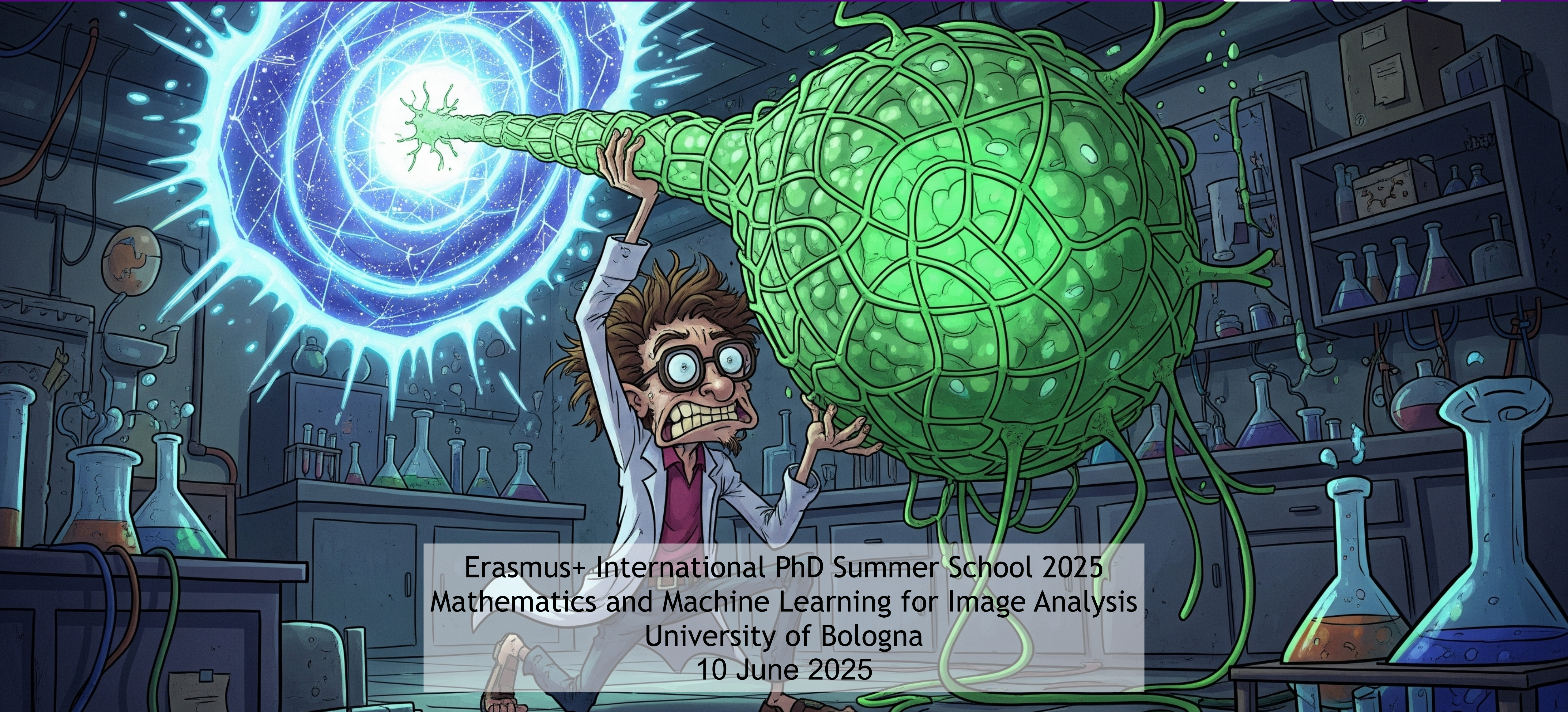
Faculty of Engineering, Department of Computer Science
UCL



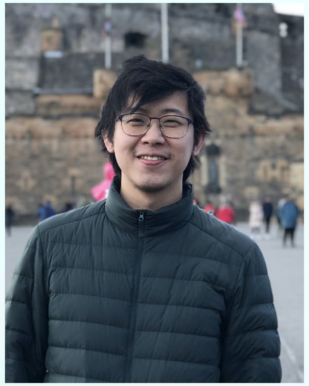
Erasmus+ International PhD Summer School 2025
Mathematics and Machine Learning for Image Analysis
University of Bologna
10 June 2025

Beyond backpropagation

Part I
- Lifted Bregman training of neural networks

Part II
- Regularised inversion of neural networks
- Inversion with theoretical guarantees?
- Conclusions & outlook

Joint work with


Xiaoyu (Victor) Wang
Heriot-Watt University


Audrey Repetti
Heriot-Watt University


Andreas Mang
University of Houston


Alexandra Valavanis
Queen Mary University of London


Azhir Mahmood
University College London

Acknowledgements:

The Alan Turing Institute

CCIMI
CANTAB CAPITAL INSTITUTE FOR THE MATHEMATICS OF INFORMATION

Queen Mary University of London
Digital Environment Research Institute

Open access papers available

JMLR 24(232) 2023 (Training)

Front. Appl. Math. Stat. 9 2013 (Inversion)

Part I: Lifted Bregman training of neural networks

Training neural networks

An L-layer (deep) neural network is a composition of activation functions σ and affine-linear transformations applied to inputs x, to produce outputs y, i.e.

$$y = \mathcal{N}(x, \Theta) = \sigma\left(W_L \left(\cdots \sigma\left(W_1 x + b_1 \right) \cdots \right) + b_L \right) ,$$

with parameters $\Theta = \left\{ (W_l, b_l) \right\}_{l=1}^{L}$

Training neural networks

An L-layer (deep) neural network is a composition of activation functions σ and affine-linear transformations applied to inputs x, to produce outputs y, i.e.

$$y = \mathcal{N}(x, \Theta) = \sigma\left(W_L\left(\cdots\sigma\left(W_1 x + b_1\right)\cdots\right) + b_L\right),$$

with parameters $\Theta = \left\{(W_l, b_l)\right\}_{l=1}^{L}$

Training usually boils down to the (approximate) minimisation of empirical risks of the form

$$E(\Theta) = \frac{1}{s}\sum_{i=1}^{s} \ell\left(y_i, \mathcal{N}(x_i, \Theta)\right)$$

Training neural networks

An L-layer (deep) neural network is a composition of activation functions σ and affine-linear transformations applied to inputs x, to produce outputs y, i.e.

$$y = \mathcal{N}(x, \Theta) = \sigma\left(W_L\left(\cdots\sigma\left(W_1 x + b_1\right)\cdots\right) + b_L\right),$$

with parameters $\Theta = \left\{(W_l, b_l)\right\}_{l=1}^{L}$

Training usually boils down to the (approximate) minimisation of empirical risks of the form

Example: $$E(\Theta) = \frac{1}{2s}\sum_{i=1}^{s} \left\| y_i - \mathcal{N}(x_i, \Theta) \right\|^2$$ Mean-Squared Error (MSE)

Training neural networks

An L-layer (deep) neural network is a composition of activation functions σ and affine-linear transformations applied to inputs x, to produce outputs y, i.e.

$$y = \mathcal{N}(x, \Theta) = \sigma\left(W_L\left(\cdots\sigma\left(W_1 x + b_1\right)\cdots\right) + b_L\right),$$

with parameters $\Theta = \left\{(W_l, b_l)\right\}_{l=1}^{L}$

Training usually boils down to the (approximate) minimisation of empirical risks of the form

Example: $$E(\Theta) = \frac{1}{2s}\sum_{i=1}^{s} \left\| y_i - \mathcal{N}(x_i, \Theta) \right\|^2$$ Mean-Squared Error (MSE)

Training neural networks

Training usually boils down to the (approximate) minimisation of empirical risks of the form

$$E(\Theta) = \frac{1}{2s} \sum_{i=1}^{s} \| y_i - \mathcal{N}(x_i, \Theta) \|^2$$

In the majority of cases, this approximate minimisation is carried out by a combination of

Gradient-based optimisation algorithm

Gradient computation via backpropagation

Training neural networks

Training usually boils down to the (approximate) minimisation of empirical risks of the form

$$E(\Theta) = \frac{1}{2s} \sum_{i=1}^{s} \left\| y_i - \mathcal{N}(x_i, \Theta) \right\|^2$$

Example: gradient descent

$$\Theta^{k+1} = \Theta^k - \frac{\tau^k}{s} \sum_{i=1}^{s} \underbrace{\left(J^{\Theta}_{\mathcal{N}(x_i, \cdot)} (\Theta^k) \right)}_{\text{Jacobian of } \mathcal{N}(x_i,\Theta) \text{ w.r.t } \Theta}{}^{\top} \left(\mathcal{N}(x_i, \Theta^k) - y_i \right)$$

Training neural networks

Training usually boils down to the (approximate) minimisation of empirical risks of the form

$$E(\Theta) = \frac{1}{2s} \sum_{i=1}^{s} \left\| y_i - \mathcal{N}(x_i, \Theta) \right\|^2$$

Example: gradient descent

$$\Theta^{k+1} = \Theta^k - \frac{\tau^k}{s} \sum_{i=1}^{s} \underbrace{\left(J^{\Theta}_{\mathcal{N}(x_i,\cdot)}(\Theta^k) \right)^\top \left(\underbrace{\mathcal{N}(x_i, \Theta^k)}_{\text{Forward pass}} - y_i \right)}_{\text{Backward pass}}$$

