Gradient flow encoding with distance optimization adaptive step size

Anonymous authors
Paper under double-blind review

Abstract

The autoencoder model uses an encoder to map data samples to a lower dimensional latent space and then a decoder to map the latent space representations back to the data space. Implicitly, it relies on the encoder to approximate the inverse of the decoder network, so that samples can be mapped to and back from the latent space faithfully. This approximation may lead to sub-optimal latent space representations. In this work, we investigate a decoder-only method that uses gradient flow to encode data samples in the latent space. The gradient flow is defined based on a given decoder and aims to find the optimal latent space representation for any given sample through optimisation, eliminating the need of an approximate inversion through an encoder. Implementing gradient flow through ordinary differential equations (ODE), we leverage the adjoint method to train a given decoder. We further show empirically that the costly integrals in the adjoint method may not be entirely necessary. Additionally, we propose a 2nd order ODE variant to the method, which approximates Nesterov's accelerated gradient descent, with faster convergence per iteration. Commonly used ODE solvers can be quite sensitive to the integration step-size depending on the stiffness of the ODE. To overcome the sensitivity for gradient flow encoding, we use an adaptive solver that priorit- ises minimising loss at each integration step. We assess the proposed method in comparison to the autoencoding model. In our experiments, GFE showed a much higher data-efficiency than the autoencoding model, which can be crucial for data scarce applications.

1 Introduction

Auto-encoders are wildly successful as artificial neural network architectures for unsupervised learn- ing Vincent et al. (2010). The appeal is centred on learning a lower dimensional representation of the input data allowing extremely efficient computation. The idea is simple as it is brilliant, provided there is correlation between the input data the latent dimension can be leveraged to output a model of the input. Nevertheless, the encoding process is semi-arbitrary as there is no direct learning of the encoder, rather than the decoder is optimised to rectify the encoding process. This sub-optimal latent space representations can lead to inefficient learning, Park et al. (2019). In part to compensate for this learning process extensive work has been done to regularise the latent space more directly, Tschannen et al. (2018).

Auto-encoders depend on the encoder to approximate the inverse of the decoder network, this ap- proximate inversion requires to be learned with additional parameters and therefore may get in the way of learning with fewer data and adversely affect the latent space structure. Some research in flow models, Dinh et al. (2015), can resolve this issue by using invertible maps however they are con- strained to equi-dimensional latent representations. What if we can eliminate the need of an encoder neural network altogether, retain the advantage of lower-dimensional latent space, while obtaining a directly optimised representation? This may allow us to create a model that can map images to lower dimensional latent space and reconstruct faithfully with fewer images and less iterations

In this work we propose a novel encoding method namely a gradient flow encoding (GFE). This decoder-only method at each training step primarily determines the optimal latent space representa- tion for each sample via a gradient flow, namely, an ordinary differential equation (ODE) solver. The decoder is updated as usual by minimising the loss between input image and its reconstruction re-

trieved from the optimal latent space representation for the image. The method, albeit being slower, is considerably superior in data efficiency and ultimately obtains better reconstructions with very few number of training samples when compared to a conventional auto-encoder (AE). A side-advantage of GFE is 'halving' the size of the AE neural network, no need for an encoder.

Using a gradient flow for encoding can be a computationally challenging task. Traditionally, ODE solvers with adaptive step size are propagating the variables in such a manner as to minimise the error of integration. These solvers are crucial for the accuracy of the ODE solution. However, they also impede training of ODE based neural networks if the underlying ODE defined by the network becomes stiff , Chen et al. (2018); Grathwohl et al. (2019). Adaptive step size becomes too small and integration takes long. When using GFE at each training step, using an ODE solver with adaptive step size can be a debilitating factor.

For integrating gradient flow in an encoding scheme, we observe that the exact path of the gradient flow may be less important than the final point of convergence. Therefore, adapting step size to minimise error in integration is not necessarily the best approach. Using a fixed step size is also not an option as it may result in poor convergence and stability issues in training the decoder network. Here, we develop an adaptive solver for the gradient flow, which adapts each optimisation ('time') step such as to directly minimise the distance, i.e. loss between input and output. Furthermore, this adaptive minimise distance (AMD) solver is modified to include a loss convergence assertion to improve performance. The AMD method can possibly make gradient flow a computationally feasible module to be used in neural network architectures.

