Explicit and data-Efficient Encoding via Gradient Flow

Kyriakos Flouris
Computer Vision Laboratory, ETH Zurich
kflouris@vision.ee.ethz.ch

Anna Volokitin
Computer Vision Laboratory, ETH Zurich
voanna@vision.ee.ethz.ch

Gustav Bredell
Computer Vision Laboratory, ETH Zurich
gustav.bredell@vision.ee.ethz.ch

Ender Konukoglu
Computer Vision Laboratory, ETH Zurich
ender.konukoglu@vision.ee.ethz.ch

Abstract

The autoencoder model typically uses an encoder to map data to a lower dimensional latent space and a decoder to reconstruct it. However, relying on an encoder for inversion can lead to suboptimal representations, particularly limiting in physical sciences where precision is key. We introduce a decoder-only method using gradient flow to directly encode data into the latent space, defined by ordinary differential equations (ODEs). This approach eliminates the need for approximate encoder inversion. We train the decoder via the adjoint method and show that costly integrals can be avoided with minimal accuracy loss. Additionally, we propose a 2^{nd} order ODE variant, approximating Nesterov's accelerated gradient descent for faster convergence. To handle stiff ODEs, we use an adaptive solver that prioritizes loss minimization, improving robustness. Compared to traditional autoencoders, our method demonstrates explicit encoding and superior data efficiency, which is crucial for data-scarce scenarios in the physical sciences. Furthermore, this work paves the way for integrating machine learning into scientific workflows, where precise and efficient encoding is critical. [1]

1 Introduction

Auto-encoders are widely successful in supervised learning due to their ability to learn lower-dimensional representations of input data, enabling efficient computation [15]. The basic idea is that when sufficient correlation exists within the input data, the latent dimension can generate a model of the input. However, since the encoder is not directly learned, the encoding process can be semi-arbitrary, leading to suboptimal latent space representations and inefficient learning [11]. To address this, efforts have been made to directly regularize the latent space [14]. Typically, the encoder approximates the inverse of the decoder, but this requires learning additional parameters, impeding learning, particularly with limited data. Flow models [4, 6] attempt to resolve this by using invertible maps, but they are restricted to equi-dimensional latent spaces. What if we could eliminate

[1] The code for this work is available at https://github.com/k-flouris/gfe.

Machine Learning and the Physical Sciences Workshop, NeurIPS 2024.

the encoder, retain the advantages of a lower-dimensional latent space, and directly optimize the representation? This could enable faithful reconstruction with fewer samples and iterations.

Currently, DeepSDF [11] optimizes latent space representations using a continuous signed distance function to model 3D shapes, where a latent vector z represents the shape encoding. Hamiltonian Neural Networks [8] and VampPrior [13] introduce physically-informed dynamics and more expressive priors for VAEs, respectively, contributing to latent space optimization. SAVAE [9] and the energy-based model approach in [10] directly aim to optimize the encoder through iterative refinement.

In this work, we propose a novel encoding method, namely gradient flow encoding (GFE). This decoder-only method explicitly determines the optimal latent space representation at each training step via a gradient flow, namely, an ordinary differential equation (ODE) solver. The decoder is updated by minimizing the loss between the input image and its reconstruction from the optimal latent space representation. Although slower, this method converges faster and demonstrates superior data efficiency and ultimately achieves better reconstructions with a minimal number of training samples compared to a conventional auto-encoder (AE). A notable advantage of GFE is the reduction in network size, as it eliminates the need for an encoder.

Using a gradient flow for encoding can be computationally demanding. ODE solvers with adaptive step sizes aim to minimize integration error, which is crucial for accuracy. However, these solvers can slow down training when the ODE becomes stiff, as step sizes shrink and make integration time-consuming [3, 7]. This is especially problematic in GFE-based training. We find that the exact gradient flow path may be less important than reaching convergence, so minimizing integration error may not be optimal. Fixed step sizes also cause poor convergence and stability in decoder training. To address this, we develop an adaptive solver that adjusts each step to minimize the loss between input and output. We also introduce a loss convergence assertion in the adaptive minimize distance (AMD) solver, making gradient flow more computationally feasible for neural networks.