Training neural networks

Potential drawbacks of previous approach:

- Backpropagation algorithm is serial in nature
- Differentiation of activation function σ is required

Alternative:

- Lifting of parameter space to aid distributed computation
- Using novel class of loss functions to avoid differentiation of σ

Lifted training of neural networks

Lifted training: rewrite

$$E(\Theta) = \frac{1}{2s}\sum_{i=1}^{s} \left\| y_i - \mathcal{N}(x_i, \Theta) \right\|^2$$

as

$$E(\Theta) = \frac{1}{2s}\sum_{i=1}^{s} \left\| y_i - x_i^L \right\|^2$$

subject to $x_i^l = \sigma(W_l x_i^{l-1} + b_l)$ for all $l \in \{1, \dots, L\}$

and $x_i^0 = x_i$

Lifted training of neural networks

Lifted training: replace

$$E(\Theta) = \frac{1}{2s} \sum_{i=1}^{s} \left\| y_i - \mathcal{N}(x_i, \Theta) \right\|^2$$

with

$$E(\Theta, X) = \frac{1}{2s} \sum_{i=1}^{s} \sum_{l=1}^{L} \left\| x_i^l - \sigma(W_l x_i^{l-1} + b_l) \right\|^2$$

where $x_i^0 = x_i$ and $x_i^L = y_i$.

The notation X is short-hand for $X = \{x_i^l\}_{i,l=1}^{s,L-1}$

- Miguel Carreira-Perpinan and Weiran Wang. Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, pages 10–19, 2014.
- Askari, Armin, Geoffrey Negiar, Rajiv Sambharya, and Laurent El Ghaoui. "Lifted neural networks." *arXiv preprint arXiv:1805.01532* (2018).

Lifted Bregman training of neural networks

Lifted Bregman training: replace

$$E(\Theta) = \frac{1}{2s} \sum_{i=1}^{s} \left\| y_i - \mathcal{N}(x_i, \Theta) \right\|^2$$

with

$$E(\Theta, X) = \frac{1}{2s} \sum_{i=1}^{s} \sum_{l=1}^{L} B_\Psi \left(x_i^l, W_l x_i^{l-1} + b_l \right)$$

with Bregman / Fenchel loss / penalty function

$$B_\Psi(y, z) = \frac{1}{2}\|y\|^2 + \Psi(y) + \left(\frac{1}{2}\| \cdot \|^2 + \Psi \right)^* (z) - \langle y, z \rangle$$

Xiaoyu Wang, MB. Lifted Bregman Training of neural networks. *JMLR* 24(232):1—51, 2023.

Lifted Bregman training of neural networks

Lifted Bregman training:

$$E(\Theta, X) = \frac{1}{2s} \sum_{i=1}^{s} \sum_{l=1}^{L} B_\Psi\left(x_i^l, W_l x_i^{l-1} + b_l\right)$$

with Bregman / Fenchel loss / penalty function

$$B_\Psi(y, z) = \frac{1}{2}\|y\|^2 + \Psi(y) + \left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)^*(z) - \langle y, z\rangle$$

What is Ψ?
And why would we replace the squared Euclidean norm with such a function?

Xiaoyu Wang, MB. Lifted Bregman Training of neural networks. *JMLR* 24(232):1—51, 2023.

Lifted Bregman training of neural networks

Suppose we have samples (x, y) and want to find W, b such that

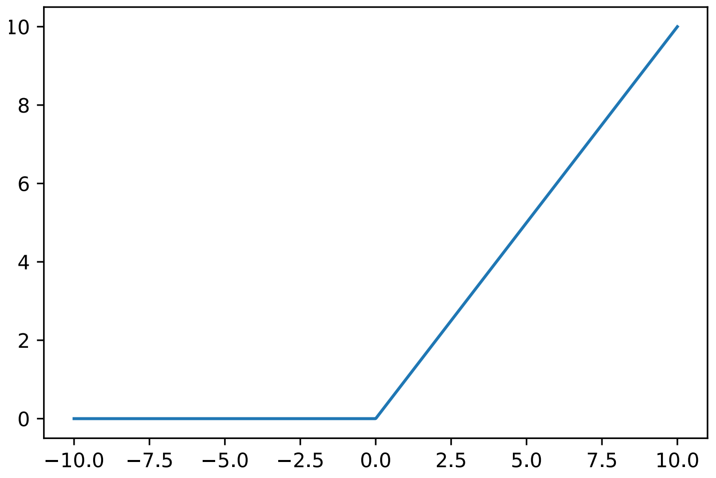
$$y = \sigma(Wx + b)$$

Here σ denotes the (usually nonlinear) activation function of the perceptron

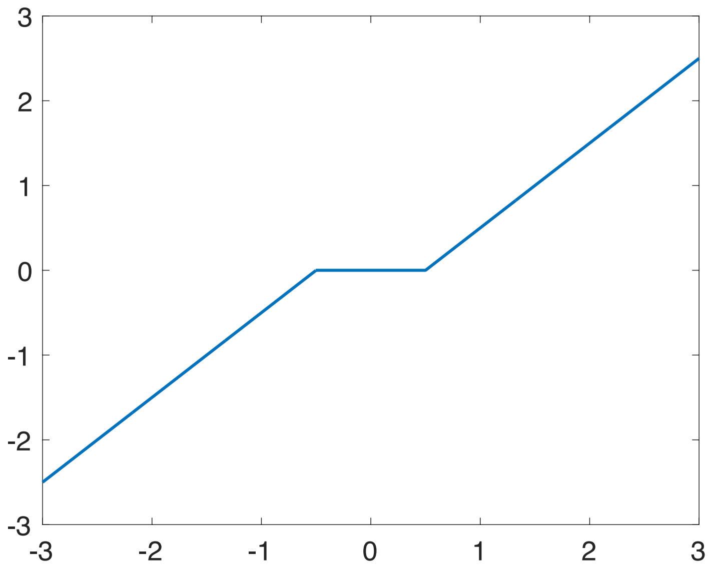
What activation functions do we allow?

Lifted Bregman training of neural networks

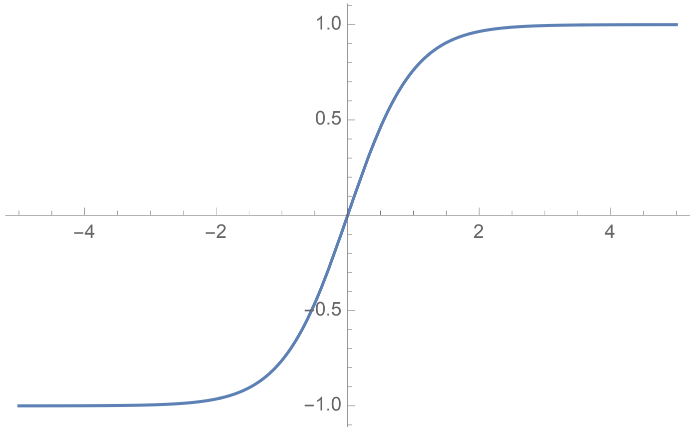
Suppose we choose common activation functions for our neural network, such as



rectifier / ramp



soft-thresholding



hyperbolic tangent

What do all these functions have in common?

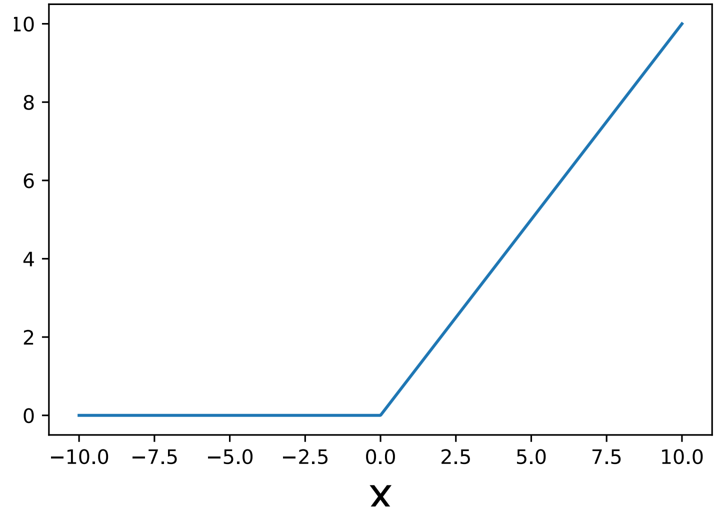
Lifted Bregman training of neural networks

All previous activation functions are *proximal maps*:

$$\sigma(z) = \mathrm{prox}_{\Psi}(z) := \arg\min_{u\in\mathbb{R}^n} \left\{ \frac{1}{2}\|u - z\|^2 + \Psi(u) \right\}$$

for some proper, convex and lower semi-continuous function $\Psi : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$

Example: $\Psi(u) = \begin{cases} 0 & u \geq 0 \\ \infty & u < 0 \end{cases} \quad \Longrightarrow \quad \sigma(z) = \max(0, z)$



Moreau, Jean Jacques. "Fonctions convexes duales et points proximaux dans un espace hilbertien." (1962).

