Enforcing idempotency in neural networks

Nikolaj Banke Jensen
Department of Computer Science
University of Oxford
nikolaj.jensen@cs.ox.ac.uk

Jamie Vicary
Department of Computer Science and Technology
University of Cambridge
jamie.vicary@cl.cam.ac.uk

Abstract

In this work, we propose a new architecture-agnostic method for training idempotent neural networks. An idempotent operator satisfies $f(x) = f(f(x))$, meaning it can be applied iteratively with no effect beyond the first application. Neural networks used in data transformation tasks, such as image generation and augmentation, represent non-linear idempotent projections. Using methods from perturbation theory we derive the recurrence relation $\mathbf{K}' \leftarrow 3\mathbf{K}^2 - 2\mathbf{K}^3$ for iteratively projecting a real-valued matrix $\mathbf{K}$ onto the manifold of idempotent matrices. Our analysis shows that for linear, single-layer MLP networks this projection 1) has idempotent fixed points, and 2) is attracting only around idempotent points. We give an extension to non-linear networks by considering our approach as a substitution of the gradient for the canonical loss function, achieving an architecture-agnostic training scheme. We provide experimental results for MLP- and CNN-based architectures with significant improvement in idempotent error over the canonical gradient-based approach. Finally, we demonstrate practical applications of the method as we train a generative network successfully using only a simple reconstruction loss paired with our method.

1 Introduction

Using neural networks as data augmentation tools is becoming more widespread in areas such as signal processing and generative artificial intelligence. In particular, networks of the form $f : X \to X$, mapping data within the same space X, are frequently used in image augmentation [6], video generation [7, 5], sorting algorithms [11], compression algorithms [9, 5], image denoising [8, 3, 5], and image generation [5], among others. While there is a variety of such transformation tasks, some of these can be considered *intrinsically idempotent* in the sense that any function which carries out the transformation is necessarily an idempotent operator. An idempotent operation is one which can be applied iteratively with no effect beyond the first application. For instance, a sorting transformation is intrinsically idempotent since sorting an already sorted data structure is redundant and such a sorting is often deterministic. On the other hand, image transformations might not always be idempotent: rotating an image by 90° twice is not the same as rotating it once, for instance. As such, some transformations are intrinsically idempotent and admit *only* idempotent solutions while other tasks are not and admit *no* idempotent solutions. This work, however, is concerned with a class of data transformation tasks which has both idempotent and non-idempotent solutions and where idempotency might be a desirable property. For example, in section 3 we study idempotency in generative networks where it is the formal requirement of one-step inference, but also denoising and image augmentation (e.g., application of effect-filters) are examples of tasks where idempotent solutions may be desirable [8, 5]. Since solutions are not intrinsically idempotent in this class, we explore actively enforcing idempotency as a component of the loss function used in training.

Preprint. Under review.

In this paper we are primarily concerned with networks $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^n$, where $\boldsymbol{\theta}$ is a collection of weight parameters. The condition that f_θ is idempotent is

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = f_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\mathbf{x})) \tag{1}$$

for all $\mathbf{x} \in \mathbb{R}^n$. If $f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{W}\mathbf{x}$ (a single-layer, fully-connected network with no bias and the identity activation function) where $\mathbf{W} \in \mathbb{R}^{n \times n}$ is the weight matrix, then condition 1 reduces to the familiar notion from linear algebra where $\mathbf{W} = \mathbf{W}^2$ and eigenvalues of $\mathbf{W}$ are either 0 or 1. Condition 1 also gives the correct notion for non-linear networks acting as idempotent projections, and can be optimized using a simple mean-squared error loss,

$$\mathcal{L}_{\text{idem}}(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} \left(f_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\mathbf{x})) - f_{\boldsymbol{\theta}}\big(\mathbf{x}\big) \right)^2, \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^n$. As we show in section 3, minimizing this loss using canonical gradient descent can yield relatively poor improvement in the idempotent loss. Additionally, due to the higher-order application of $f_{\boldsymbol{\theta}}$ the number of terms in the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{idem}}$ grows exponentially in the number of layers, making the approach computationally expensive for certain architectures.