Here, an adjoint method, Butcher & Goodwin (2008), as also used in other recent works Chen et al. (2018), is implemented to properly optimise the latent space and decoder of the GFE. Consequently, for efficiency considerations we show that a full adjoint method is not necessarily needed and an approximation is utilised. Furthermore, we present the implementation of a Nesterov 2nd order ODE solver with accelerated convergence per training data size. Ultimately, the approximate GFE utilising AMD (GFE-amd) is employed for testing and comparison with a traditional AE solver.

2 RELEVANT WORKS

DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation Park et al. (2019): The authors replace the auto-encoder network with an auto-decoder where, a similar to here, latent vector z is introduced. z represents the encoding of a desired shape. They map this latent vector to a 3D shape represented by a continuous signed Distance Function. I.e. For shape i with function f_τ and coordinate $\mathbf{x}$, $f_\tau(z_i, \mathbf{x}) \approx SDF^i(\mathbf{x})$. By conditioning of the network output on a latent vector, they model multiple SDFs with a single neural network.

3 METHOD

An auto-encoder funnels an input y into a lower dimensional latent space representation z using an encoder network E and reconstructs it back using a decoder network D. Here, the encoder E and decoder D networks can be thought as approximate inverses of each other. During training, each sample is mapped to the latent space using E, mapped back using D and the reconstruction error is minimised with respect to the parameters of both of the networks.

At any point in this training process, given the decoder an optimal latent space representation for each sample can be defined as the z^* that minimises the reconstruction error. The encoder however, does not map the sample to that optimal z^*. So, instead of trying to get the reconstruction of z^* to get closer to the sample, its parameters are updated to get the reconstruction of $E(z)$ to get closer to the sample. This may not be the most efficient use of data samples. Alternatively, one can determine z^* for each sample and update the decoder's parameters according to this optimal latent representation.

Determination of z^* for each sample y can be formulated as an optimisation problem

$$z^* = \arg_z \min l(y, D(z, \theta)),$$

where θ represent 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 obvious form can be the L_2 distance $\|D(z, \theta) - y\|_2^2$. The optimisation can be achieved by a gradient

decent minimisation. In order to integrate the minimisation in the training of the decoder network, a continuous gradient decent algorithm is implemented via a solution to an ordinary differential equation $dz/dt = -\alpha(t)\nabla_z l(y, D(z(t), \theta))$, where time t is the continuous parameter describing the extent of the minimisation and $\alpha(t)$ is a scaling factor that can vary with time. When the extremum is reached the z comes to a steady state. In practice we compute the optimal z^* by integrating the ODE

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

where z_0 is the initialisation of the optimisation, which is set as 0 vector in our experiments. Consequently to the minimisation of $z \to z^* \equiv z(t = \tau)$ for a given D ('forward model'), the decoder is trained with a total loss function for a given training set M ('backward'),

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

At each iteration while searching for the $\arg_\theta \min \mathcal{L}(\theta)$, a new z^* is recalculated for each sample. $\arg_\theta \min \mathcal{L}(\theta)$ is computed via the adjoint method as explained in Section 3.1.

3.1 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 minimised 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

$$dz/dt = -\alpha(t)\nabla_z l(y, D(z(t), \theta)), \ \text{with } z(0) = 0 \tag{2}$$

$$d\lambda/dt = -\alpha(t)\lambda^T \nabla_z^2 l(y, D(z(t), \theta)), \ \text{with } \lambda(\tau) = -\nabla_z l(y, D(z(\tau), \theta)) \tag{3}$$

$$d_\theta l(y, D(z^*, \theta)) = \partial_\theta l(y, D(z^*, \theta)) - \int_0^\tau \alpha(t)\lambda^T \partial_\theta \nabla_z l(z(t), \theta)dt, \tag{4}$$

where we used $z^* = z(\tau)$. Equations [2-4] define the so called adjoint method for gradient flow optimisation of the loss. Due to the cost of solving all three equations we empirically find that for this work sufficient and efficient optimisation 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 [2-4] to

$$dz/dt = -\alpha(t)\nabla_z l(y, D(z(t), \theta)), \ \text{with } z(0) = 0 \tag{5}$$

$$d_\theta \mathcal{L} = \partial_\theta l(y, D(z^*, \theta)), \tag{6}$$

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.