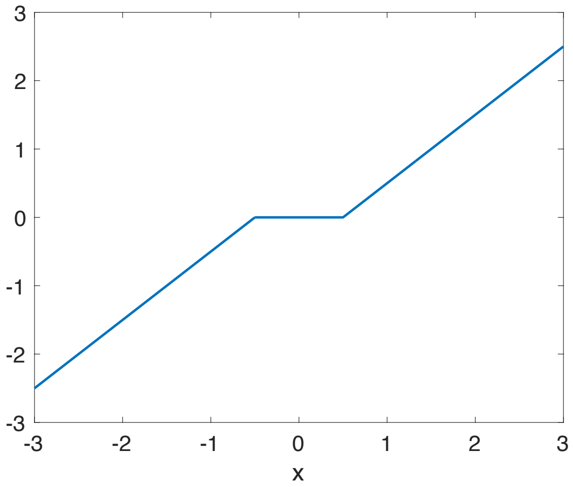
Lifted Bregman training of neural networks

All previous activation functions are *proximal maps*:

$$\sigma(z) = \text{prox}_{\Psi}(z) := \arg\min_{u \in \mathbb{R}^n} \left\{ \frac{1}{2}\|u - z\|^2 + \Psi(u) \right\}$$

for some proper, convex and lower semi-continuous function $\Psi : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$

Example: $\Psi(u) = \alpha|u|$ $\quad\Longrightarrow\quad \sigma(z) = \begin{cases} z - \alpha & z > \alpha \\ 0 & |z| \leq \alpha \\ z + \alpha & z < -\alpha \end{cases}$



Moreau, Jean Jacques. "Fonctions convexes duales et points proximaux dans un espace hilbertien." (1962).

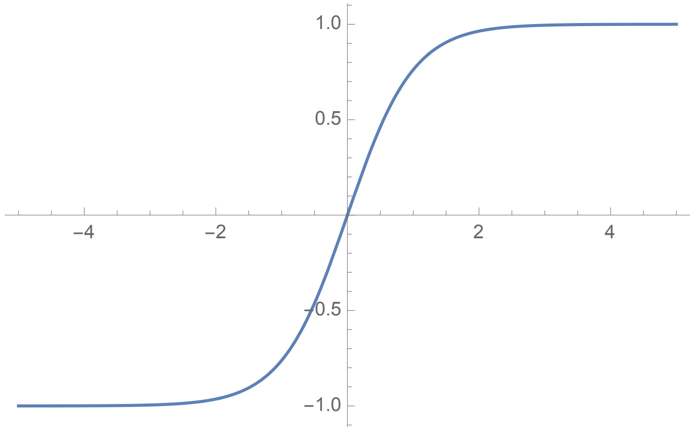
Lifted Bregman training of neural networks

All previous activation functions are *proximal maps*:

$$\sigma(z) = \text{prox}_{\Psi}(z) := \arg\min_{u \in \mathbb{R}^n} \left\{ \frac{1}{2}\|u - z\|^2 + \Psi(u) \right\}$$

for some proper, convex and lower semi-continuous function $\Psi : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$

Example: $\Psi(u) = \begin{cases} u \tanh^{-1}(u) + \frac{1}{2}\left(\log(1 - u^2) - u^2\right) & |u| < 1 \\ \infty & \text{otherwise} \end{cases}$



$$\Longrightarrow \qquad \sigma(z) = \tanh(z)$$

Combettes, Patrick L., and Jean-Christophe Pesquet. "Deep neural network structures solving variational inequalities." *Set-Valued and Variational Analysis* (2020): 1-28.

Lifted Bregman training of neural networks

All previous activation functions are *proximal maps*:

$$\sigma(z) = \text{prox}_{\Psi}(z) := \arg\min_{u \in \mathbb{R}^n} \left\{ \frac{1}{2}\|u - z\|^2 + \Psi(u) \right\}$$

for some proper, convex and lower semi-continuous function $\Psi : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$

Lots of works focus on proximal maps as activation functions, e.g.

Hasannasab, M., Hertrich, J., Neumayer, S., Plonka, G., Setzer, S., & Steidl, G. (2020). Parseval proximal neural networks. Journal of Fourier Analysis and Applications, 26, 1-31.
Combettes, Patrick L., and Jean-Christophe Pesquet. "Deep neural network structures solving variational inequalities." *Set-Valued and Variational Analysis* (2020): 1-28.
Hertrich, J., Neumayer, S., & Steidl, G. (2021). Convolutional proximal neural networks and plug-and-play algorithms. *Linear Algebra and its Applications*, *631*, 203-234.
Le, H. T. V., Repetti, A., & Pustelnik, N. (2023). PNN: From proximal algorithms to robust unfolded image denoising networks and Plug-and-Play methods. *arXiv preprint arXiv:2308.03139*.

and many many more…

Lifted Bregman training of neural networks

Suppose we have samples (x, y) and want to find W, b such that

$$y = \sigma(Wx + b) = \text{prox}_\Psi(Wx + b)$$

$$\iff \quad y = \arg\min_z \left\{ \frac{1}{2}\|z - (Wx + b)\|^2 + \Psi(z) \right\}$$

$$\iff \quad Wx + b - y \in \partial\Psi(y)$$

where $\partial\Psi(y) = \left\{ p \;\middle|\; \Psi(z) \geq \Psi(y) + \langle p, z - y \rangle, \forall z \right\}$ is the subdifferential of Ψ

Lifted Bregman training of neural networks

Suppose we have samples (x, y) and want to find W, b such that

$$y = \sigma(Wx + b)$$

$$\iff \qquad Wx + b \in \partial\left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)(y)$$

$$\iff \qquad \frac{1}{2}\|y\|^2 + \Psi(y) + \left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)^*(Wx + b) = \langle y, Wx + b\rangle$$

where

$$\left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)^*(z) = \sup_x \left\{\langle x, z\rangle - \frac{1}{2}\|x\|^2 - \Psi(x)\right\}$$

Legendre, Adrien-Marie. Mémoire sur l'intégration de quelques équations aux différences partielles. In Histoire de l'Académie royale des sciences, avec les mémoires de mathématique et de physique. Paris: Imprimerie royale. pp. 309–351, 1789
Werner Fenchel, *Convex cones, sets, and functions*. Princeton University, 1953
Theorem 23.5, Ralph Tyrell Rockafellar, Convex analysis, Princeton university press, 1970

Lifted Bregman training of neural networks

Lifted Bregman network

$$E(\Theta, X) = \frac{1}{2s} \sum_{i=1}^{s} \sum_{l=1}^{L} B_\Psi \left(x_i^l , W_l \, x_i^{l-1} + b_l \right)$$

with Bregman / Fenchel function

$$B_\Psi(y, z) = \frac{1}{2}\|y\|^2 + \Psi(y) + \left(\frac{1}{2}\| \cdot \|^2 + \Psi \right)^* (z) - \langle y, z \rangle$$

Wang, X., & MB. A Lifted Bregman Formulation for the Inversion of Deep Neural Networks. *Front. Appl. Math. Stat.* 9, (2023).

Lifted Bregman training of neural networks

Lifted Bregman network

$$E(\Theta, X) = \frac{1}{2s}\sum_{i=1}^{s}\sum_{l=1}^{L} B_\Psi\left(x_i^l, W_l x_i^{l-1} + b_l\right)$$

with Bregman / Fenchel function

$$B_\Psi(y, z) = \frac{1}{2}\|y\|^2 + \Psi(y) + \left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)^*(z) - \langle y, z\rangle$$

What is great about this function?

1. $B_\Psi(y,z) = \frac{1}{2}\|y - \text{prox}_\Psi(z)\|^2 + D_\Psi^{\text{prox}_{\Psi^*}(z)}(y, \text{prox}_\Psi(z)) \geq \frac{1}{2}\|y - \text{prox}_\Psi(z)\|^2 \geq 0$, for all y, z
2. $\nabla_2 B_\Psi(y,z) = \text{prox}_\Psi(z) - y$
3. $B_\Psi(y,z) = E_z(y) - E_z(\text{prox}_\Psi(z)) = D_{E_z}^0(y, \text{prox}_\Psi(z))$ for $E_z(u) := \frac{1}{2}\|u - z\|^2 + \Psi(u)$

Xiaoyu Wang, MB. Lifted Bregman Training of neural networks. *JMLR* 24(232):1—51, 2023.

Lifted Bregman training of neural networks

Lifted Bregman network

$$E(\Theta, X) = \frac{1}{2s} \sum_{i=1}^{s} \sum_{l=1}^{L} B_\Psi \left(x_i^l , W_l \, x_i^{l-1} + b_l \right)$$

with Bregman / Fenchel loss

$$B_\Psi(y, z) = \frac{1}{2}\|y\|^2 + \Psi(y) + \left(\frac{1}{2}\| \cdot \|^2 + \Psi \right)^* (z) - \langle y, z \rangle$$

Optimality conditions for W_j and b_j:

$$0 = \left(\sigma \left(W_j \, x_{j-1} + b_j \right) - x_j \right) x_{j-1}^\top$$

$$0 = \sigma \left(W_j \, x_{j-1} + b_j \right) - x_j$$

Xiaoyu Wang, MB. Lifted Bregman Training of neural networks. *JMLR* 24(232):1—51, 2023.