For optimizing the latent space and decoder in GFE, we implement an adjoint method [2], which has also been used in other recent works [3]. To improve efficiency, we demonstrate that a full adjoint method may not always be necessary, and an approximation can be effectively utilized. Moreover, we introduce a Nesterov second-order ODE solver, which accelerates convergence for each training data size. Ultimately, the approximate GFE utilizing AMD (GFE-amd) is employed for testing and comparison with a traditional AE solver. Our flow-inspired model offers a solution for the stringent data efficiency and robustness demands in fields like physics, astronomy, and materials science. By utilizing the flow-model for encoding, we contribute to algorithms that create interpretable, accurate predictive models, enabling more robust discoveries and insights in scientific research.

2 Method

An auto-encoder maps an input y to a lower-dimensional latent space z via an encoder E, then reconstructs it using a decoder D, with E and D acting as approximate inverses. During training, each sample is mapped to z by E, reconstructed by D, and the reconstruction error is minimized with respect to both networks' parameters. At any point, an optimal latent representation z^* minimizes the reconstruction error, but the encoder doesn't directly map to z^*. Instead, $E(z)$ is updated to make its reconstruction closer to the sample, which may not be efficient. Alternatively, z^* can be found and used to update the decoder directly.

Determination of z^* for each sample y can be formulated as an optimization problem: $z^* = \arg\min_z l(y, D(z, \theta))$, where θ represents the parameters of the decoder network and $l(\cdot, \cdot)$ is a distance function between the sample and its reconstruction by the decoder, which can be defined based on the application. One common form is the L_2 distance $\|D(z, \theta) - y\|_2^2$. The optimization can be achieved by gradient descent minimization. To integrate this minimization into the training of the decoder network, a continuous gradient descent algorithm is implemented via the solution to an ODE:

$$\frac{dz}{dt} = -\alpha(t)\nabla_z l(y, D(z(t), \theta)), \quad z(0) = 0, \tag{1}$$

where time t is the continuous parameter describing the extent of the minimization, and $\alpha(t)$ is a scaling factor that can vary with time. When the extremum is reached, z reaches a steady state. In

practice, we compute the optimal z^* by integrating the ODE:

$$z^* = z(\tau) = \int_0^\tau -\alpha(t)\nabla_z l(y, D(z(t), \theta))\, dt, \quad z(0) = z_0, \tag{2}$$

where z_0 is the initialization of the optimization, which is set as the zero vector in our experiments. Consequently, after minimizing $z \to z^* \equiv z(t = \tau)$ for a given decoder D (the 'forward model'), the decoder is trained with a total loss function for a given training set M (the 'backward step'):

$$\mathcal{L}(\theta) = \sum_{m=1}^{M} l(y_m, D(z_m^*, \theta)). \tag{3}$$

At each iteration, while searching for $\arg\min_\theta \mathcal{L}(\theta)$, a new z^* is recalculated for each sample. $\arg\min_\theta \mathcal{L}(\theta)$ is computed via the standard adjoint method, as explained in App. A.1.

2.1 Nesterov's 2nd-Order Accelerated Gradient Flow

The gradient flow described above is based on naive gradient descent, which may be slow in convergence. The convergence per iteration can be further increased by considering Nesterov's accelerated gradient descent. A second-order differential equation approximating Nesterov's accelerated gradient method has been developed in [12], and additionally incorporated into Adam [5]. This 2^{nd} order ODE for z is given by:

$$\frac{d^2 z}{dt^2} + \frac{3}{t}\frac{dz}{dt} + \nabla_z l(y, D(z, \theta)) = 0, \tag{4}$$

with the initial conditions $dz/dt|_{t=0} = z(0) = 0$. To incorporate this within the gradient flow encoding framework, we split the 2^{nd} order ODE into two interacting 1^{st} order equations and solve them simultaneously. Specifically, we solve:

$$\frac{dv}{dt} = -\frac{3}{t+\epsilon}v + \nabla_z l(y, D(z, \theta)), \quad \frac{dz}{dt} = v, \tag{5}$$

where ϵ ensures stability at small t.

2.2 Adaptive minimise distance solver

Step size is crucial in discretized optimization algorithms for reaching a local extremum, including solving the gradient flow equation for optimal latent space representation. Fixed time-step solvers can cause instabilities during decoder training due to stiffness, as predefined time slices may not adapt to rapid changes in the gradient flow, see App. A.2. For example, a 4^{th} order Runge-Kutta method with a fixed grid uses δt slices in a logarithmic series to manage variations near $t = 0$, but this can still lead to instability, particularly in forward and backward passes. While adaptive step-size ODE solvers can theoretically address these issues, stiffness in the gradient flow equation remains a challenge.