In this work, we propose an alternative method for training neural networks to satisfy condition 1. Using ideas from Perturbation Theory [4] we derive a function g which solves $\mathbf{M}' = g(\mathbf{M})$ such that if $\mathbf{M} \in \mathbb{R}^{n \times n}$ is an "almost" idempotent matrix, then $\mathbf{M}' \in \mathbb{R}^{n \times n}$ is perfectly idempotent (i.e., $(\mathbf{M}')^2 = \mathbf{M}'$). In this work, we focus on one such function:

$$g(\mathbf{M}) = 3\mathbf{M}^2 - 2\mathbf{M}^3 \tag{3}$$

Although we assume $\mathbf{M}$ is close to idempotent, we show that in practice g can be used to derive matrices which are within machine precision of perfect idempotency even when the input matrix $\mathbf{M}$ is relatively far from idempotent. At a high level, this process is based on a recurrence relation $\mathbf{M}' \leftarrow (1-\gamma)\mathbf{M} + \gamma g(\mathbf{M})$, taking small γ-sized steps in the direction of $g(\mathbf{M})$. While this recurrence relation derives idempotent matrices – and can therefore be used to train single-layer networks with identity activations to be idempotent – we also give a more general application of Eq. 3 as a minor modification of the backpropagation algorithm, yielding an architecture agnostic and efficient algorithm for finding idempotent networks.

In section 2 we give a detailed description of the method used to derive Eq. 3 and alternative solutions. We also show that while there exists non-idempotent fixed points to Eq. 3, these points are repelling under the recurrence relation $\mathbf{M}' \leftarrow (1-\gamma)\mathbf{M} + \gamma g(\mathbf{M})$ for $0 \leq \gamma < 1$, giving credence to the use of such a recurrence relation in practice. Finally, in section 2 we derive a full training scheme for training arbitrary neural network architectures of the form $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^n$. In section 3, we present experimental data for a variety of fully-connected network architectures, showing that our method outperforms ordinary backpropagation under ideal conditions. We also replicate the results of Shocher et al. in [10] by applying our method on a U-net style DCGAN network to successfully create a generative network for MNIST and CelebA datasets. Lastly, sections 4 and 5 discuss how our method distinguishes itself from related approaches as well as practical limitations of our method.

2 Method

2.1 An idea from Perturbation Theory

Perturbation Theory comprises methods for finding an approximate solution to a problem by starting from the exact solution of a related, simpler problem and adding successive "perturbations" to the system. It is a diverse set of tools used to reason about complex dynamical systems often used in physics and quantum chemistry [2]. We refer the reader to Bla et el. [4] for a detailed treatment of the topic.

Definition 1 (Near-idempotent to order n) *Let the matrix $\mathbf{P} \in \mathbb{R}^{m \times m}$ satisfy $\mathbf{P} = \mathbf{P}^2$. Let $\mathbf{D} \in \mathbb{R}^{m \times m}$ be arbitrary (e.g., noise) where there exists some $n \in \mathbb{N}$ such that $\mathbf{D}^{n+1}$ has coefficients with absolute value below $\epsilon \ll 1$. We then say that $\mathbf{K} = \mathbf{P} + \mathbf{D}$ is **near-idempotent to order** n.*

Using definition 1 we may define the following ansatz in terms of the *near-idempotent* $\mathbf{K}$:

$$\mathbf{K}' = \alpha_1\mathbf{K} + \alpha_2\mathbf{K}^2 + \cdots + \alpha_j\mathbf{K}^j. \tag{4}$$