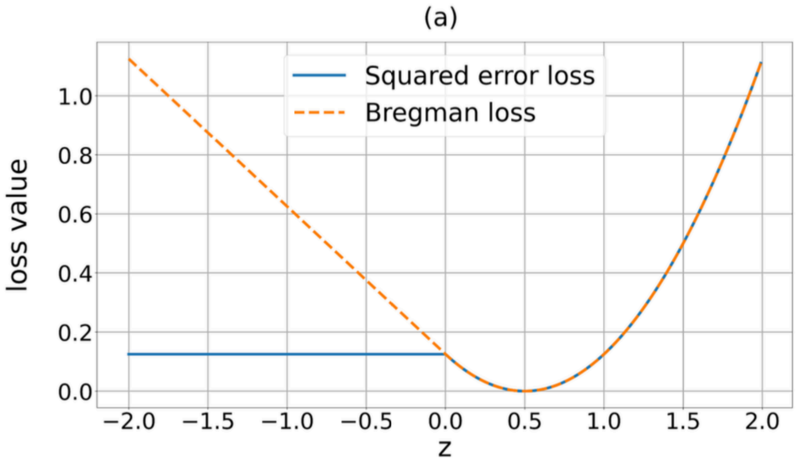
Lifted Bregman training of neural networks

Illustration:

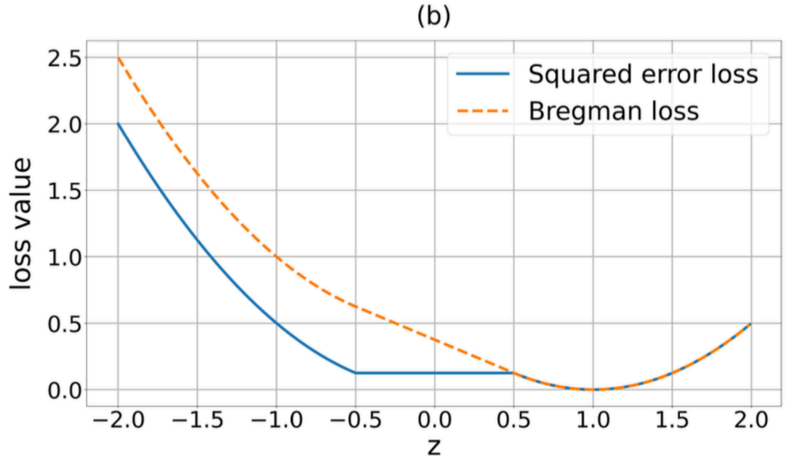
$$\frac{1}{2}|\sigma(z) - 1/2|^2$$

vs

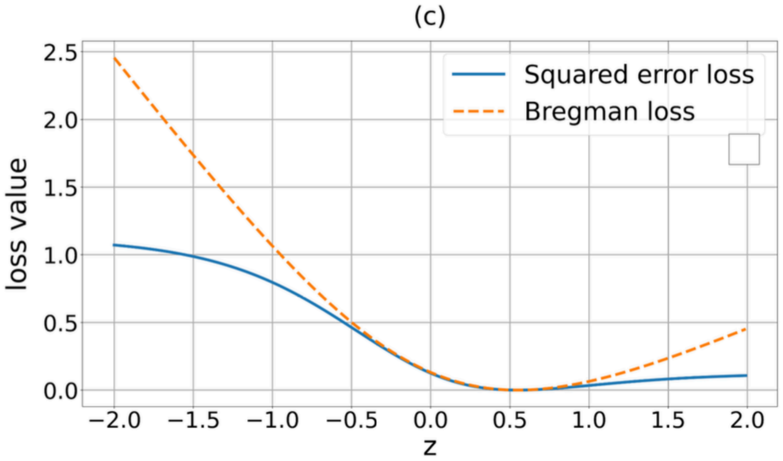
$$B_\Psi(1/2, z)$$



$\sigma(z) = \max(0, z)$



$\sigma(z) = \text{soft-thresholding}(z, 1/2)$

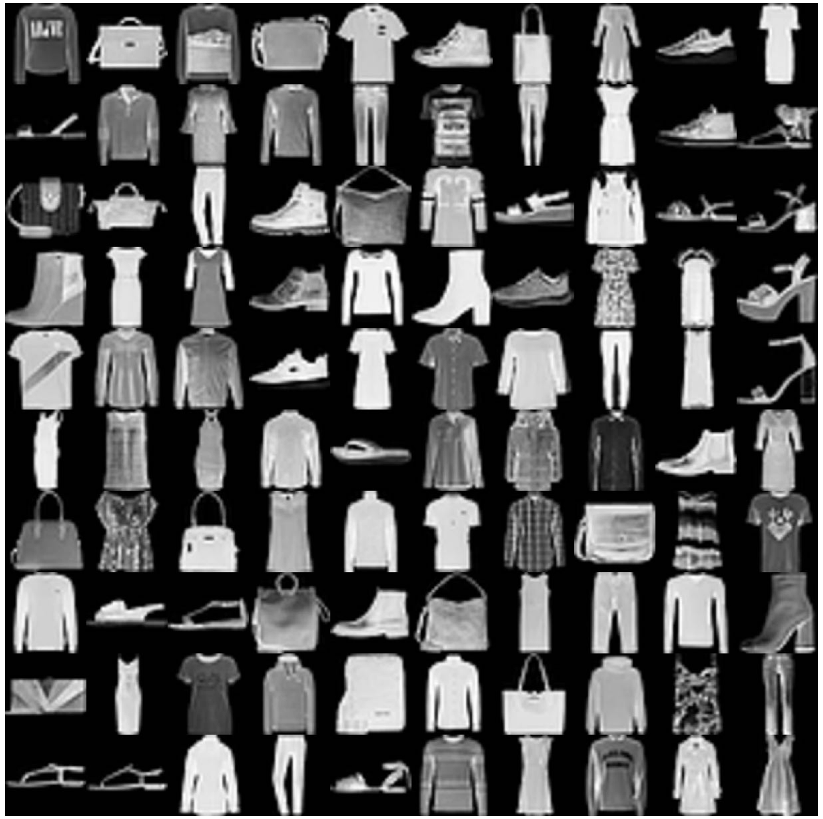


$\sigma(z) = \tanh(z)$

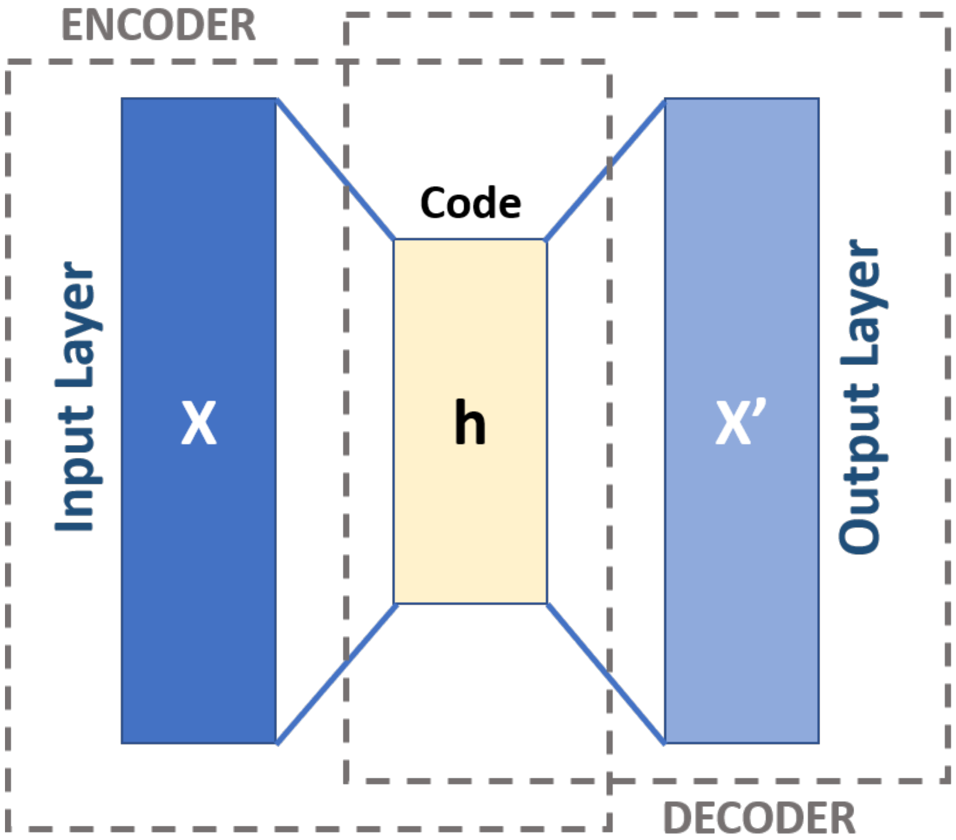
Numerical results

Sparse (denoising) autoencoder toy example

Fashion MNIST
image codec



©becominghuman.ai



© Wikimedia commons

Images are centred

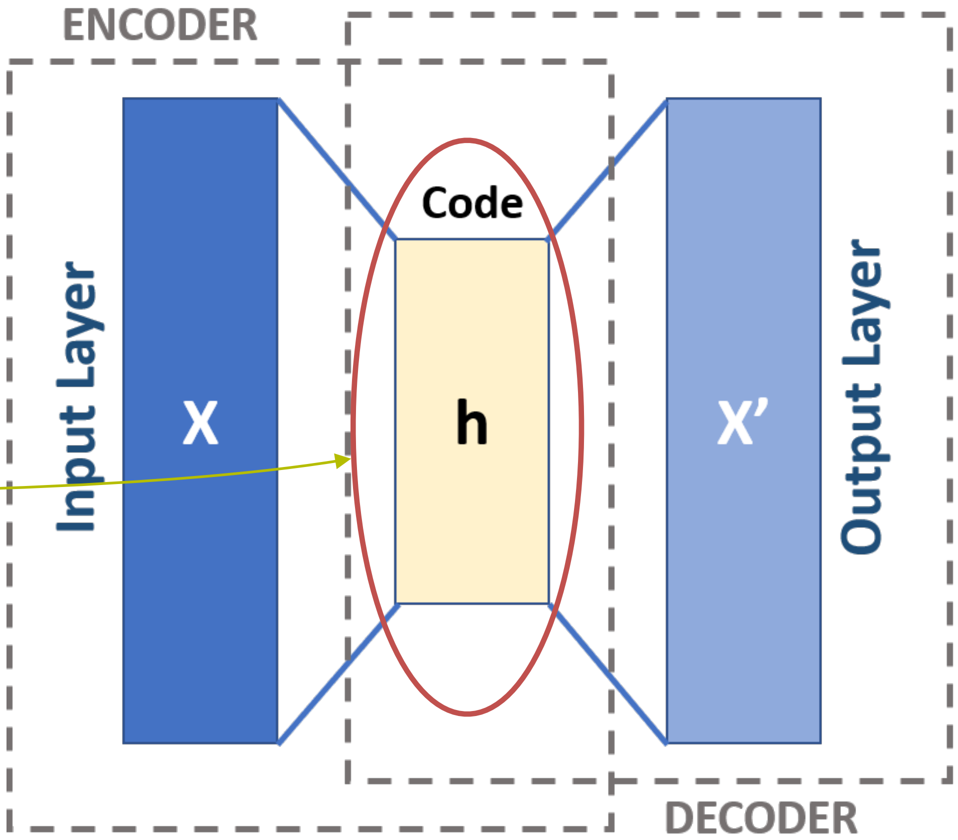
Xiao, H., Rasul, K. and Vollgraf, R., 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Numerical results

Sparse (denoising) autoencoder toy example

$$\min \sum_{i=1}^{s} \left[\sum_{j=1}^{5} B_{\Psi_j}\left(x_j^i, W_j x_{j-1}^i + b_j\right) \right] + \alpha \left\| x_3^i \right\|_1$$

Idea: make encoding sparse



[© Wikimedia commons]

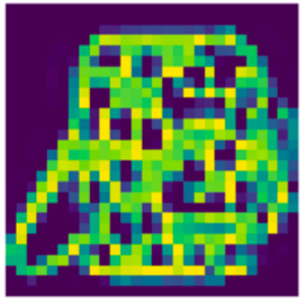
Network architecture:

$$\sigma_j(z) = \begin{cases} \max(0,z) & j \in \{1,2,4\} \\ \text{soft-thresholding}(z,\rho) & j = 3 \\ z & j = 5 \end{cases}$$