3.2 NESTEROV 2nd ORDER ACCELERATED GRADIENT FLOW

The gradient flow described above is based on naive gradient descent. This method may be slow in convergence. The convergence per iteration can be further increased by considering Nesterov's accelerated gradient descent. A second differential equation approximating Nesterov's accelerated gradient method has been developed in Su et al. (2016). This 2nd order ODE equation for z reads

$$\frac{d^2 z}{dt^2} + \frac{3}{t}\frac{dz}{dt} + \nabla_z l(y, D(z, \theta)) = 0 \tag{7}$$

for $dz/dt|_0 = z(0) = 0$. To be able to use this in the framework of the gradient flow encoding we split the 2nd order ODE into two interacting 1st order equations and solve the simultaneously. Specifically, solving

$$\frac{dv}{dt} = \frac{3}{t + \epsilon}v + \nabla_z l(y, D(z, \theta)), \qquad \frac{dz}{dt} = v, \tag{8}$$

where ϵ ensures stability at small t.

3.3 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 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 Figure 3. To demonstrate this, we experimented with a 4th 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, see Figure 1. For the GFE full adjoint method the integrands for each time slice are saved during the forward pass so they can be use for the calculation of the adjoin variable λ in the backward pass. We used the same strategy for both basic gradient flow and the 2nd order model.
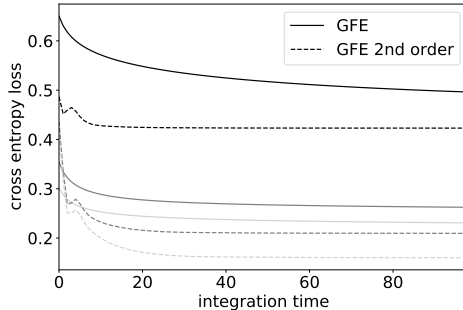

Figure 1: Typical convergence curve of z during an ODE integration for GFE and GFE 2nd order using the fixed grid ODE solver. Experiments were performed on MNIST. Mean cross-entropy loss plotted against time for three training iterations, darker corresponds to earlier iteration. Curves are obtained using the same batch of training images. The plot shows how convergence curves of z changes with training iterations. Darker curves are for earlier iterations. In later iterations, the convergence curves reach lower loss, meaning updates of the decoder network's parameters lead to more faithful latent space representations, as expected.

3.4 ADAPTIVE MINIMISE DISTANCE (AMD) SOLVER

The step size in any optimisation algorithm is of fundamental importance for reaching a local extremum in any optimisation problem. The same applies to the search for optimal latent space representation for each sample by solving gradient flow equation through its time discretisation. As discussed, fixed time step or time grid solvers empirically lead to instabilities during training of the decoder. Adaptive step size ODE solvers can theoretically solve the instabilities but due to the possible stiffness of the gradient flow equation, their use is impeded. Note that, the adaptive step size solvers focus on an accurate integration, which is not necessarily an advantage for integrating gradient flow in the training of a decoder. A solver that focuses on reaching a local extremum while searching for the optimal z would be more advantageous. The path it takes to reach there may be less crucial as long as it reaches the same local extremum. This is in contrast to generic ODE solvers where the path can substantially change the end point, thus an focusing on accurate integration is

more important. To this end, we develop an adaptive step size method, which guarantees a reduction of the loss at each step.

The method follows a similar structure to an explicit ODE solver, such as the feed-forward Euler method, but without a fixed grid. The problem lies in solving Equation 5 while taking time-steps of appropriate size that reduce $l(y, D(z(t), \theta))$ at each t. This approach is in essence a gradient descent method that uses step-size selection mechanisms Bertsekas (2016). Lastly, viewing the time-step in solving $dz/dt = -\alpha(t)\nabla_z l(y, D(z(t), \theta))$ as a tool to minimise the loss makes $\alpha(t)$ obsolete. Its role is now overtaken by the time-step δt and it can be set to 1 for all t.