Adaptive step-size ODE solvers address stiffness in gradient flow equations but prioritize accurate integration, which isn't always useful for decoder training. A more effective approach minimizes loss at each step, regardless of the integration path. To achieve this, we develop an adaptive step-size method resembling an explicit ODE solver, like the forward Euler method, but without a fixed grid. The challenge is solving ODE 1 while selecting time-steps that reduce $l(y, D(z(t), \theta))$. Essentially, this is gradient descent with adaptive step-size selection [1]. In this framework, $\alpha(t)$ is redundant and can be set to 1, with the time-step δt taking over its role.

In the AMD method, at each time t_n, δt_n is chosen by finding the smallest $m = 0, 1, ...$ that satisfies:

$$l\left(y, D(z(t_n) - \beta^m s_n \nabla_z l(y, D(z(t_n), \theta)), \theta)\right) < l\left(y, D(z(t_n), \theta)\right), \tag{6}$$

with $\beta \in (0, 1)$ (set to 0.75 in our experiments) and s_n as a scaling factor. At each time point t_n, the time-step is set as $\delta t_n = \beta^m s_n$. The scaling factor is updated at each iteration as $\hat{s}_n = \max(\kappa s_{n-1}, s_0)$, $s_n = \min(\hat{s}_n, s_{max})$, $s_{max} = 10$, $s_0 = 1$, $\kappa = 1.1$. Based on this, $t_{n+1} = t_n + \delta t_n$ and

$$z(t_{n+1}) = z(t_n) - \delta t_n \nabla_z l(y, D(z(t_n), \theta)). \tag{7}$$

If the chosen time-step goes beyond τ, a smaller step is used to reach τ exactly. The solution of the integral of Eq. 1 is then $z(\tau)$. Furthermore, the AMD solver monitors the gradient of the convergence curve to determine if the loss function is sufficiently optimized, allowing it to assign a new final τ' and stop early to avoid unnecessary integration.

3 Results and Discussion

A direct comparison of GFE-amd to a conventional AE for MNIST training is shown in Fig. 1. Further experimental details can be found in App. B. The x-axis shows the number of training images processed, not iterations. The training uses mini-batch data with replacement, revisiting the data multiple times. GFE-amd demonstrates significantly better learning per image, nearly converging at 800,000 images, see Fig. 1 left. This is a consequence of efficient latent space optimization. However, this comes with higher computational cost per iteration due to the ODE solver, see Fig. 1 right. Both models use the Adam optimizer, therefore the difference can be attributed to better gradients the GFE-amd model generates to update the decoder network at each training iteration.

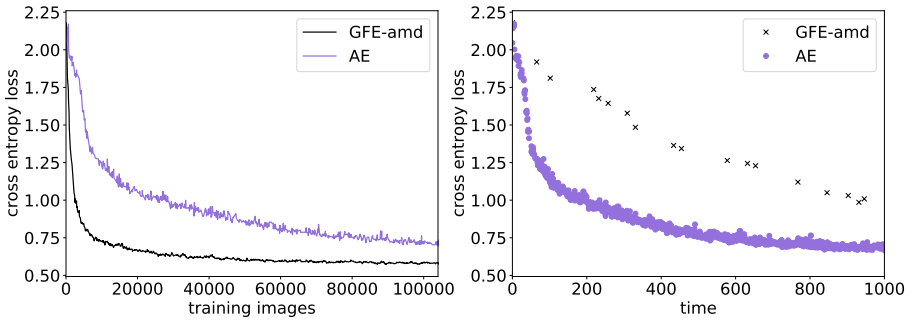

Figure 1: **Left** Validation mean cross-entropy loss vs. number of MNIST training images for GFE-amd and AE methods, with GFE-amd showing significant convergence with minimal training data. **Right** Validation mean cross-entropy loss vs. time for GFE-amd and AE methods, with AE being faster due to more iterations in the same time span.

This increase in computation is not necessarily a disadvantage, considering the efficient learning of the GFE-amd method. In Table 1 right, the average cross entropy loss for a complete test-set is recorded for both methods for some small number of training images. The GFE-amd is able to learn quite well even after seeing a tiny fraction of the total training data. Furthermore, the GFE-amd method noticeably improves an AE trained decoder when it is used to test, the result of an optimized latent space even without a network parameters update.