This poses $\mathbf{K}'$ as the linear combination of higher orders of near-idempotent matrices. If we further constrain $(\mathbf{K}')^2 - \mathbf{K}' = \mathbf{0}$, the result is a linear program in variables α_i. Importantly, for all equations of the program, any term in which $\mathbf{D}$ appears at least $n+1$ times can be considered "negligible" and ignored. This simplification vastly reduces the problem and allows approximate solutions to the linear program. The coefficients α_i can be thought of as parameterizing a projection g such that $\mathbf{K}' = g(\mathbf{K})$ for an arbitrary near-idempotent $\mathbf{K}$. The requirement that $\mathbf{K}'$ be idempotent and that $\mathbf{K}$ is only near-idempotent implies that a solution g is a projection onto the manifold of idempotent matrices; we call g an **idempotent corrector** as it must "make $\mathbf{K}$ idempotent".

Note that definition 1 places no restrictions on the distribution from which $\mathbf{D}$ is drawn, hence $\mathbf{K}$ and the underlying $\mathbf{P}$ have no meaningful relation. Similarly, the linear program above also places no assumptions on the relationship between $\mathbf{K}'$ and $\mathbf{K}$.

In the case when $n = 1$ we consider $\mathbf{D}^2 \approx \mathbf{0}$ and the expression $(\mathbf{K}')^2 - \mathbf{K}' = \mathbf{0}$ can be expanded and reduced by recursively applying the assumptions

$$\mathbf{D}^2 \approx \mathbf{0}, \quad \mathbf{P}^2 = \mathbf{P}, \quad \mathbf{XDYDZ} \approx \mathbf{0} \text{ for all } \mathbf{X}, \mathbf{Y}, \mathbf{Z}$$

In the case when $j = 2$, i.e., the ansatz 4 is $\mathbf{K}' = \alpha_1\mathbf{K} + \alpha_2\mathbf{K}^2$, there exists no solutions for α_i. When $j = 3$, however, there is exactly one solution when $\alpha_1 = 0$, $\alpha_2 = 3$ and $\alpha_3 = -2$, hence g is

$$g(\mathbf{K}) = 3\mathbf{K}^2 - 2\mathbf{K}^3 \tag{5}$$

For $j > 3$ there exists families of solutions (see Appendix A), but we consider primarily the case when $j = 3$ as this requires fewer higher-order terms of $\mathbf{K}$.

2.2 Fixed Points and Stability Analysis

Undoubtedly, a required property of any idempotent corrector g is that every idempotent matrix is a fixed point, but it may also be desirable to find if any non-idempotent matrices are fixed points. Concretely, we wish to characterize solutions to $\mathbf{K} = 3\mathbf{K}^2 - 2\mathbf{K}^3$.

In general, we place no restrictions on the matrix $\mathbf{K} \in \mathbb{R}^{m\times m}$, hence it might not be diagonalizable directly. It is well known, however, that for every square matrix $\mathbf{K}$ there exists an invertible matrix $\mathbf{P}$ such that

$$\mathbf{K} = \mathbf{PJP}^{-1}$$

where $\mathbf{J} \in \mathbb{C}^{m\times m}$ is the Jordan Normal form [1] of $\mathbf{K} \in \mathbb{R}^{m\times m}$. From this the dual problem

$$\mathbf{J} = 3\mathbf{J}^2 - 2\mathbf{J}^3$$

can be constructed. The block-diagonal structure of $\mathbf{J}$ imposes up to four equations per eigenvalue of $\mathbf{K}$ (see Appendix B):

$$\lambda = 3\lambda^2 - 2\lambda^3 \tag{6}$$

$$1 = 6\lambda - 6\lambda^2 \quad \text{Only when } d_\lambda \geq 2. \tag{7}$$

$$0 = 3 - 6\lambda \quad \text{Only when } d_\lambda \geq 3. \tag{8}$$

$$0 = 0 - 2 \quad \text{Only when } d_\lambda \geq 4. \tag{9}$$

where d_λ denote the algebraic multiplicity of eigenvalue λ of $\mathbf{K}$. Clearly, this system of equations is inconsistent when $d_\lambda \geq 2$, hence geometric multiplicity of every eigenvalue must also be exactly 1. Together this implies that $\mathbf{J}$ is diagonalizable for any fixed point $\mathbf{K}$. Furthermore, the solutions which satisfy only eq. 6 are

