Large project 1.6 - Importance Weighted Autoencoders

Project group 8:
Alexander Gutell, Filip Derksen, Madeleine Lindström, Ludvig Skare

January 2024

Abstract

The Variational Autoencoder (VAE) is a generative machine learning model aimed to approximate a posterior inference. The autoencoder consists of a generative process and a recognition model, where the recognition model is trained using a variational lower bound. In 2015, Burda et al. published an improved version of the VAE called Importance Weighted Autoencoders (IWAE), where a strictly tighter lower bound was used. The goal of this paper was to reproduce the findings of Burda et al. This was done in Google Colab, utilizing their GPU, and using Pytorch to train model architectures on the benchmark datasets MNIST and Omniglot. Minor adjustments were made to the models to lower runtime, or fix issues during training. Models were evaluated by plotting training losses, computing negative log-likelihoods (NLL) on the test data and reconstructing and generating data. With these adjustments, we received consistently slightly higher NLL values than the corresponding values in the original paper. The NLL values did however accurately represent the advantages of IWAE compared to VAE, as shown by Burda et al. To conclude, our reimplementation showed similar results to the results of Burda et al.

1 Introduction

The Variational Autoencoder (VAE) model was first introduced by Kingma & Welling in 2013 [1]. The model was later expanded by Rezende et al. to include multiple latent layers [2]. The Importance Weighted Autoencoders (IWAE) model was introduced by Burda et al. in 2016 [3]. In the following section, the VAE and IWAE models for multiple latent layers are explained, as well as the Xavier initialization of model parameters.

1.1 VAE

In the VAE and IWAE models, the aim is to create a recognition model to approximate a posterior inference [3]. This is done by a generative process $p(\mathbf{x}|\theta)$, followed by a recognition model $q(\mathbf{h}|\mathbf{x})$. The generative process can be factorized as

$$p(\mathbf{x}|\theta) = \sum_{\mathbf{h}^1,...,\mathbf{h}^L} p(\mathbf{h}^L|\theta)p(\mathbf{h}^{L-1}|\mathbf{h}^L,\theta)...p(\mathbf{x}|\mathbf{h}^1,\theta)$$

Here, θ is a vector of parameters of the model, $\mathbf{h} = \{\mathbf{h}^1, \mathbf{h}^2, ..., \mathbf{h}^L\}$ the latent variables for the stochastic latent layers $l = \{1, 2, ..., L\}$ and $\mathbf{h}^0 = \mathbf{x}$. We assume that the sampling and probability evaluation of $p(\mathbf{h}^l|\mathbf{h}^{l+1})$ is tractable for each layer l. In the generative model, we also have deterministic layers; these are however not shown here.

Moreover, we assume that we can factorize our recognition model $q(\mathbf{h}|\mathbf{x})$ analogously:

$$q(\mathbf{h}|\mathbf{x}) = q(\mathbf{h}^1|\mathbf{x})q(\mathbf{h}^2|\mathbf{h}^1)...q(\mathbf{h}^L|\mathbf{h}^{L-1})$$

and that sampling and probability evaluation of $q(\mathbf{h}^{l+1}|\mathbf{h}^l)$ is tractable for each layer l.

In the paper by Burda et al., the recognition distribution $q(\mathbf{h}^l|\mathbf{h}^{l-1},\theta)$ is reparameterized as a Gaussian distribution for each stochastic layer l, with mean $\mu = \mu(\mathbf{h}^{l-1},\theta)$ and covariance $\Sigma = \Sigma(\mathbf{h}^{l-1},\theta)$ [3]. The reparameterized latent variable $\mathbf{h}^l$ is given by

$$\mathbf{h}^l(\epsilon^l, \mathbf{h}^{l-1}, \theta) = \Sigma(\mathbf{h}^{l-1},\theta)^{1/2}\epsilon^{\mathbf{l}} + \mu(\mathbf{h}^{l-1},\theta),$$

where $\epsilon^l \sim N(\mathbf{0}, \mathbf{I})$.

