




AutoEncoderToolkit.jl: A Julia package for training (Variational) Autoencoders

Manuel Razo-Mejia ^{1,2}

¹ Department of Biology, Stanford University, CA, United States of America ² For correspondence, contact mrazo@stanford.edu

DOI: [10.21105/joss.06794](https://doi.org/10.21105/joss.06794)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: Fabian Scheipl 

Reviewers:

- [@sandeshkatakam](#)
- [@avik-pal](#)

Submitted: 10 May 2024

Published: 29 July 2024

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

With the advent of generative models, the field of unsupervised learning has exploded in the last decade. One of the most popular generative models is the variational autoencoder (VAE) (Kingma & Welling, 2014). VAEs assume the existence of a joint probability distribution between a high-dimensional data space and a lower-dimensional latent space. Using a variational inference approach, the VAE parametrizes this joint distribution with two neural networks—an encoder and a decoder. This approach allows the model to approximate the underlying low-dimensional structure that generated the observed data and generate new samples by sampling from the learned latent space. Several variations of the original VAE have been proposed to extend its capabilities and tackle different problems. Here, we present AutoEncoderToolkit.jl, a Julia package for training VAEs and its extensions. The package is built on top of the Flux.jl deep learning library (Innes, 2018) and provides a simple and flexible interface for training different flavors of VAEs. Furthermore, the package offers a set of utilities for the geometric analysis of the learned latent space.

Statement of need

Collecting and analyzing large high-dimensional datasets have become routine in several scientific fields. Therefore, the need to understand and uncover the underlying low-dimensional structure of these datasets has become more pressing than ever. VAEs have shown great promise in this regard, with applications ranging from single-cell transcriptomics (Lopez et al., 2018) to protein design (Lian et al., 2022) to the discovery of the governing equations of dynamical systems (Champion et al., 2019). However, most of these tools have been exclusively developed for the Python ecosystem. Julia is a promising high-performance language for scientific computing, with a growing ecosystem for deep learning, state-of-the-art automatic differentiation, and a strong GPU-accelerated computing backend. Furthermore, the programming paradigm of Julia, based on multiple dispatch, allows for a more flexible, modular, and composable codebase. AutoEncoderToolkit.jl aims to provide a simple and flexible interface for training VAEs and their extensions in Julia, taking full advantage of the language programming paradigm.

Software Description

Encoder & Decoder definition

AutoEncoderToolkit.jl takes a probability-theory-based approach to the package design. Taking advantage of the multiple dispatch paradigm of Julia, the package can easily combine and extend the encoders and decoders to quickly prototype different VAE architectures. In

other words, independent of the design of the multi-layer perceptron used for either the encoder or the decoder, what defines their behavior is their associated probability distribution. For example, let us consider the loss function for the standard VAE model, given by the evidence lower bound (ELBO):

$$\text{ELBO} = \langle \log p_{\theta}(x|z) \rangle_{q_{\phi}(z|x)} - D_{KL}(q_{\phi}(z|x) || p(z)), \quad (1)$$

where $p_{\theta}(x|z)$ —defined by the decoder with parameters θ —is the likelihood of the data given the latent variable, $q_{\phi}(z|x)$ —defined by the encoder with parameters ϕ —is the posterior distribution of the latent variable given the data, $D_{KL}(q_{\phi}(z|x) || p(z))$ is the Kullback-Leibler divergence between the posterior and the prior distribution of the latent space, and $\langle \cdot \rangle_{q_{\phi}(z|x)}$ is the expected value over the posterior distribution. By defining the decoder type as either `BernoulliDecoder` or `SimpleGaussianDecoder`, the user can decide whether the decoder parametrizes a likelihood function $p_{\theta}(x|z)$ as a Bernoulli distribution or a Gaussian distribution with constant diagonal covariance, respectively. This design choice allows for the quick prototyping of different architectures without the overhead of defining new specific losses for each type of decoder.

Furthermore, the design allows for the easy extension of the available encoders and decoders that can directly integrate into any available VAE model. For example, let us assume that for a particular problem, the user wants to define a decoder whose outputs are the parameters for independent Poisson distributions, each with a different parameter λ_i . In other words, on the decoder side, the decoder returns a vector of parameters λ for each of the dimensions of the data. The user can define a new decoder type

```
struct PoissonDecoder <: AbstractVariationalDecoder
    decoder::Flux.Chain
end # struct
```

With this struct defined, the user only needs to define two methods: one for the forward pass of the decoder

```
function (decoder::PoissonDecoder)(z::AbstractArray)
    # Forward pass through decoder
    return (λ=decoder.decoder(z),)
end # function
```

and another for the likelihood of the data given the latent variable

```
function decoder_loglikelihood(
    x::AbstractArray,
    z::AbstractVector,
    decoder::PoissonDecoder,
    decoder_output::NamedTuple;
)
    # Extract the lambda parameter of the Poisson distribution
    λ = decoder_output.λ

    # Compute log-likelihood
    loglikelihood = sum(x .* log.(λ) - λ - loggamma.(x .+ 1))

    return loglikelihood
end # function
```

where we use the log-likelihood function of multiple independent Poisson distributions, given by

$$\ln p(x|\lambda) = \sum_i x_i \log(\lambda_i) - \lambda_i - \log(\Gamma(x_i + 1)). \quad (2)$$

With these methods defined, the `PoissonDecoder` can be directly integrated into any of the different VAE models provided by the package.