$$\lambda \in \{0, 0.5, 1\}$$

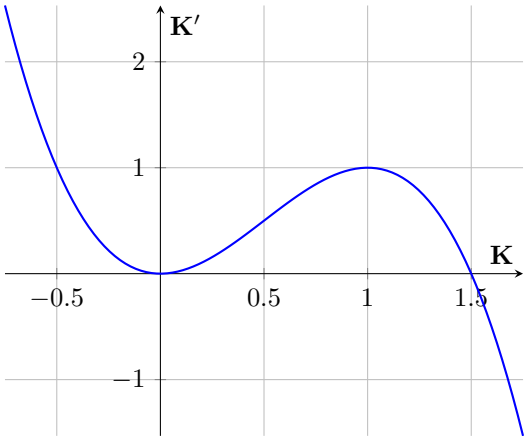


Figure 1: Plot of $\mathbf{K}' = 3\mathbf{K}^2 - 2\mathbf{K}^3$ in the case $\mathbf{K}$ is scalar.

hence any fixed point of $\mathbf{K} = 3\mathbf{K}^2 - 2\mathbf{K}^3$ must have eigenvalues in this set. Consequently, all idempotent matrices are fixed points, but there exists also non-idempotent fixed points.

Although the initial derivation of $g(\mathbf{K}) = 3\mathbf{K}^2 - 2\mathbf{K}^3$ relies on $\mathbf{K}$ being near-idempotent to the first order, we consider more generally the behaviour of g around the fixed points when applied repeatedly as a recurrence relation. Note first that $h(\lambda) = 3\lambda^2 - 2\lambda^3$ has derivative $h'(\lambda) = 6\lambda - 6\lambda^2$, and so for each fixed point of g we have

$$h'(0) = 0, \quad h'(0.5) = 1.5, \quad h'(1) = 0.$$

Since $|h'(\lambda)| < 1$ for $\lambda \in \{0, 1\}$ these points are attracting whilst $|h'(\lambda)| > 1$ for $\lambda = 0.5$, hence this point is repelling. In other words, if the idempotent corrector g applied as a recurrence relation to $\mathbf{M}$ converges at some point $\mathbf{M}'$, then $\mathbf{M}'$ will be approximately idempotent unless $\mathbf{M}$ has an eigenvalue of exactly 0.5.

Furthermore, figure 2 shows the result of applying the idempotent corrector recursively 10 times for each point on the complex plane. The attracting regions around 0 and 1 are large, hence any matrix that is "reasonably close" to idempotent will be projected onto a (within machine precision) idempotent matrix.