For VAE, the recognition model is trained by maximizing the variational lower bound on the log-likelihood, defined as [3]

$$\begin{aligned} L_k^{VAE}(\mathbf{x}) &= \mathbb{E}_{q(\mathbf{h}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{h})}{q(\mathbf{h}|\mathbf{x})}\right] \\ &= \mathbb{E}_{\mathbf{h}_1,\mathbf{h}_2,...,\mathbf{h}_k\sim q(\mathbf{h}|\mathbf{x})}\left[\frac{1}{k}\sum_{i=1}^{k}\log\frac{p(\mathbf{x},\mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})}\right]. \end{aligned} \tag{1}$$

1.2 IWAE

For the Importance Weighted Autoencoder (IWAE), introduced by Burda et al. [3], the same architecture as for VAE is used. I.e. the same assumptions about $p(\mathbf{x}|\theta)$ and $q(\mathbf{h}|\mathbf{x})$ are made and the same reparameterization of $\mathbf{h}^l$ is used for each layer l, as described

above. The difference is that the IWAE is trained by maximizing a strictly tighter lower bound than for VAE, defined as

$$L_k^{IWAE}(\mathbf{x}) = \mathbb{E}_{\mathbf{h}_1,\mathbf{h}_2,\ldots,\mathbf{h}_k \sim q(\mathbf{h}|\mathbf{x})} \left[\log \frac{1}{k} \sum_{i=1}^{k} \frac{p(\mathbf{x}, \mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})} \right], \tag{2}$$

where $w_i = \frac{p(\mathbf{x},\mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})}$ are the unnormalized importance weights for the joint distribution.

1.3 Xavier initialization

The deterministic layers of the different models were initialized using the Xavier initialization. This initializes all the biases to be 0 and the weights W_{ij} for each node at each layer is uniformly sampled with respect to the size of the current and previous layer. W_{ij} is given by

$$W_{ij} \sim U[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}]$$

where U[-a,a] is the uniform distribution and n_j is the size (number of nodes) of the jth layer. This initialization is quite common is VAE architectures [4].

1.4 Scope and Objective

The objective of this paper is to attempt to recreate the negative log-likelihood (NLL) values for the different model architectures and different k-values. It does not include the analysing of active units that Burda et Al. did in their paper, which is a measure of how active and expressive the latent spaces were. Moreover, it also includes generated samples from the VAE and IWAE models for each dataset.

2 Method

To replicate the paper, we used Google Colab and utilized their GPU to run our algorithms. In the following section, the architectures, differences between the implementations of VAE and IWAE, training and evaluation of the models are described.

2.1 Architectures

Two models with different neural architectures were trained during the experiment. Pytorch was utilized to build the architecture along with the activation functions. Henceforth, observations are referred to as $\mathbf{x}$, and the first and second stochastic layers as $\mathbf{h}^1$ and $\mathbf{h}^2$, respectively.

1. The first architecture contained $\mathbf{h}^1$ with 50 units and two deterministic, 200-unit layers in between it and $\mathbf{x}$ as well as between $\mathbf{x}$ and the output of the decoder.

2. The second architecture contained $\mathbf{h}^1$ and $\mathbf{h}^2$ with 100 and 50 units, respectively. In between $\mathbf{x}$ and $\mathbf{h}^1$, there were two deterministic layers with 200 units each. Between $\mathbf{h}^1$ and $\mathbf{h}^2$, there were two deterministic layers with 100 units each.

Every deterministic hidden unit used the tanh nonlinearity. The stochastic layers were modeled to follow independent and identically distributed Gaussian distributions, except for the output layer, which was modeled with Bernoulli distributions. The output in the final layer was activated using the sigmoid function.

2.2 VAE vs. IWAE

The models were trained to maximize $L_k^{VAE}(\mathbf{x})$ and $L_k^{IWAE}(\mathbf{x})$, defined in Eq. 1 and 2, for VAE and IWAE respectively. This was done by minimizing the negative lower bounds, or losses; $-L_k^{VAE}(\mathbf{x})$ and $-L_k^{IWAE}(\mathbf{x})$.

2.3 Training

When training the different models for VAE and IWAE we, utilized the two architectures (model types) presented in Section 2.1. All models were initialized with the Xavier heuristic (Section 1.3) and trained with an Adam optimizer with the parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-4}$ and minibatches of 100. We utilized a scheduler for our learning rate defined as $0.001 \cdot 10^{-i/5}$ where i iterates from 0 to 5. The models were trained with 3^i epochs for each learning rate resulting in a total number of $\sum_{i=0}^{5} 3^i = 364$ epochs. This scheduler results in an exponentially increasing number of epochs for each more refined learning rate. Training was done for each model in which we varied the k-value (k = 1, 5, 50), the dataset type (MNIST or Omniglot) and loss function ($-L^{VAE}(\mathbf{x})$ or $-L_k^{IWAE}(\mathbf{x})$). Models for MNIST used 60 000 training data points and 10 000 test data points and models for Omniglot used 24 345 training data points and 8070 test data points. All data points were binarized 28x28 images.