Implemented models

At the time of this writing, the package has implemented the following models:

Table 1: List of implemented (variational) autoencoder models

Name	Reference
Deterministic Autoencoder	
Vanilla Variational Autoencoder (VAE)	(Kingma & Welling, 2014)
β -Variational Autoencoder (β -VAE)	(Higgins et al., 2017)
Maximum-Mean Discrepancy Variational Autoencoder (InfoVAE)	(Zhao et al., 2018)
InfoMax Variational Autoencoder (InfoMaxVAE)	(Rezaabad & Vishwanath, 2020)
Hamiltonian Variational Autoencoder (HVAE)	(Caterini et al., 2018)
Riemannian Hamiltonian Variational Autoencoder (RHVAE)	(Chadebec et al., 2020)

Other than the deterministic autoencoder, all the models listed above use the same underlying VAE struct as part of their definition. Some of them, like the InfoMaxVAE and RHVAE require additional neural networks for training or inference. However, other than those additional elements, the training routines for all models are virtually the same. This design choice allows users to quickly explore different VAE models for their specific applications without writing new training routines for each model.

Moreover, extending the list of VAE models is also straightforward as contributions to the package only need to focus on a general definition of the loss function associated with the new VAE model without the need to define specific terms for each type of encoder or decoder.

Differential geometry utilities

In recent years, there has been a growing interest in understanding the geometric properties of the latent space learned by VAEs (Arvanitidis et al., 2021; Chadebec & Allasonnière, 2022). This is because the non-linearities of the encoder and decoder networks can induce complex geometries in the latent space, where the Euclidean distance between points in the latent space does not necessarily reflect the true distance between the corresponding data points. Thus, tools from differential geometry such as geodesic distance, parallel transport, and curvature can provide deeper insights into the structure of the learned latent space. AutoEncoderToolkit.jl provides a set of utilities for the geometric analysis of the latent space. For example, at the time of this writing, the NeuralGeodetics module provides the tools to approximate geodesic curves between points in latent space for the Riemannian Hamiltonian VAE (RHVAE) model. This is achieved by utilizing a neural network to approximate the geodesic equation (Chen et al., 2018) in the latent space using the explicit representation of the Riemannian metric learned by the RHVAE model (Chadebec et al., 2020).

GPU support

AutoEncoderToolkit.jl offers GPU support for CUDA.jl compatible GPUs out of the box.

Documentation

Documentation is available at (<https://mrzomej.github.io/AutoEncoderToolkit.jl>), where there are worked-out examples and tutorials on how to use the package.

Acknowledgements

I would like to thank Griffin Chure, Madhav Mani, and Dmitri Petrov for their advice and helpful discussions during the development of this package. I also want to thank the Schmidt Science Fellows program for funding part of this work via a postdoctoral fellowship. I would also like to thank the reviewers of this manuscript for their helpful comments and suggestions. The journal's transparent peer review process has dramatically improved the quality of this work.

References

- Arvanitidis, G., Hansen, L. K., & Hauberg, S. (2021, December 13). *Latent Space Oddity: On the Curvature of Deep Generative Models*. <https://doi.org/10.48550/arXiv.1710.11379>
- Caterini, A. L., Doucet, A., & Sejdinovic, D. (2018). *Hamiltonian Variational Auto-Encoder*. 11. <https://doi.org/10.48550/arXiv.1805.11328>
- Chadebec, C., & Allasonnière, S. (2022, November 3). *A Geometric Perspective on Variational Autoencoders*. <https://doi.org/10.48550/arXiv.2209.07370>
- Chadebec, C., Mantoux, C., & Allasonnière, S. (2020, October 22). *Geometry-Aware Hamiltonian Variational Auto-Encoder*. <https://doi.org/10.48550/arXiv.2010.11518>
- Champion, K., Lusch, B., Kutz, J. N., & Brunton, S. L. (2019). Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45), 22445–22451. <https://doi.org/10.1073/pnas.1906995116>
- Chen, N., Klushyn, A., Kurle, R., Jiang, X., Bayer, J., & Smagt, P. (2018). Metrics for Deep Generative Models. *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, 1540–1550. <https://proceedings.mlr.press/v84/chen18e.html>
- Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., & Lerchner, A. (2017). *B-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework*.
- Innes, M. (2018). Flux: Elegant machine learning with Julia. *Journal of Open Source Software*, 3(25), 602. <https://doi.org/10.21105/joss.00602>
- Kingma, D. P., & Welling, M. (2014, May 1). *Auto-Encoding Variational Bayes*. <https://doi.org/10.48550/arXiv.1312.6114>
- Lian, X., Praljak, N., Subramanian, S. K., Wasinger, S., Ranganathan, R., & Ferguson, A. L. (2022). *Deep learning-enabled design of synthetic orthologs of a signaling protein* [Preprint]. *Molecular Biology*. <https://doi.org/10.1101/2022.12.21.521443>
- Lopez, R., Regier, J., Cole, M. B., Jordan, M. I., & Yosef, N. (2018). Deep generative modeling for single-cell transcriptomics. *Nature Methods*, 15(12), 1053–1058. <https://doi.org/10.1038/s41592-018-0229-2>
- Rezaabad, A. L., & Vishwanath, S. (2020, January 7). *Learning Representations by Maximizing Mutual Information in Variational Autoencoders*. <https://doi.org/10.48550/arXiv.1912.13361>
- Zhao, S., Song, J., & Ermon, S. (2018, May 30). *InfoVAE: Information Maximizing Variational Autoencoders*. <https://doi.org/10.48550/arXiv.1706.02262>