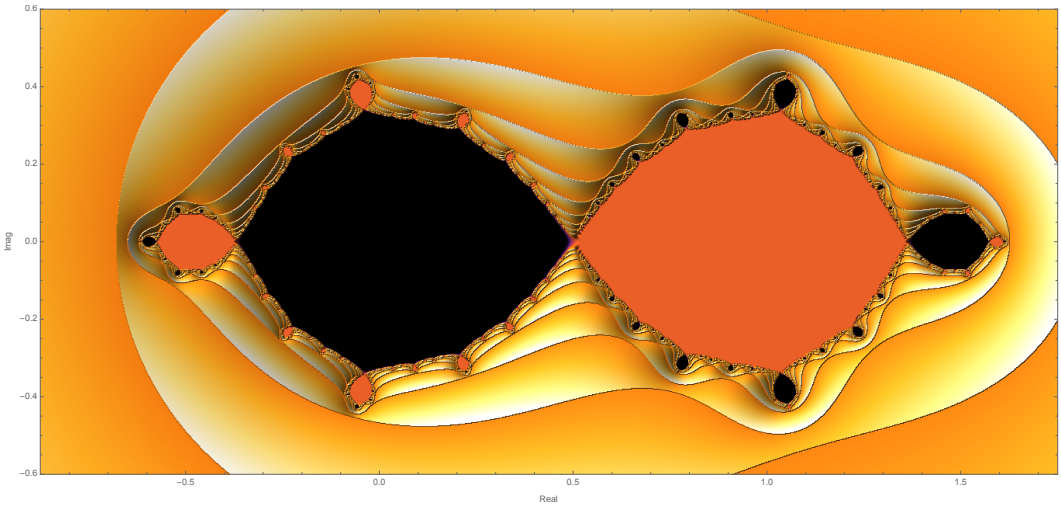


Figure 2: 10-time recursive application of $h(\lambda) = 3\lambda^2 - 2\lambda^3$ for each point on the complex plane. Black areas denote points converging onto 0, while orange areas denote points converging onto 1.

2.3 Deriving a Training Scheme

Gradient-based optimization techniques use the gradient of a non-convex loss function as the directional information used to update the hypothesis at each time step. This highlights a core difference between our approach and conventional gradient-based approaches, since the recurrence relation derived above (and shown in figure 1) exactly describes the "direction" to move in to reduce idempotent error. Out method only evaluates g – finding its derivative is unimportant.

Consider a neural network $f_{\boldsymbol{\theta}} : \mathbb{R}^m \to \mathbb{R}^m$ together with its application to input $\mathbf{x} \in \mathbb{R}^m$, denoted $\mathbf{y} = f_{\boldsymbol{\theta}}(\mathbf{x})$. We might then consider the recurrence relation above in the following form:

$$\mathbf{y}' = 3f_{\boldsymbol{\theta}}(\mathbf{y}) - 2f_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\mathbf{y}))$$

This describes a desired change in the output of the network which we denote $\Delta f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{y}' - \mathbf{y}$. In other words, $\Delta f_{\boldsymbol{\theta}}(\mathbf{x})$ describes the desired change in $\mathbf{y}$ which moves $\mathbf{y}$ towards an idempotent projection much in the same way that the quantity $\frac{\partial(-\mathcal{L}(\mathbf{y}))}{\partial \mathbf{y}}$ describes the direction which reduces the idempotent loss function $\mathcal{L}(\mathbf{y})$ in eq. 2. In this work, we therefore define

$$\frac{\partial(-\mathcal{L}(\mathbf{y}))}{\partial \mathbf{y}} \equiv \Delta f_{\boldsymbol{\theta}}(\mathbf{x}) \tag{10}$$

as an alternative quantity to the traditional, analytical solution to $\frac{\partial(-\mathcal{L}(\mathbf{y}))}{\partial \mathbf{y}}$.

To complete the scheme, we consider how a change in the output $\mathbf{y}$ can be propagated to a change in the parameters $\boldsymbol{\theta}$ of $f_{\boldsymbol{\theta}}$. This, however, is a straightforward application of the chain rule as it is calculated conventionally in backpropagation.

In practice, the definition 10 can be implemented in common machine learning frameworks, such as Jax[1] and PyTorch[2] as a user-defined automatic differentiation rule (see Appendix C).

3 Experimental Results

To evaluate the training scheme suggested in section 2 we compare relative performance between the two methods: "Ordinary Backpropagation" with the quantity $\frac{\partial(-\mathcal{L}(\mathbf{y}))}{\partial \mathbf{y}}$ resolved at runtime by automatic differentiation, and "Modified Backpropagation" with the modified backpropagation rule for $\frac{\partial(-\mathcal{L}(\mathbf{y}))}{\partial \mathbf{y}}$. To demonstrate the flexibility of the approach, we report results for four diverse MLP-style networks, as described in table 1.