2.4 Evaluation

Evaluation was done using an approximation of the negative log likelihood (NLL) on the test set. The approximation was done by computing $L_{5000}(\mathbf{x})$ for each network.

3 Results

In the following section, some results of the models are shown. The results include reconstructed and generated images, training losses and NLL.

3.1 Reconstructed and Generated Images

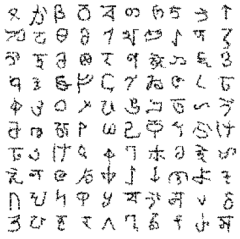
In Figure 1, original images and their reconstructions are shown, as well as generated samples from Model 1 and k=5.

Original MNIST datapoint batch

Reconstructed MNIST batch from VAE model

Reconstructed MNIST batch from IWAE model

Generated images from z-space, trained on MNIST



Original Omniglot datapoint batch

Reconstructed Omniglot batch from VAE model

Reconstructed Omniglot batch from IWAE model

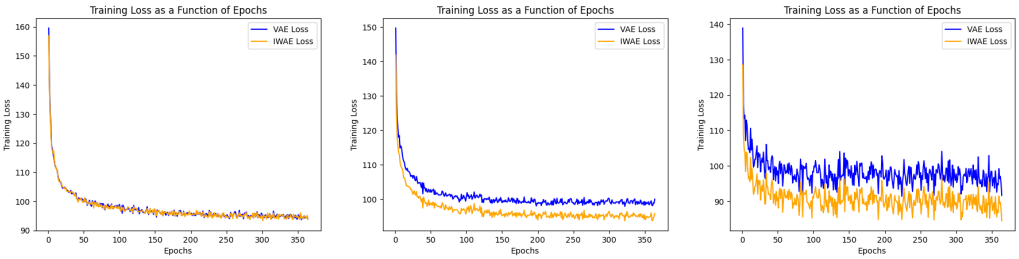
Generated images from z space, trained on Omniglot

Figure 1: Original MNIST/Omniglot datapoints, reconstructed datapoints and generated samples from Model 1 with 1 stochastic layer with k = 5.

3.2 Training Losses

3.2.1 Model 1 - MNIST

In Figure 2, training losses for Model 1, using MNIST, are shown.

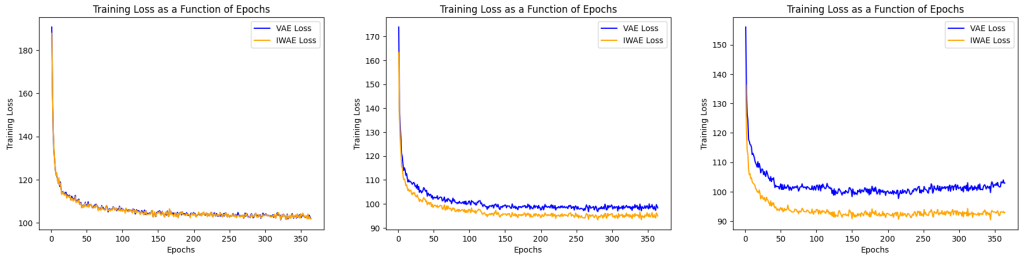


k=1 k=5 k=50

Figure 2: Training losses for VAE and IWAE training for model 1 with 1 stochastic layer on MNIST for $k = 1, 5, 50$.

3.2.2 Model 2 - MNIST

In Figure 3, training losses for Model 1, using MNIST, are shown.



k=1 k=5 k=50

Figure 3: Training losses for VAE and IWAE training for model 2 with 2 stochastic layers on MNIST for $k = 1, 5, 50$.

3.3 NLL

Negative log likelihoods (NLL) on test sets are shown in Table 1.