To verify the overall quality of the method both the AE and GFE-amd are tested when converged as shown in Table 1 left. The GFE-amd performs very similar to AE both for MNIST, Segmented MNIST (SegMNIST) and FMNIST. It is worth noting that the GFE-amd trainings are on average converged at $1/12^{th}$ of the number of iterations relative to the AE. For the SegMNIST the networks are fully trained while seeing only the first half (0-4) of the MNIST labels and they are tested with the second half (5-9) of the labels. The GFE-amd shows a clear advantage over the AE emphasizing the versatility of a GFE-amd trained neural network.

The interpretable nature of the proposed method inherently supports scalability, suggesting that it will remain data-efficient even when applied to real-world or large-scale datasets.

Table 1: Average cross-entropy loss: **Left** Table shows different MNIST training sizes (percentage is of total) the mixed column refers to AE training and then GFE-amd testing. **Right** Table shows complete training tests results on different datasets.

Training Images	AE	GFE-amd	Mixed
480 (0.24%)	0.2660	**0.2098**	0.2634
1920 (0.98%)	0.2488	**0.1558**	0.2323
5760 (2.9%)	0.1954	**0.1136**	0.1829

Dataset	AE	GFE-amd
MNIST	0.0843	**0.0830**
SegMNIST	0.1205	**0.1135**
FMNIST	**0.2752**	**0.2764**

4 Conclusions

To this end, a gradient flow encoding, decoder-only method was investigated. The decoder-dependent gradient flow searches for the optimal latent space representation, which eliminates the need for an approximate inversion. The full adjoint solution and its approximation are leveraged for training and compared in various settings. Furthermore, we present a 2^{nd} order ODE variant to the method, which approximates Nesterov's accelerated gradient descent, with faster convergence per iteration. Additionally, an adaptive solver that prioritizes minimizing loss at each integration step is described and utilized for comparative tests against the autoencoding model. While each training step takes longer, the gradient flow encoding converges faster and demonstrates much higher data efficiency than the autoencoding model. This flow-inspired approach opens up new possibilities for efficient and robust data representations in scientific fields such as physics and materials science, where accurate modeling and data efficiency are paramount.

Acknowledgement

This project was supported by grants #2022-531 and #2022-643 of the Strategic Focus Area "Personalized Health and Related Technologies (PHRT)" of the ETH Domain (Swiss Federal Institutes of Technology).

References

[1] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, 2016.

[2] J. C. Butcher and N. Goodwin. *Numerical methods for ordinary differential equations*, volume 2. Wiley Online Library, 2008.

[3] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[4] L. Dinh, D. Krueger, and Y. Bengio. NICE: non-linear independent components estimation. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[5] T. Dozat. Incorporating Nesterov Momentum into Adam. In *Proceedings of the 4th International Conference on Learning Representations*, pages 1–4.

[6] K. Flouris and E. Konukoglu. Canonical normalizing flows for manifold learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 27294–27314. Curran Associates, Inc., 2023.

[7] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

[8] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks, 2019.

[9] Y. Kim, S. Wiseman, A. Miller, D. Sontag, and A. Rush. Semi-amortized variational autoencoders. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2678–2687. PMLR, 10–15 Jul 2018.

[10] B. Pang, T. Han, E. Nijkamp, S.-C. Zhu, and Y. N. Wu. Learning latent space energy-based prior model, 2020.

[11] J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 165–174, 06 2019.

[12] W. Su, S. Boyd, and E. J. Candès. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. *Journal of Machine Learning Research*, 17(153):1–43, 2016.

[13] J. M. Tomczak and M. Welling. Vae with a vampprior, 2018.

[14] M. Tschannen, O. Bachem, and M. Lucic. Recent advances in autoencoder-based representation learning. *ArXiv*, abs/1812.05069, 2018.

[15] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010.

A Method details

A.1 The adjoint method for the gradient flow

As described above, after finding, $z^* \equiv z(\tau) = \arg_z \min l(y, D(z, \theta))$ the total loss is minimized with respect to the model parameters. The dependence of z^* to θ creates an additional dependence of $\mathcal{L}(\theta)$ to θ via z^*. For simplicity, let us consider the cost of only one sample y, effectively $l(y, D(z^*, \theta))$. We will compute the total derivative $d_\theta l(y, D(z^*, \theta))$ for this sample, and the derivative of the total cost for a batch of samples can be computed as the sum of the sample derivatives in the batch. The total derivative $d_\theta l(y, D(z^*, \theta))$ is computed as

$$d_\theta l(y, D(z^*, \theta)) = \partial_\theta l(y, D(z^*, \theta)) + \partial_{z^*} l(y, D(z^*, \theta)) \partial_\theta z^*.$$