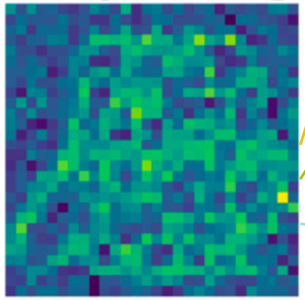
Layer dimensions 784-784-784-784-784

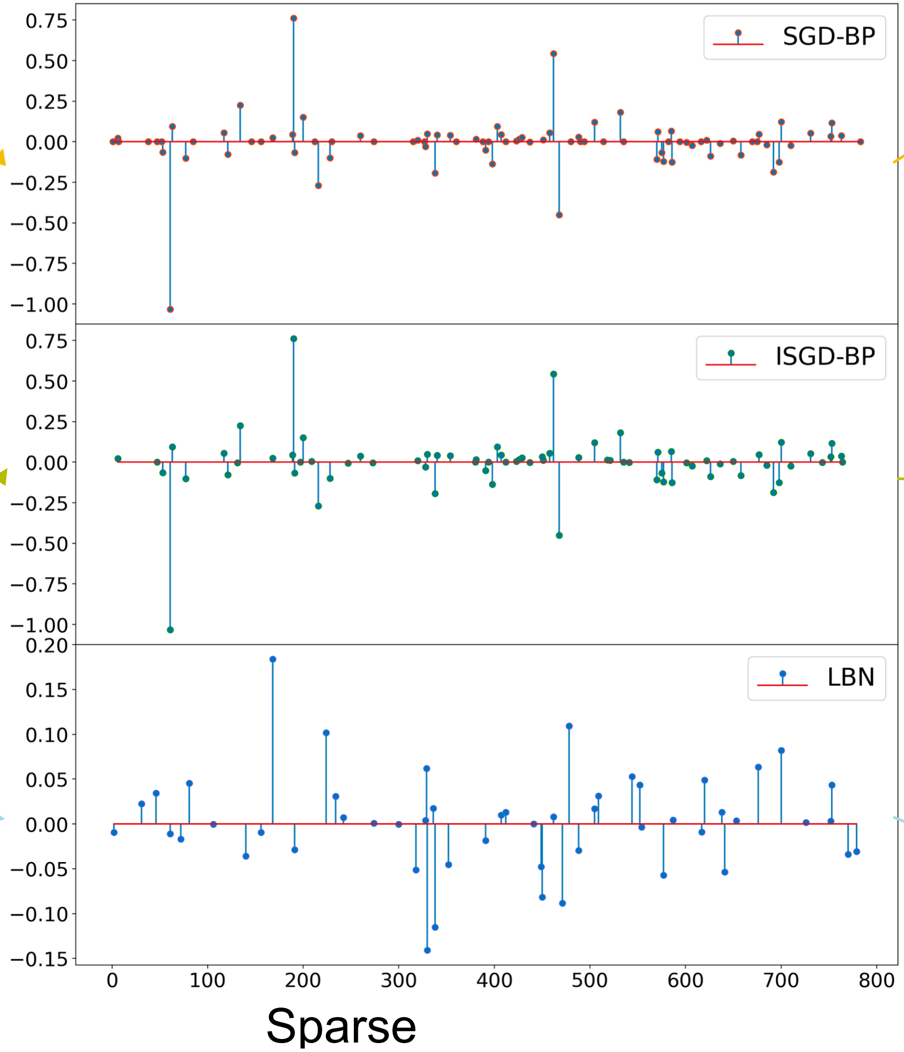
Numerical results

Groundtruth

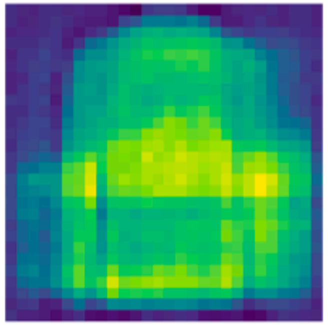


Noisy image

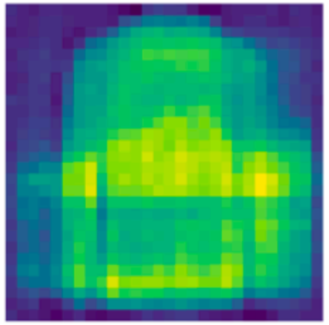




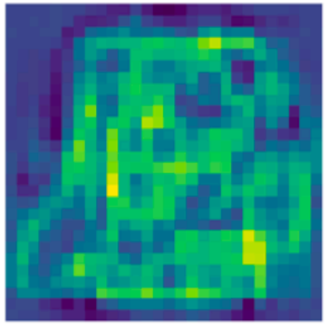
SGD-BP



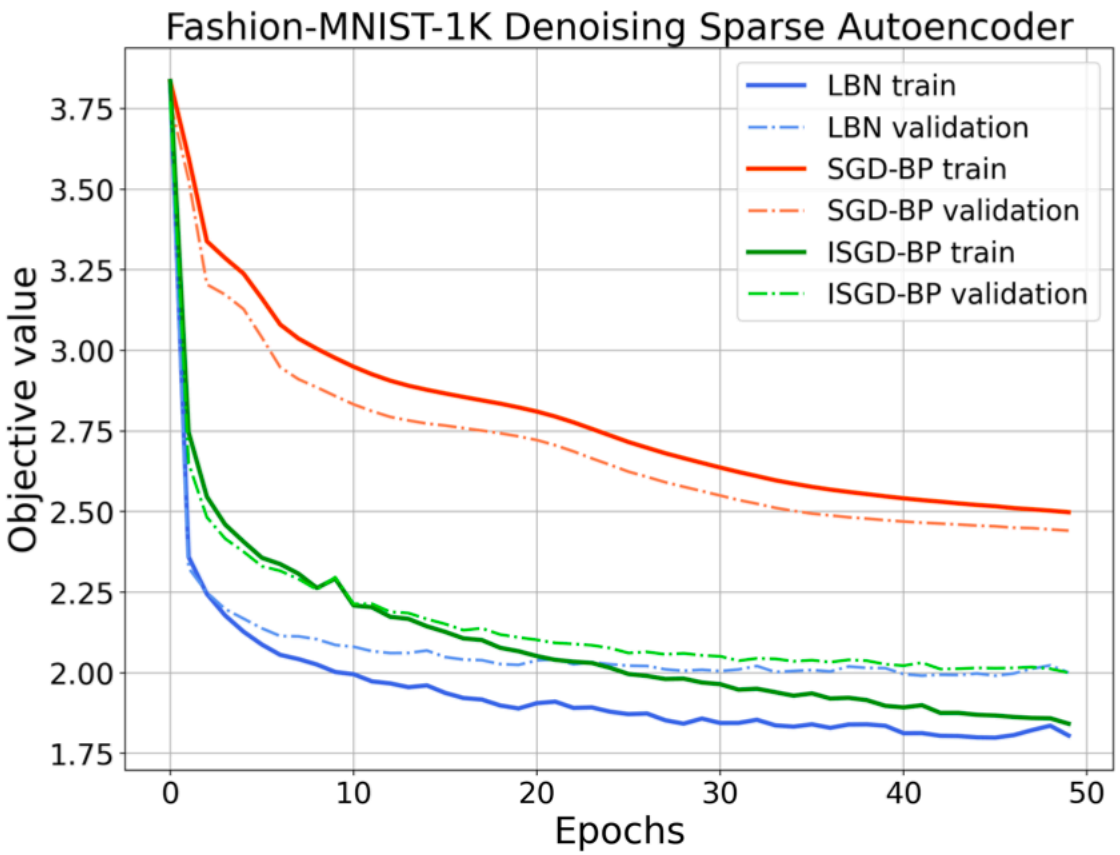
ISGD-BP



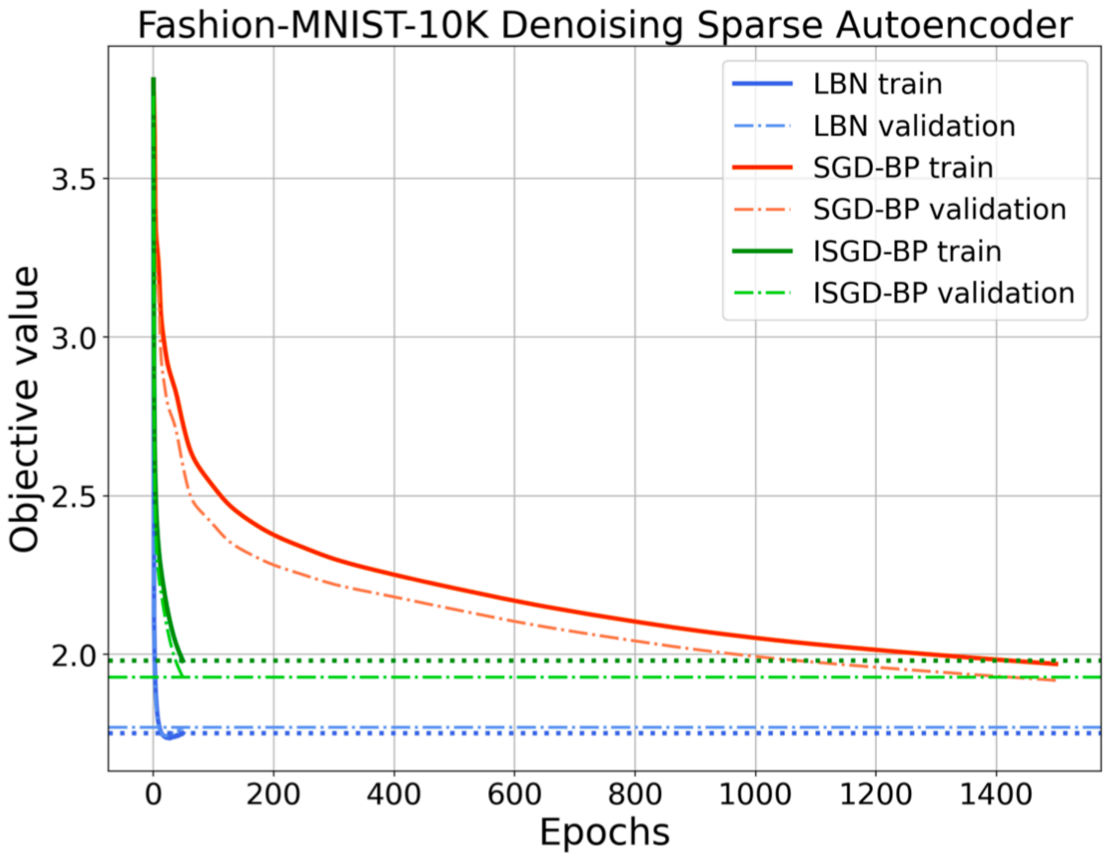
LBN



LBN minimisation via implicit SGD and proximal gradient descent for subproblems*



Trained on 1000 images



Trained on 10000 images

*implementation details are on extra slide for the Q&A if anyone is interested

Example: Proximal Neural Networks (PNNs) for image denoising

A more traditional way to solve denoising problems is using proximal maps of the form

$$\hat{x} = \arg\min_{x \in \mathbb{R}^n} \left\{ \frac{1}{2}\|x - z\|^2 + g(Lx) \right\}$$

This problem can for instance be solved with the dual forward backward algorithm*, i.e.

$$u_{k+1} = \mathrm{prox}_{\tau_k g^*}\left(u_k - \tau_k L(L^* u_k - z)\right) \qquad \text{for } k = 0,1,\ldots$$

$$\hat{x} = z - \lim_{k \to \infty} L^* u_k$$

Alternatively, one can unroll the algorithm for a fixed no. of iterations k^* and learn trainable parameters L_k, i.e.

$$u_{k+1} = \mathrm{prox}_{\tau_k g^*}\left(u_k - \tau_k L_k\left(L_k^* u_k - z\right)\right) \quad \text{for } k = 0,1,\ldots,k^* - 1$$

$$x_{k^*} = z - L_{k^*}^* u_{k^*}$$

*P. L. Combettes, Đ. Dũng, and B. C. Vũ, “Dualization of signal recovery problems,” Set-Valued Var. Anal., vol. 18, no. 3, pp. 373–404, 2010.

Example: Proximal Neural Networks (PNNs) for image denoising

Alternatively, one can unroll the algorithm for a fixed no. of iterations k^* and learn trainable parameters L_k, i.e.

$$u_{k+1} = \text{prox}_{\tau_k g^*}\left(u_k - \tau_k L_k \left(L_k^* u_k - z\right)\right) \quad \text{for } k = 0,1,\ldots,k^* - 1$$