		MNIST		OMNIGLOT	
		VAE	IWAE	VAE	IWAE
# stoch. layers	k	NLL	NLL	NLL	NLL
1	1	93,17	93,16	116,89	116,94
	5	97,33	87,02	116,37	112,83
	50	96,90	86,11	116,60	109,66
2	1	105,78	105,74	129,65	122,70
	5	102,42	93,90	127,80	118,18
	50	106,77	92,75	122,41	112,78

Table 1: Negative log-likelihood losses for the different models, datasets and k-values.

4 Discussion

According to Table 1, the results obtained seem fairly consistent with those of the original report. For Model 1, there is a discernible trend where the NLL for IWAE decreases as the value of k increases across both datasets examined. Nevertheless, the NLL values for VAE demonstrate significant variability, oscillating between 93.16 and 97.33 for the MNIST dataset, which suggests a high degree of variance in the test results. This variability is evident in the training plots as well, raising concerns about the reliability of the results due to potential random or systemic fluctuations. Despite this, the recorded values for Model 1 are comparably close to those anticipated, suggesting an acceptable level of accuracy.

For model 2, the results appear to deviate somewhat from those reported by Burda et al. While the original report produces values approximately in the range of 108 to 103, model 2 in this report produces values roughly between 130 and 112. This disparity is likely caused by the second model not being sufficiently converged for the parameters selected for training. In the original report, where $i = 7$, the training would have completed 3280 passes, whereas this report only completed 364 passes, using $i = 5$, with a five times higher batch size. Therefore, since our training only completed roughly one ninth of the original training, it is quite understandable that the results are higher. Notice moreover that model 1 was able to convergence despite these differences because it has significantly less trainable weights. However, model 2 is also likely slightly wrong considering the different results.

Moving our attention to figure 1 we see how our models performed when they regenerated the training data. It can clearly be observed that model 1 performed well in their reconstructions for both MNIST and Omniglot, which is evidence of a well performing model. A handful of the reconstructions are incorrect but that is to be expected from such a simple model architecture. We also find images of generated images from model 1 architectures trained on MNIST and Omniglot data in figure 1. These are noticeably worse where many samples are similar to but not equal to a definitive digit. A couple samples have zero resemblance to any known digit in the case of MNIST.

The training methodology outlined in this report largely mirrors the approach taken by Burda et al., with two notable exceptions pertaining to parameter selection. Specifically, minibatches of size 100 were employed in conjunction with the Adam optimizer, and the summation index i was fixed at 5. These differences from the original configuration, which utilized minibatches of size 20 and an index of i = 7, were introduced mainly for computational feasibility where models could be trained in under 4 hours instead of 30 hours. Although these modifications raise the NLL (negative log-likelihood) values in Table 1 by restructuring the training data along with training for significantly fewer epochs, they seemed effective in evaluating the models' overall test performance.

Interestingly, the Tanh activation function sometimes produced poor test outcomes and/or vanishing gradients for our larger model 2 which had a deeper neural network architecture with two stochastic layers. This effect was particularly noticeable when $k = 1, 5$, leading to test scores around 200. To mitigate this issue without introducing normalization layers—which would alter the network architectures—the LeakyReLU function, a default in PyTorch, was employed instead. However, this substitution means that not all results were obtained using the same activation function, potentially affecting

the consistency of the results.

A more recent example of how the findings of Burda et al. is used is in the comparative study of Bond-Taylor et al (2021), in which deep generative models, including VAEs, are studied. Using IWAEs, the ELBO could be improved, which in conjunction with various other improvements resulted in an improved posterior approximation [5].

Future studies based on this paper would aim to reproduce the active units results of the original Burda et Al. paper with the focus of analysing how IWAE increases the expressiveness of the z-space. We would furthermore have liked to focus more on finetuning model 2 with two stochastic layers in order to make it perform better than model 1. In addition to this it would've been interesting to generate samples from both stochastic layers for model 2 and compare the resulting images to those of model 1.

To conclude, our reimplementation of the models by Burda et al. (with minor adjustments), showed similar results.

References

[1] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[2] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic backpropagation and approximate inference in deep generative models". In: *International conference on machine learning*. PMLR. 2014, pp. 1278–1286.

[3] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. *Importance Weighted Autoencoders*. 2016. arXiv: `1509.00519 [cs.LG]`.

[4] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: `https://proceedings.mlr.press/v9/glorot10a.html`.

[5] Sam Bond-Taylor et al. "Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models". In: *IEEE transactions on pattern analysis and machine intelligence* (2021).