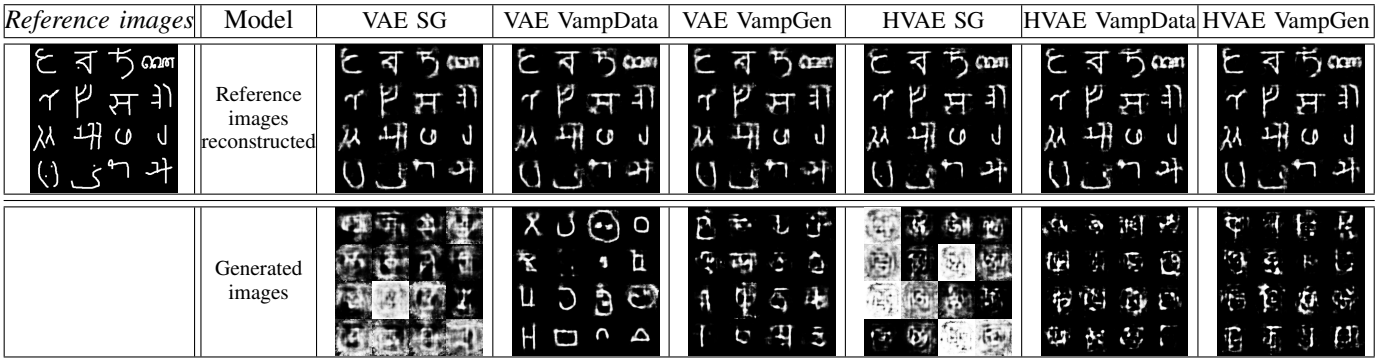


Fig. 6: Results for the *OMNIGLOT* dataset.

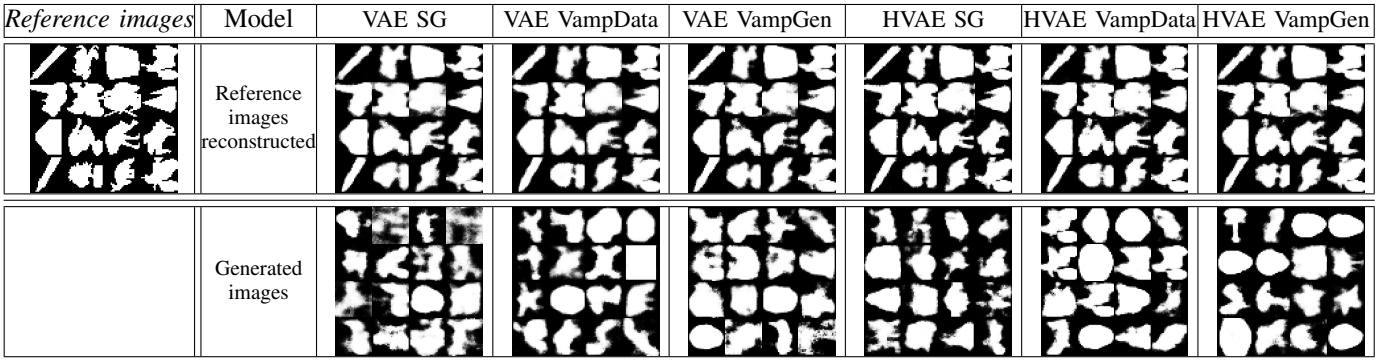


Fig. 7: Results for the *CALTECH* dataset.

Above we present various results from trained VAE models. For each model and dataset we present reconstructed images, as well as generated ones. Each model is denoted by its architecture (Vanilla / HVAE), and by the type of prior we use (Standard Gaussian, Vamp Data, Vamp Generated). Vamp Data refers to pseudo-inputs taken from the training samples and Vamp Generated to trained pseudo-inputs.

In the generated images we can see, that the over-regularizing effect of the single Gaussian prior does effect mainly the more complex data-sets, whereas we cannot see any significant difference on the *MNIST* dataset. However, the reconstruction of images does not seem to depend on the complexity of the dataset nor the prior or architecture used which shows that the training process is sufficient.

The *MNIST* dataset converges to results of high quality for all architectures and priors. The images are reconstructed with only little noise. The generated ones are difficult to distinguish from the original or reconstructed ones.