Identifier	Architecture	No. Parameters
B1	Linear(5,5)	30
B2	Linear(100, 250) Linear(250, 250) Linear(250, 250) Linear(250, 250) Linear(250, 100)	238 600
B3	Linear(4096, 1024) Linear(1024, 4096)	8 393 728
B4	Linear(784, 1024) Linear(1024, 2048) Linear(2048, 784)	4 509 456

Table 1: Four neural networks for testing. Each "Linear(n, m)" block is parameterized by its input dimension n and its output dimension m, corresponding to the underlying $\mathbf{W} \in \mathbb{R}^{n \times m}$ weight matrix. Every block has an associated bias vector and LeakyReLU(0.2) activation function. B1 represents a trivial network, B2 represents a relatively deep network, B3 represents a relatively wide network, and B4 represents a practically useful network.

The dataset used for training in this section is drawn from a normal distribution with mean 0 and standard deviation 1. To prevent concerns about overfitting, the distribution is sampled i.i.d. at each epoch during training. Furthermore, a batch size of 1000 is used, although comparable results have been found using batch sizes between 32 and $10\,000$.

[1] https://github.com/jax-ml/jax

[2] https://pytorch.org

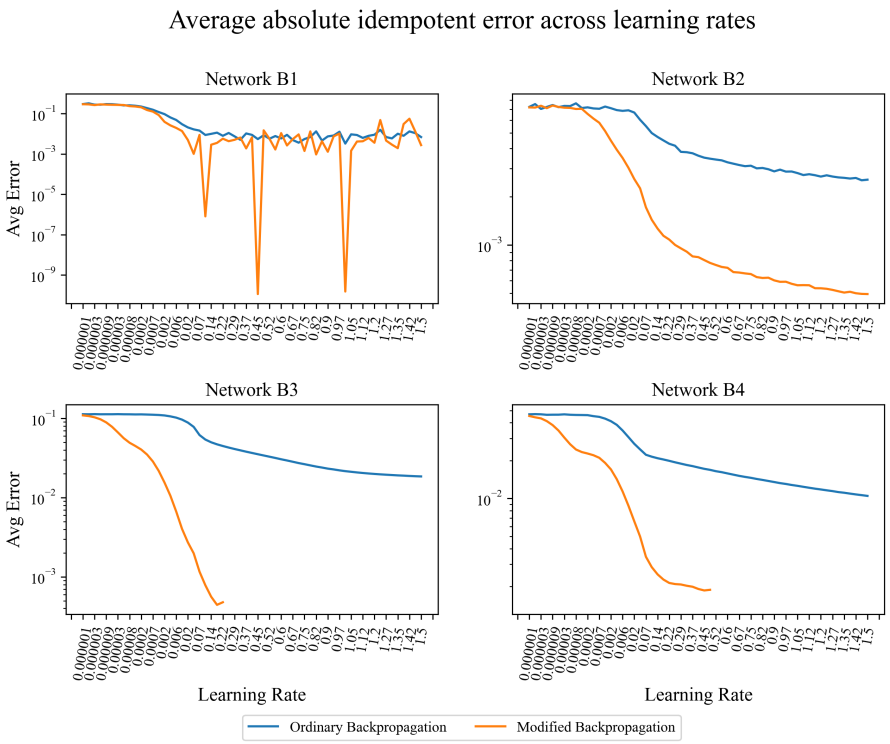


Figure 3: Average of 10 runs of each algorithm for a variety of learning rates. Networks are randomly initialized and trained for 2 500 epochs. Runs which did not return a solution with lower idempotent error than the initial value are discarded, and the average is over remaining runs. For networks B3 and B4, learning rates > 0.22 and > 0.52 respectively had no runs with improvement in error. For "Modified Backpropagation" on B1, some runs resulted in approximately $\mathbf{0}$ which, due to floating-point imprecision, results in the error spikes.