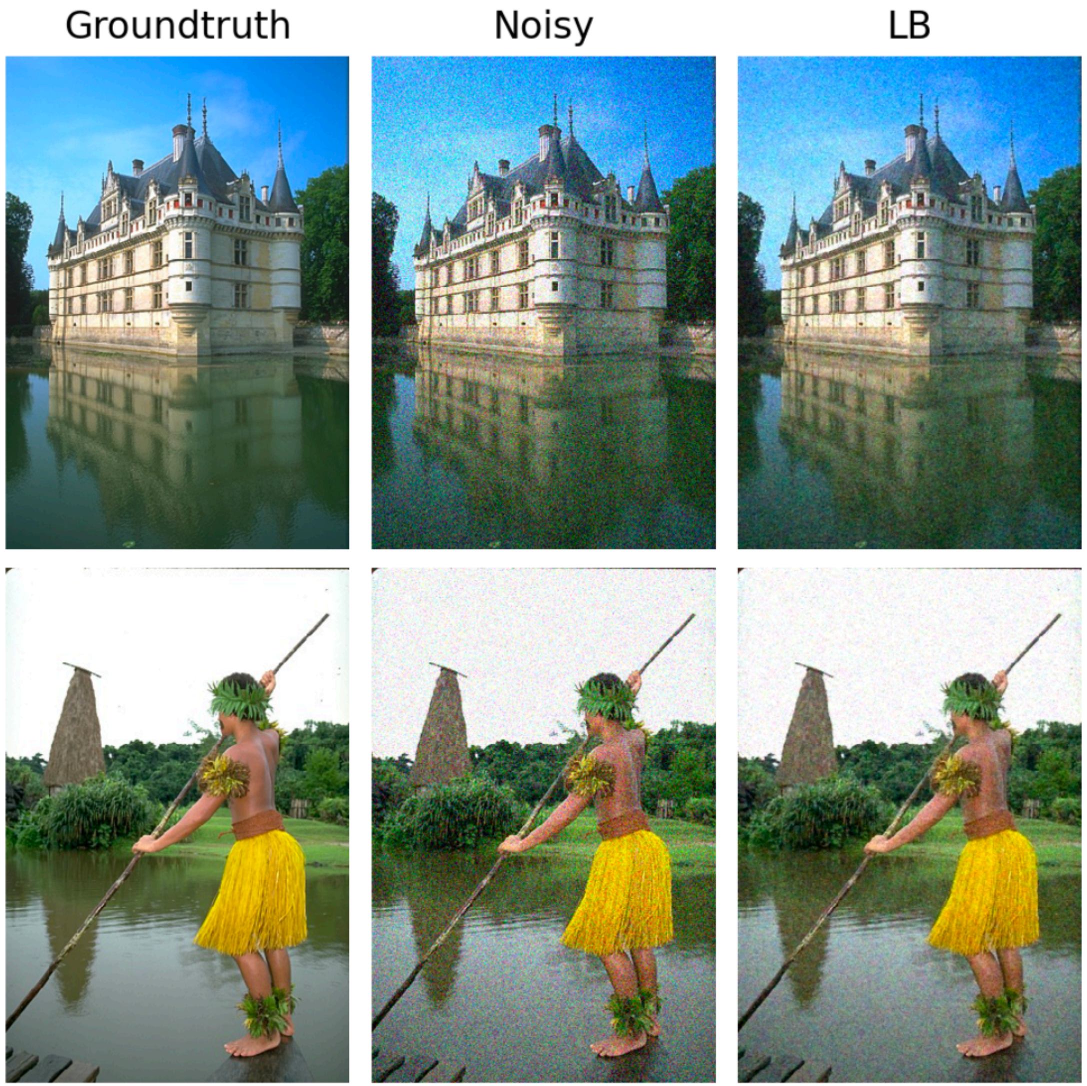
$$x_{k^*} = z - L_{k^*}^* u_{k^*}$$

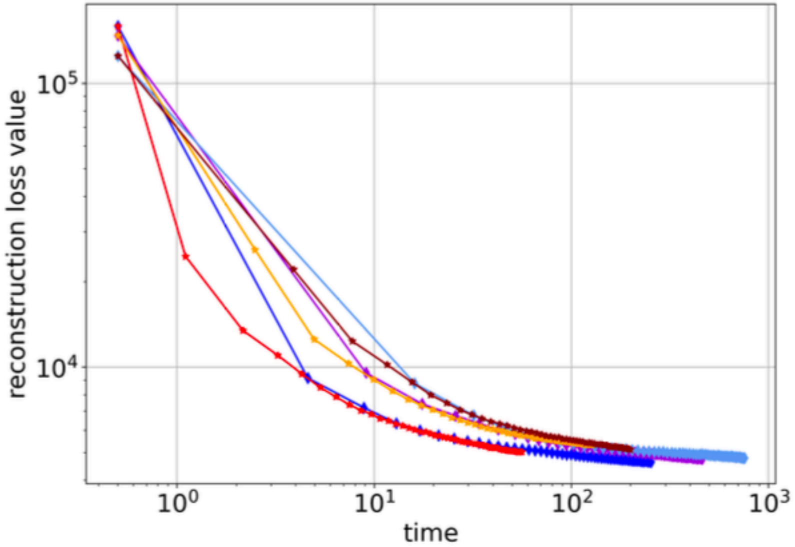
Approach perfectly suits lifted Bregman approach, i.e.

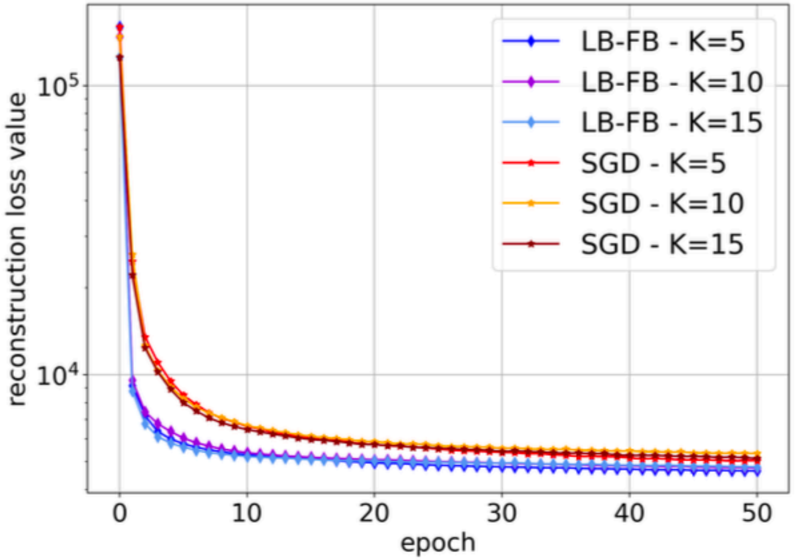
$$\min \sum_{i=1}^{s} \left[\ell(z^i - L_{k^*}^* u_{k^*}^i, \bar{x}^i) + \sum_{k=0}^{k^*-1} B_{\tau^k g^*}\left(u_{k+1}^i, u_k^i - \tau_k L_k \left(L_k^* u_k^i - z^i\right)\right) \right]$$

X. Wang, MB, A. Repetti, A lifted Bregman strategy for training unfolded proximal neural network gaussian denoisers, in: 2024 IEEE 34th International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, 2024, pp. 1–6.

Example: Proximal Neural Networks (PNNs) for image denoising







X. Wang, MB, A. Repetti, A lifted Bregman strategy for training unfolded proximal neural network gaussian denoisers, in: 2024 IEEE 34th International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, 2024, pp. 1–6.

Part II: Regularised inversion of neural networks

Inverting neural networks

We consider the (deterministic) inverse problems of the form

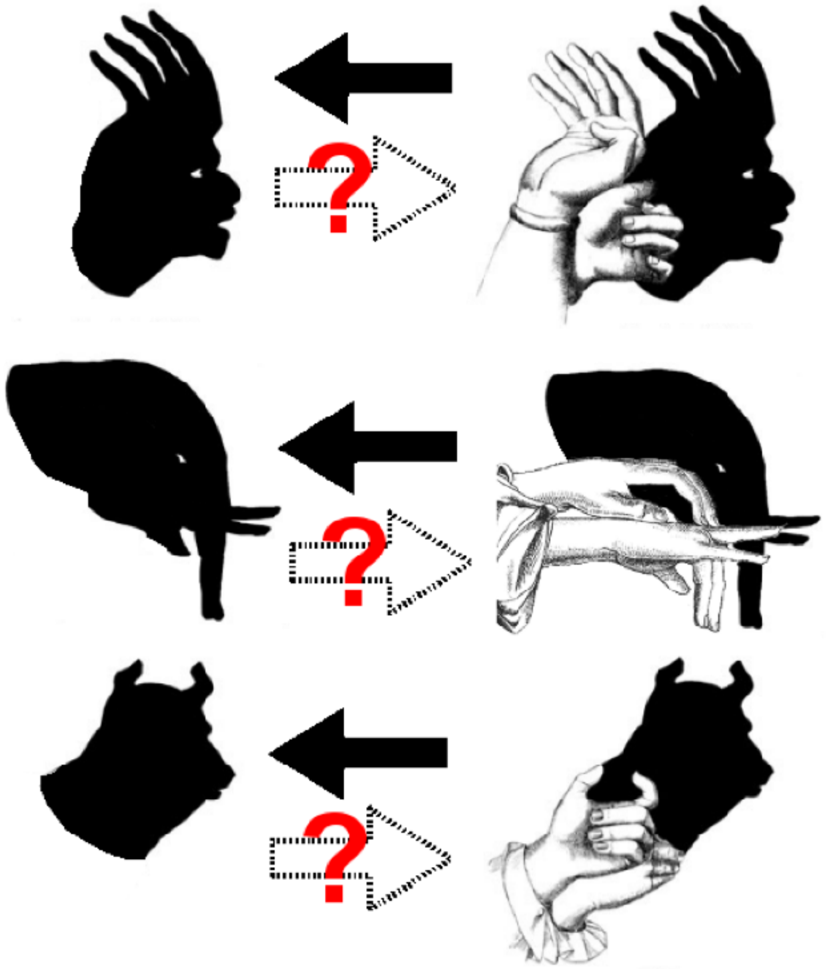
$$N(u^\dagger) = f$$

where the goal is to recover $u^\dagger$ for given data f

N is a neural network

$$f \in \text{range}(N) = \text{measured data}$$

$$u^\dagger = \text{unknown solution}$$



Engl, H. W., Hanke, M., & Neubauer, A. (1996). *Regularization of inverse problems* (Vol. 375). Springer Science & Business Media.

Benning, M., & Burger, M. (2018). Modern regularization methods for inverse problems. *Acta Numerica*, *27*, 1-111.