The derivative $\partial_{z^*} l(y, D(z^*, \theta)) d_\theta z^*$ can be computed using the adjoint method and leads to the following set of equations

$$dz/dt = -\alpha(t) \nabla_z l(y, D(z(t), \theta)), \text{ with } z(0) = 0 \tag{8}$$

$$d\lambda/dt = -\alpha(t) \lambda^T \nabla_z^2 l(y, D(z(t), \theta)), \text{ with } \lambda(\tau) = -\nabla_z l(y, D(z(\tau), \theta)) \tag{9}$$

$$d_\theta l(y, D(z^*, \theta)) = \partial_\theta l(y, D(z^*, \theta)) - \int_0^\tau \alpha(t) \lambda^T \partial_\theta \nabla_z l(z(t), \theta) dt, \tag{10}$$

where we used $z^* = z(\tau)$. Equations [8-10] define the so called adjoint method for gradient flow optimization of the loss. Due to the cost of solving all three equations, we empirically find that for this work sufficient and efficient optimization can be accomplished by ignoring the integral ("adjoint function") part of the method. Theoretically, this is equivalent to ignoring the higher order term of the total differential $d_\theta l(y, D(z^*, \theta)) = \partial_\theta l(y, D(z^*, \theta)) + \partial_z l(y, D(z^*, \theta)) \partial_\theta z \approx \partial_\theta l(y, D(z^*, \theta))$. Reducing Equations [8-10] to:

$$dz/dt = -\alpha(t) \nabla_z l(y, D(z(t), \theta)), \text{ with } z(0) = 0 \tag{11}$$

$$d_\theta \mathcal{L} = \partial_\theta l(y, D(z^*, \theta)), \tag{12}$$

i.e. optimise the latent space via solving an ordinary differential equation and minimise the loss "naively" with respect to the parameters ignoring the dependence of $z(\tau)$ to θ.

A.2 Fixed grid ODE solver

Being an ODE, the gradient flow can be solved with general ODE solvers. Unfortunately, generic adaptive step size solvers are not useful because the underlying ODE becomes stiff quickly during training, the step size is reduced to extremely small values and the time it takes to solve gradient flow ODEs at each iteration takes an exorbitant amount of time. Fixed time-step or time grid solvers can be used, despite the stiffness. However, we empirically observed that these schemes can lead to instabilities in the training, see Fig. 2. To demonstrate this, we experimented with a 4^{th} order Runge-Kutta method with fixed step size. The δt slices are predefined in logarithmic series such as δt is smaller closer to $t = 0$, where integrands, $-\nabla_z l(y, D(z, \theta))$, are more rapidly changing. Similarly, α is empirically set to $e^{(-2t/\tau)}$ to facilitate faster convergence of z. For the GFE full adjoint method, the integrands for each time slice are saved during the forward pass so they can be used for the calculation of the adjoin variable λ in the backward pass. We used the same strategy for both basic gradient flow and the 2^{nd} order model.

Algorithm 1: training GFE-amd for one sample per batch

Input: z(0)
Output: $\mathbf{y}_b$, θ_D, $\mathbf{z}_b^*$ **for** $b \in \{1, 2, ..., B\}$ **do**
  while $\frac{d}{db} L_{CE}(b) < 0.01$ **do**
    while $l(z(t_b^n + \delta t_b^n) < l(z(t_b^n))$ **do**
      $z(t_b^{n+1}) \leftarrow z(t_b^n) + [-\nabla_z l(z(t_b^n))]\delta t_b^n$
      $\delta t_b^{n+1} \leftarrow \beta^r \delta t_b^n$
      $r \leftarrow r + 1$
    end
    $n \leftarrow n + 1$
  end
  $z_b^* \leftarrow z(t_b^n)$
  $L(\theta_D) \leftarrow l(y, D(z^*, \theta_D)$
  $\theta_D \leftarrow$ backpropagate $L(\theta_D)$
end

B Experiments on the proposed methods

For training with MNIST and FashionMNIST datasets, we implement a sequential linear neural network. The decoder network architecture corresponds to four gradually increasing linear layers, with ELU non-linearities in-between. The exact reverse is used for the encoder of the AE.

The network training is carried out with a momentum gradient decent optimiser (RMSprop), learning rate 0.0005, $\varepsilon = 1 \times 10^{-6}$ and $\alpha = 0.9$. The GFE and AE are considered trained after one epoch and twelve epochs respectively.