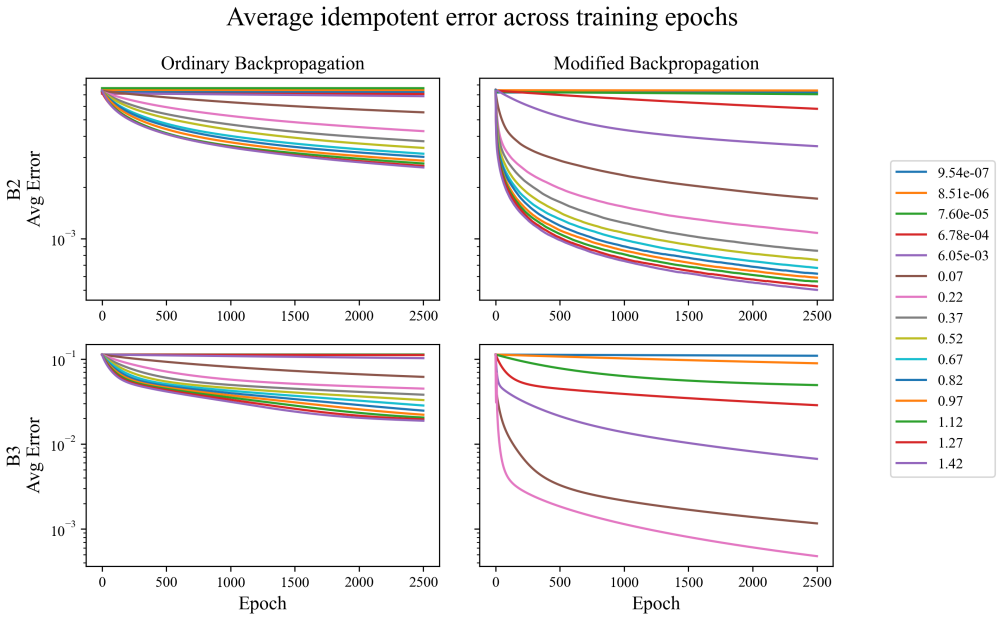
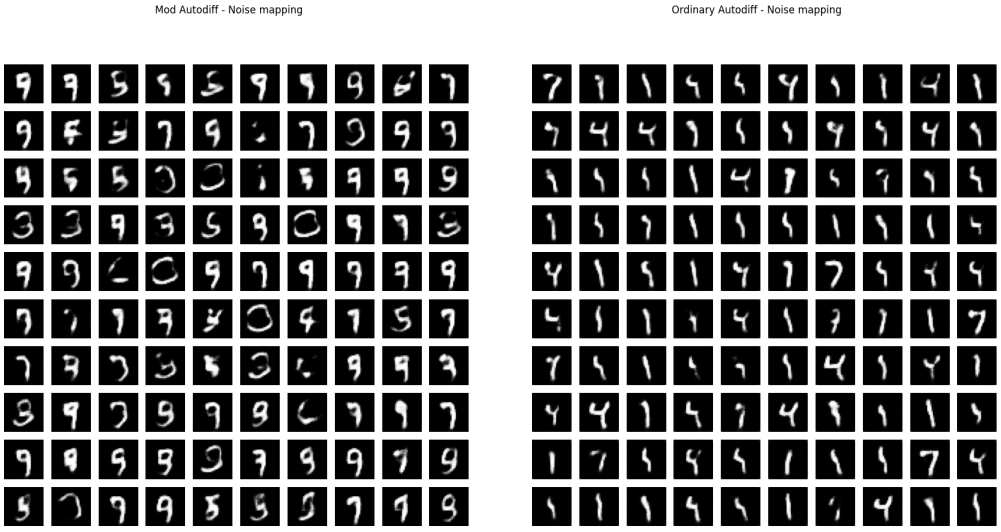


Figure 4: On networks B2 and B3, the average idempotent error across 10 runs for each learning rate is reported for each algorithm. Each column of graphs represents one algorithm. "Modified Backpropagation" achieves lower idempotent error at lower learning rates than "Ordinary Backpropagation".

TODO: Include a section on qualitative differences in behaviour of modified and ordinary backpropagation. Does one method travel a different route in the loss landscape? This small section is to be suggestive of distinguishing features of the new method.