In the AMD method, at each time t the time step is chosen based on finding the smallest $m = 0, 1, \dots$, that satisfies

$$l\left(y, D(z(t_n) - \beta^m s_n \nabla_z l(y, D(z(t_n), \theta)), \theta)\right) < l\left(y, D(z(t_n), \theta)\right) \tag{9}$$

with $\beta \in (0, 1)$ (set as 0.75 in our experiments) and s_n is a scaling factor. At each time point t_n the time step is chosen 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

$$z(t_{n+1}) = z(t_n) - \delta t_n \nabla_z l(y, D(z(t_n), \theta) \tag{10}$$

At the end, if time step chosen goes beyond τ, a smaller time step is used to reach τ exactly. The solution of the integral 5 is then $z(\tau)$. Furthermore, the AMD solver is using the gradient of the convergence curve (see Figure 1) to assert if the loss function is sufficiently optimised to assign a new final τ' and stop in order to avoid unnecessary integration.

4 EXPERIMENTAL SETUP

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. A schematic diagram of the GFE method is shown in Figure 2.
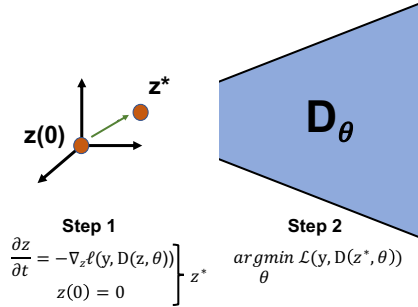

Figure 2: Schematic diagram of the GFE implementation. The two optimisation correspond to (Step 1) the latent space optimisation i.e. 'encoding' and (Step 2) the parameter update of the neural network.

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.

5 RESULTS AND DISCUSSION

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 2 and 3 to be solved for each slice of the integral in Equation 4. 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 5 and 6 requires only $\mathcal{O}(N)$ passes. From Figure 3 (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 2nd order GFE is implemented in Section 3.2. From Figure 3 (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 Figure 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 3.4. The black curve in Figure 3 (right) shows a clear improvement of the GFE-amd over the later methods. Importantly, this result is robust to O of the experiment.
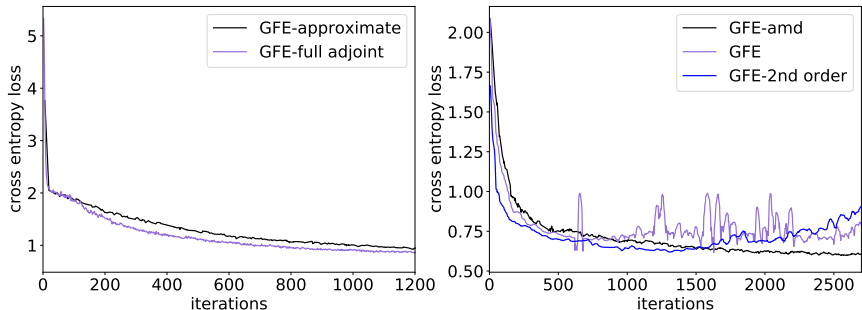

Figure 3: **Left** Validation mean cross-entropy loss plotted against MNIST training iterations for the approximate and full adjoint GFE methods. The full adjoint has a slight advantage over the approximate. **Right** Validation mean cross-entropy loss plotted against MNIST training iterations for the GFE, 2nd order GFE and GFE-amd methods. The GFE-amd is both more stable and approaches a better convergence relative to the other methods

A direct comparison of the GFE-amd to a conventional AE for an MNIST training can be seen in Figure 4. The x-axis shows the number of training images (instead of iterations) the algorithm sees until that point in the training. The training is based on mini-batch training using the data with replacement, going over the training data multiple times. The GFE-amd is substantially superior in learning per training image, reaching near convergence with at 800000 images, see Figure 4 (left). This is a consequence of the efficiently optimised latent space. Nevertheless, this comes at a higher computational cost for each iteration due to the ODE solver as seen from Figure 4 (right). Importantly, the optimisation of the network parameters is performed using Adam optimiser for both AE and GFE models. So the difference we see can be attributed to better gradients GFE model generates to update the decoder network at each training iteration.