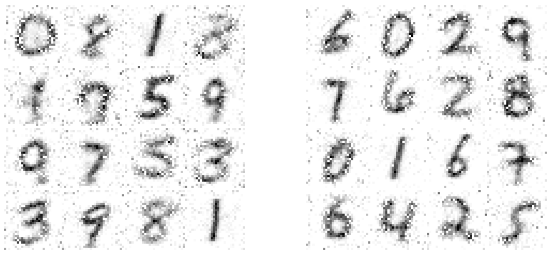
Observing the more complex *Caltech* dataset, we can see that changing the prior does clearly improve the quality of the generated images. The use of the Vamprior does decrease the amount of noise in the images. The use of the Hierachical architecture on the other hand does not seem to have an influence on the quality.

These observations do manifest when comparing the results of the *Omniglot* dataset. The generated images with a Network using the Standard Gaussian prior are very noisy and the shapes are difficult to recognise. The Vampriors on the other hand are able to generate images with less noise where shapes can clearly be identified. However, it seems that the images generated with a single Gaussian do show more complex objects whereas the Vampriors do generate more simple but clearer shapes.

Furthermore, in this dataset we can for the first time see a significant difference between the Vanilla and Hierarchical architecture. Contrary to our expectations, the Hierarchical Networks do seem to perform worse than the Vanilla ones. We cannot explain what exactly caused this. It is most likely related to an error in our implementation.

Regarding Vampdata and Vampgen models, it is hard to conclude on which is generating better images. For instance *OMNIGLOT* shows that using pseudo-inputs from training samples improved the quality of the generated images greatly in comparison to the trained pseudo-inputs. For this specific dataset, one can argue that non-trained pseudo-inputs can perform better as the dataset itself is hard to learn and so are the trained pseudo-inputs. On the other hand, it seems like the learned pseudo-inputs attempt to create more complex shapes that approximate better the original data.

Another interesting feature of the models that we can observe are the trained pseudo-inputs. We extract a subset of them for the *MNIST* dataset for both the VAE and HVAE architectures and plot them after.



Trained pseudo-inputs with VAE architecture

Trained pseudo-inputs with HVAE architecture

Fig. 8: Trained pseudo-inputs on MNIST dataset.

When looking at the trained Pseudo-inputs of Figure 8 we can confirm the result of [3]. In our networks, too, the pseudo-inputs seem to be trained as representatives of the original data. To some extent this pushes further the argument for using data points as pseudo-inputs, since the pseudo-inputs seem to converge to the shape of data points in that case. This reaches the limit of the explainability of this model: perhaps the noise superimposed to the digits is a side effect of the random initialization, but it could also contain key information for the resulting prior. Without that knowledge it is not safe to assume that the process of training pseudo-inputs converges to the shape of data points.

V. Discussion

In this report we attempted to re-implement the most important aspects of the article 'VAE with a VampPrior' [5]. The results we created show that a multi-model prior which regularizes less and can improve performance on more complex datasets. However, we couldn't report an advantage in performance when using the Hierachical structure for the Neural network. This might be due to errors in the implementation of the model itself or in the computation of the marginal log-likelihood. Since we were not able to solve this issue, we cannot support nor reject their claims about the Hierachical Variational Autoencoder. Furthermore, we can see in the qualitative and quantitative results, that the Networks do produce significantly poorer results for the *OMNIGLOT* dataset compared to the others. One explanation for this might be the low complexity of the Network itself, since we did not change the number of units per layer nor the number of layers. All datasets are trained on the same architecture. Therefore, in order to improve the results on the *OMNIGLOT* dataset one can try to build a more complex model by adding units and layers to the Neural Network.

On the other hand, even when not performing as well on the *OMNIGLOT* as on the other data-sets, the usage of the Vampriors does increase the performance significantly compared to the single Gaussian prior. Considering that the VampPrior as extension to the VAE does make the model more complex in implementing and training, we would conclude that the usage of the VampPrior is only justified when modeling complex data and the Gaussian as a prior leads to over-regularization.

We would like to point out that most of the motivations and quantitative justifications of these methods are very well illustrated in the original paper which considerably simplified the understanding. The only negative note we can report is an unclear description of the architecture of some Networks and their implementation which made it necessary to look at the original code implementation in order to reproduce the results.

As mentioned, we could not reproduce all of them, but for most of the datasets and models we were able to get values and images which seem to support their conclusions. As most of their claims are supported sufficiently on a quantitative but also qualitative perspective the quality of their work does seem more than acceptable.

References

[1] Leonid Datta. A survey on activation functions and their relation with xavier and he normal initialization, 2020.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[4] Pegah Salehi, Abdolah Chalechale, and Maryam Taghizadeh. Generative adversarial networks (gans): An overview of theoretical model, evaluation metrics, and recent developments, 2020.

[5] Jakub Tomczak and Max Welling. Vae with a vampprior. In *International Conference on Artificial Intelligence and Statistics*, pages 1214–1223. PMLR, 2018.