3.1 Application to Generative Networks



(a) Modified Backpropagation (b) Ordinary Backpropagation

Figure 5: U-DCGAN architecture trained on MNIST.

TODO: Trim the above graph, no need to show 100 samples for each model.

TODO: Include a similar graph as the above for CelebA dataset.

TODO: Produce graph for latent-space analysis and write comments.

4 Related Work

Review Idempotent Generative Networks, contrasting our work on a gradient-free approach. Have others applied perturbation theory to ML? We suffer same problems as GANs: mode collapse.

5 Conclusion

Give a conclusion of central idea: the use of perturbation analysis to find an iterator which we have successfully applied to a range of toy examples and GAN scenarios.

References

[1] Steven H. Weintraub. *Jordan Canonical Form. Theory and Practice*. Springer, 2009.

[2] Joseph O. Hirschfelder, W. Byers Brown, and Saul T. Epstein. "Recent Developments in Perturbation Theory". In: ed. by Per-Olov Löwdin. Vol. 1. Advances in Quantum Chemistry. Academic Press, 1964, pp. 255–374. DOI: https://doi.org/10.1016/S0065-3276(08)60381-0. URL: https://www.sciencedirect.com/science/article/pii/S0065327608603810.

[3] Ademola E. Ilesanmi and Taiwo O. Ilesanmi. "Methods for image denoising using convolutional neural network: a review". In: *Complex & Intelligent Systems* 7.5 (Oct. 2021), pp. 2179–2198. ISSN: 2198-6053. DOI: 10.1007/s40747-021-00428-4. URL: https://doi.org/10.1007/s40747-021-00428-4.

[4] Tosio Kato. *Perturbation Theory for Linear Operators*. Springer, 1995.

[5] Ming-Yu Liu et al. "Generative Adversarial Networks for Image and Video Synthesis: Algorithms and Applications". In: *Proceedings of the IEEE* 109.5 (2021), pp. 839–862. DOI: 10.1109/JPROC.2021.3049196.

[6] Yuzhen Lu et al. "Generative adversarial networks (GANs) for image augmentation in agriculture: A systematic review". In: *Computers and Electronics in Agriculture* 200 (2022), p. 107208. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2022.107208. URL: https://www.sciencedirect.com/science/article/pii/S0168169922005233.

[7] Siwei Ma et al. "Image and Video Compression With Neural Networks: A Review". In: *IEEE Transactions on Circuits and Systems for Video Technology* 30.6 (2020), pp. 1683–1698. DOI: 10.1109/TCSVT.2019.2910119.

[8] Yuxin Mao et al. "Deep Idempotent Network for Efficient Single Image Blind Deblurring". In: *IEEE Transactions on Circuits and Systems for Video Technology* 33.1 (2023), pp. 172–185. DOI: 10.1109/TCSVT.2022.3202361.

[9] A. Namphol, S.H. Chin, and M. Arozullah. "Image compression with a hierarchical neural network". In: *IEEE Transactions on Aerospace and Electronic Systems* 32.1 (1996), pp. 326–338. DOI: 10.1109/7.481272.

[10] Assaf Shocher et al. *Idempotent Generative Network*. 2023. arXiv: 2311.01462 [cs.CV]. URL: https://arxiv.org/abs/2311.01462.

[11] T. Tambouratzis. "A novel artificial neural network for sorting". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29.2 (1999), pp. 271–275. DOI: 10.1109/3477.752799.

A Solutions to the ansatz

Show up to some order (approx 10?) the families of solutions to the ansatz when $\mathbf{K}$ *is near-idempotent to the first order. Also give a concrete example (like in your slides) for how we actually compute the corrector.*

B Jordan normal form analysis

Give the full Jordan Normal Form analysis.

C Automatic differentiation rule

Describe the implementation of the autodiff rule in PyTorch. Show the code and the computational graph highlighting the differences between ordinary and modified backpropagation.