This increase in computation is not necessarily a disadvantage considering the efficient learning of the GFE-amd method. In Table 1, 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 optimised 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 2. The GFE-amd performs very similar to AE both for MNIST, SegmentedMNIST 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 segmented MNIST 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
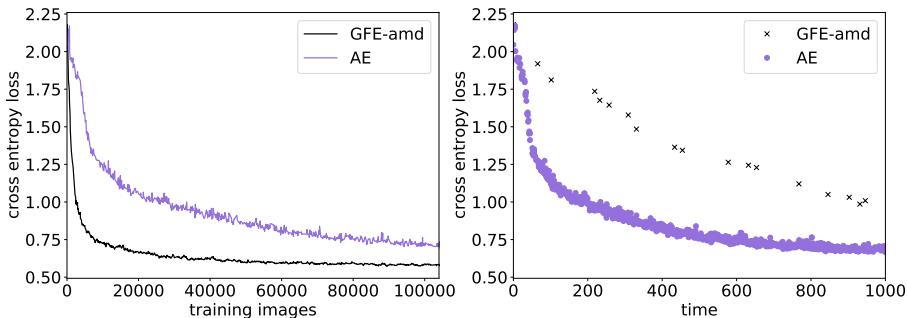
Figure 4: **Left** Validation mean cross-entropy loss plotted against number of MNIST training images for the GFE-amd and AE methods. The former shows a significant convergence with a tiny amound of training images. **Right** Validation mean cross-entropy loss plotted against time for the GFE-amd and AE methods. The latter is significantly faster to the former as much more iterations are carried out in the same time-span

Number of Training images	AE	GFE-amd	train:AE test:GFE-amd
480 (0.24%)	0.2660	0.2098	0.2634
960 (0.49%)	0.2618	0.1987	0.2525
1920 (0.98%)	0.2488	0.1558	0.2323
3840 (1.95%)	0.2195	0.1336	0.2038
5760 (2.9%)	0.1954	0.1136	0.1829

Table 1: Test-set average cross entropy loss for different number of training data, % reflects the percentage relative to total training data needed for convergence of the method. Carried out for the AE GFE-amd methods. The forth column represents testing the decoder of an AE trained network with GFE-amd. The GFE-amd is far superior here in learning with a limited image sample.

(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.

Dataset	AE (Test-set)	GFE-amd (Test-set)
MNIST	0.0843	0.0830
SegmentedMNIST	0.1205	0.1135
FMNIST	0.2752	0.2767

Table 2: Test-set average cross entropy loss for trained networks with the AE and GFE-amd methods. Segmented MNIST represents training for half the labels and testing with the other half.

Sample test-set reconstructions with a fixed network random seed for GFE-amd and AE methods are shown in Figure 5. From Figure 5 (a) it is evident that the GFE-amd is superior in producing accurate reconstructions with the very limited amount of data. Figure 5 (b) indicates that both GFE-amd and AE generate similar reconstructions when properly trained.