Initially a relative comparison between the full adjoint and the approximate fixed grid GFE methods is carried out to assess the relevance of the higher order term. Specifically, we carry out MNIST experiments for a fixed network random seed, where we trained the Decoder using the different GFE methods and computed cross entropy loss over the validation set. The proper adjoint solution requires Equations 8 and 9 to be solved for each slice of the integral in Eq. 10. Given N time-slices (for sufficient accuracy $N \approx 100$), this requires $\mathcal{O}(5N)$ calls to the model D for each training image. The approximate method as in Equations 11 and 12 requires only $\mathcal{O}(N)$ passes. From Fig. 2 (left) it is evident that the 5-fold increase in computational time is not cost-effective as the relative reduction in loss convergence per iteration is not significant.

Furthermore, to increase convergence with respect to training data, the accelerated gradient flow 2^{nd} order GFE is implemented in Section 2.1. From Fig. 2 (right), the accelerated gradient method increases initially the convergence per iteration relative to GFE, nevertheless it is slightly more computationally expensive due to solving a coupled system. Additionally, from the same Fig. certain stability issues are observed for both GFE and second order GFE methods later on despite the initial efficient learning. In order to guarantee stability, the GFE-amd method is implemented as explained in Section 2.2. The black curve in Fig. 2 (right) shows a clear improvement of the GFE-amd over the later methods. Importantly, this result is robust to O of the experiment.

C Further results

Sample test-set reconstructions with a fixed network random seed for GFE-amd and AE methods are shown in Fig. 3. From Fig. 3 (a) it is evident that the GFE-amd is superior in producing accurate reconstructions with the very limited amount of data. Fig. 3 (b) indicates that both GFE-amd and AE generate similar reconstructions when properly trained.

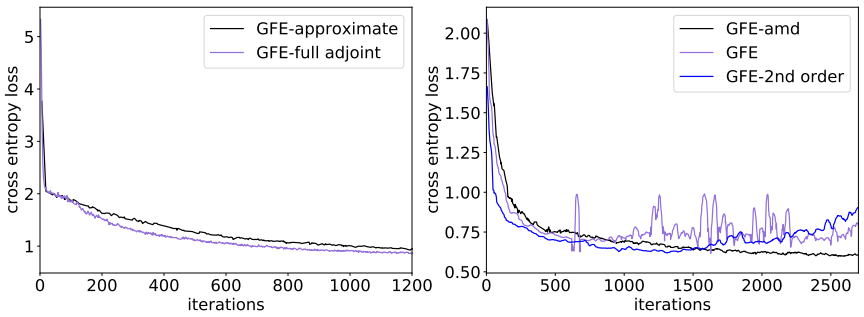

Figure 2: **Left** Validation mean cross-entropy loss plotted against MNIST training iterations for the approximate and full adjoint GFE methods. The full adjoint has a slight advantage over the approximate. **Right** Validation mean cross-entropy loss plotted against MNIST training iterations for the GFE, 2^{nd} order GFE and GFE-amd methods. The GFE-amd is both more stable and approaches a better convergence relative to the other methods

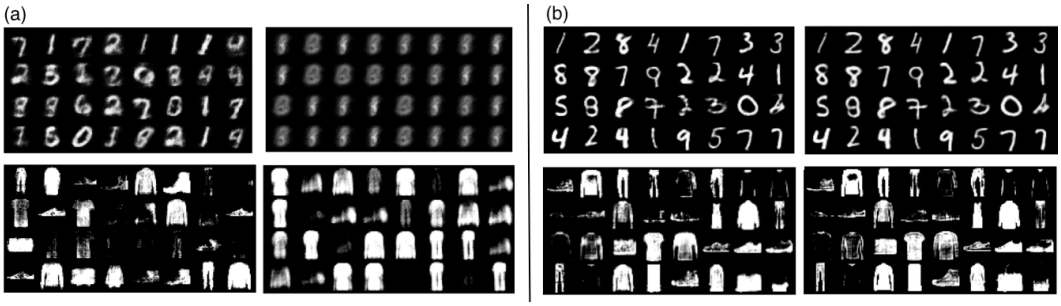

Figure 3: **(a)** Test-set reconstructions for trained GFE-amd (left) and AE (right) that only see 1% of MNIST (top) and FashionMNIST (bottom) training images. **(b)** Test-set reconstructions for fully trained GFE-amd (left) and AE (right) with MNIST (top) and FashionMNIST (bottom) training images. Note: The labels are identical in the respective reconstructions.