Finally, to further compare latent space representation, we visualize the samples in the latent space using the t-distributed stochastic neighbour embedding (t-SNE) map, van der Maaten & Hinton (2008). This is calculated for the GFE optimised $z \to z*$. This is shown for AE and GFE, MNIST trained neural networks in Figure 6. The latent space representations are similar when both models see the entire MNIST dataset multiple times. Similar t-SNE plots for models that only see 1% of the data during training are given in Figure 7. Latent space structure of GFE is very similar for both cases, while AE's latent space structure is very different, not clustering the different numbers. This result is inline with Table 1. GFE uses training images more efficiently at each iteration thanks to latent space optimisation to invert the decoder rather than using an approximation through an encoder network.
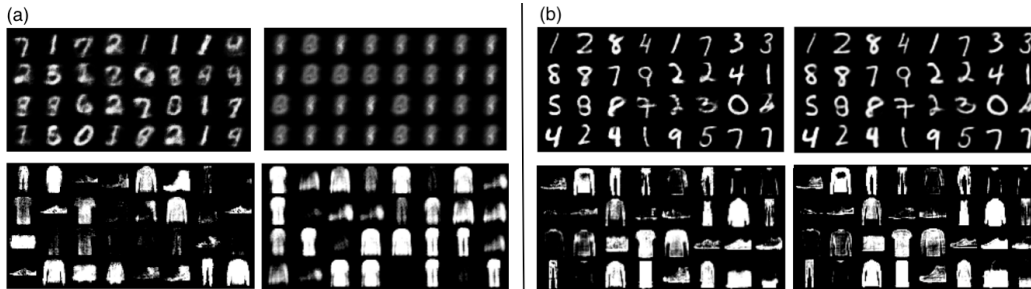
Figure 5: **(a)** Test-set reconstructions for trained GFE-amd (left) and AE (right) that only see 1% of MNIST (top) and FashionMNIST (bottom) training images. **(b)** Test-set reconstructions for fully trained GFE-amd (left) and AE (right) with MNIST (top) and FashionMNIST (bottom) training images. Note: The labels are identical in the respective reconstructions.
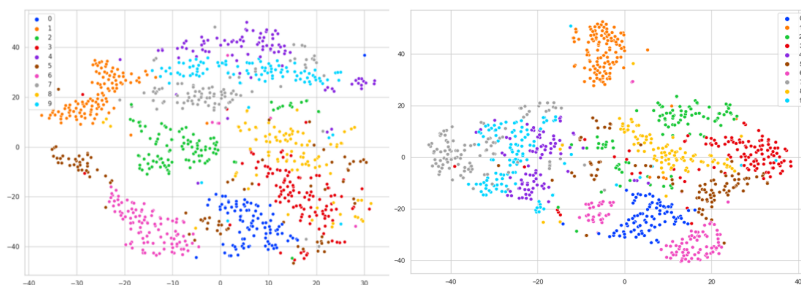

Figure 6: t-SNE map of the latent space plotted for MNIST trained **Left** GFE and **Right** AE methods.
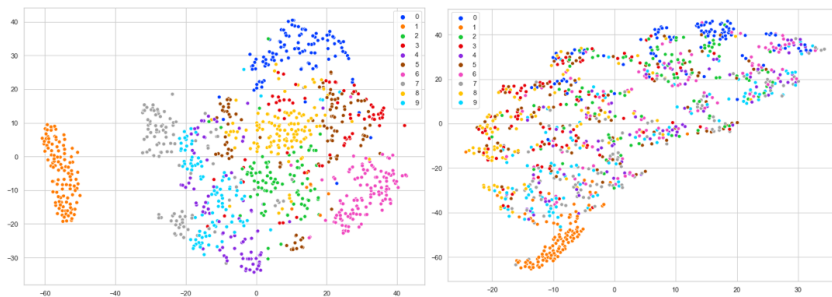

Figure 7: t-SNE map of the latent space plotted for MNIST trained with 1% training data **Left** GFE and **Right** AE methods. The GFE latent space is well optimised even with a fraction of the data.

6 CONCLUSIONS

To this end, a gradient flow encoding, decoder-only method was investigated. The decoder depended gradient flow searches for the optimal latent space representation, which eliminates the need of an approximate inversion. The full adjoint solution and its approximation or leveraged for training and compared. Furthermore, we present a 2nd order ODE variant to the method, which approximates Nesterov's accelerated gradient descent, with faster convergence per iteration. Additionally, an adaptive solver that prioritises minimising loss at each integration step is described and utilised for comparative tests to the autoencoding model. The gradient flow encoding shows a much higher data-efficiency than the autoencoding model.

REFERENCES

Dimitri P. Bertsekas. *Nonlinear programming*. Athena Scientific, 2016.

John Charles Butcher and Nicolette Goodwin. *Numerical methods for ordinary differential equations*, volume 2. Wiley Online Library, 2008.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015. URL http://arxiv.org/abs/1410.8516.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

Jeong Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. pp. 165–174, 06 2019. doi: 10.1109/CVPR.2019.00025.

Weijie Su, Stephen Boyd, and Emmanuel 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. URL http://jmlr.org/papers/v17/15-084.html.

Michael Tschannen, Olivier Bachem, and M. Lucic. Recent advances in autoencoder-based representation learning. *ArXiv*, abs/1812.05069, 2018.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371â3408, December 2010. ISSN 1532-4435.