Inverting neural networks

We consider the (deterministic) inverse problems of the form

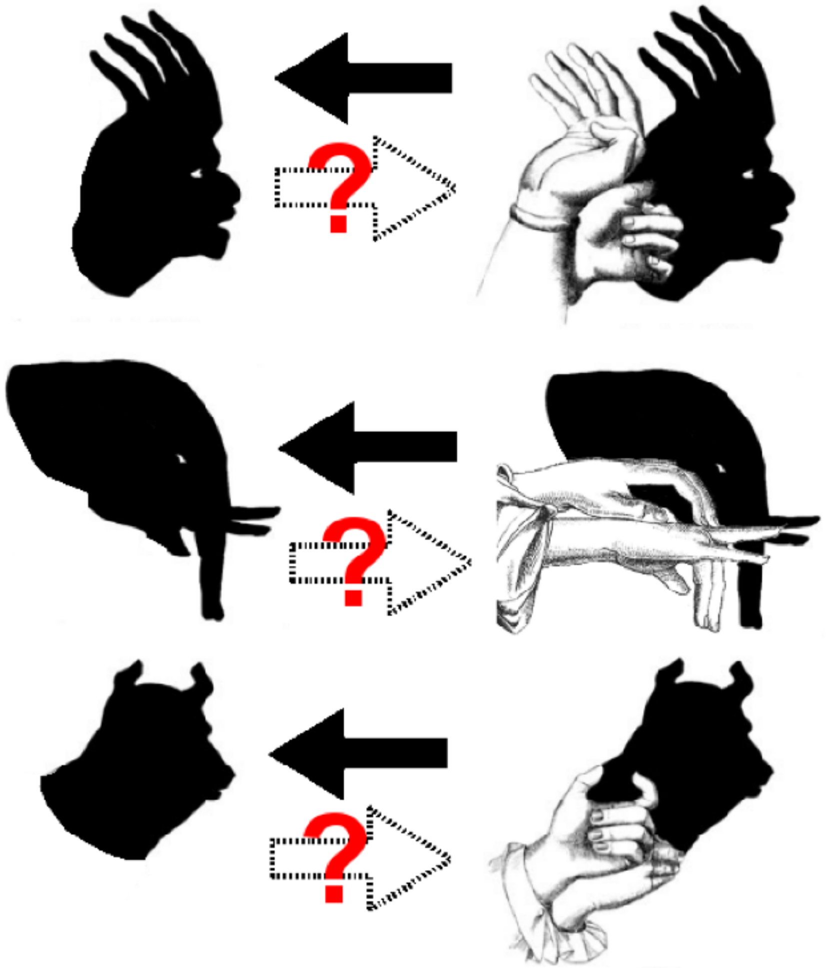
$$N(u^\dagger) = f^\delta$$

where the goal is to recover $u^\dagger$ for given data f^δ

N is a neural network

$f^\delta =$ measured data

$u^\dagger =$ unknown solution

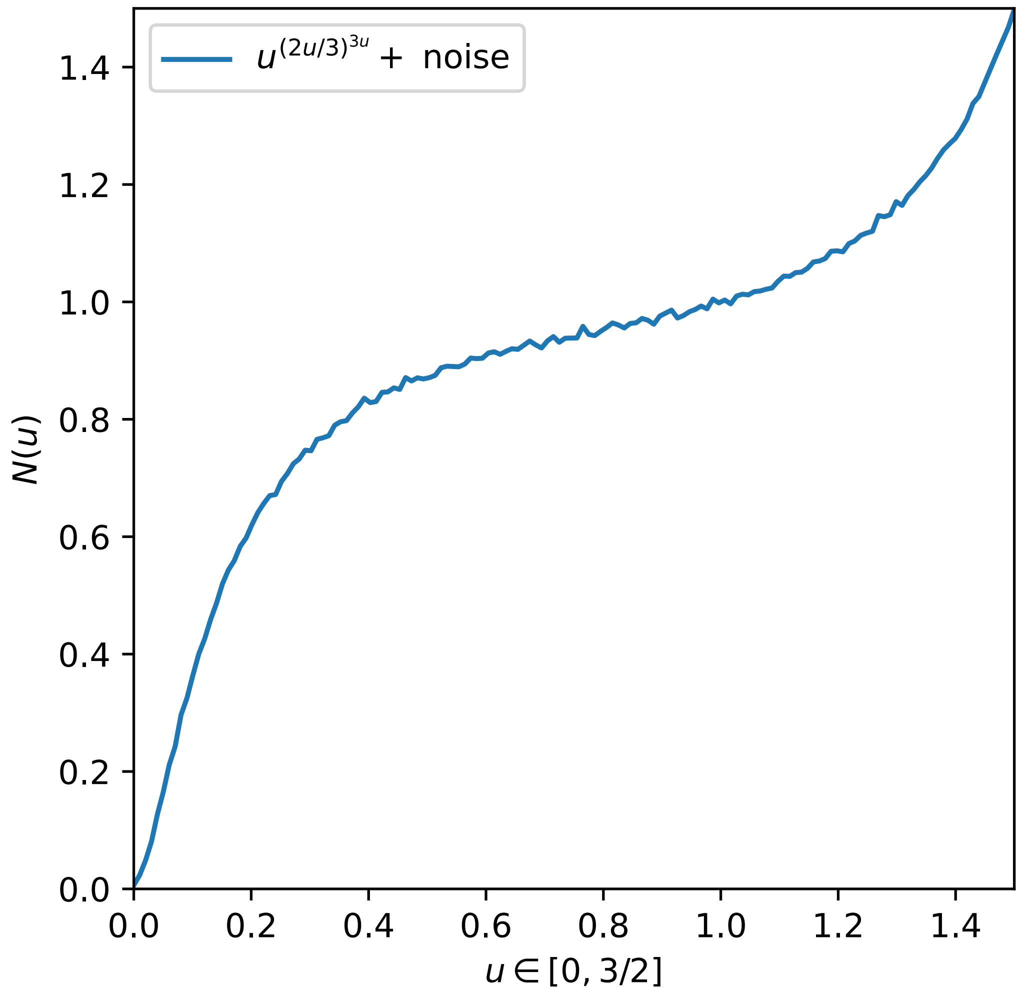


Engl, H. W., Hanke, M., & Neubauer, A. (1996). *Regularization of inverse problems* (Vol. 375). Springer Science & Business Media.

Benning, M., & Burger, M. (2018). Modern regularization methods for inverse problems. *Acta Numerica*, *27*, 1-111.

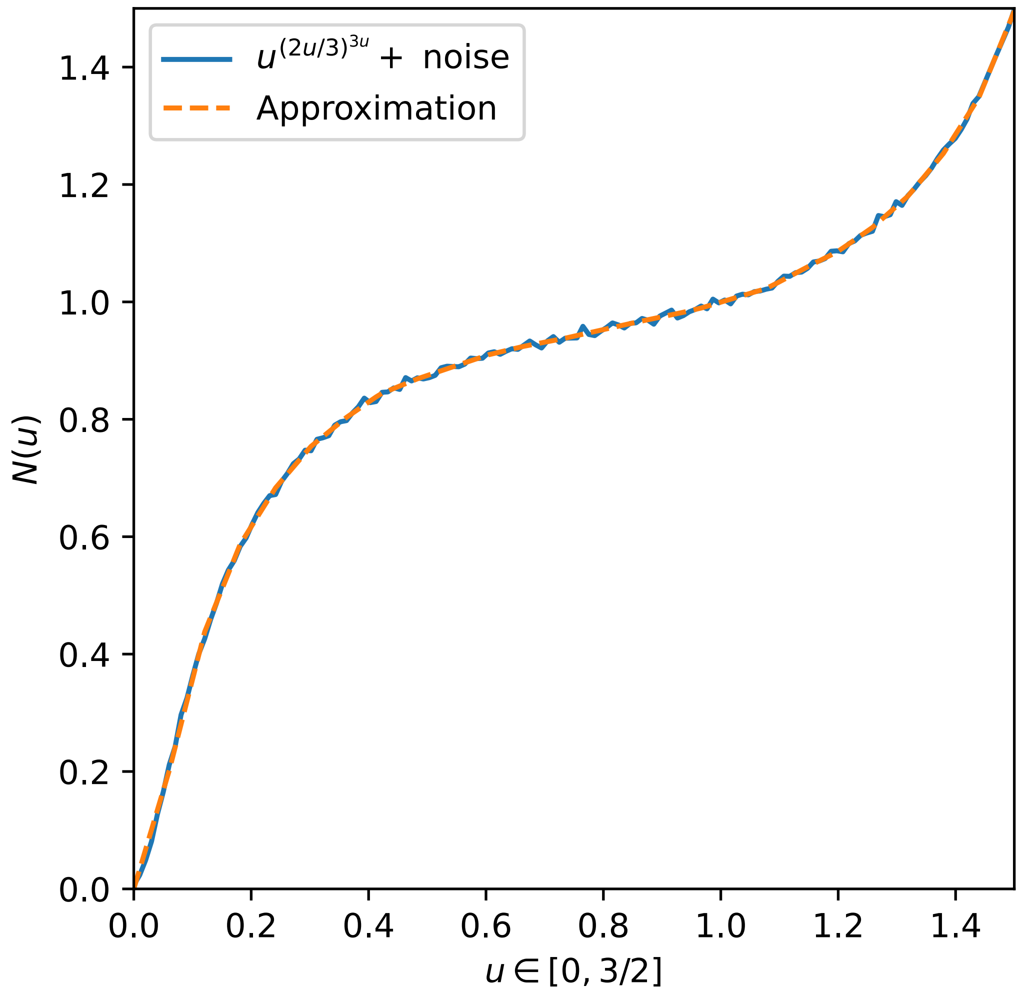
Inverting neural networks

Example:



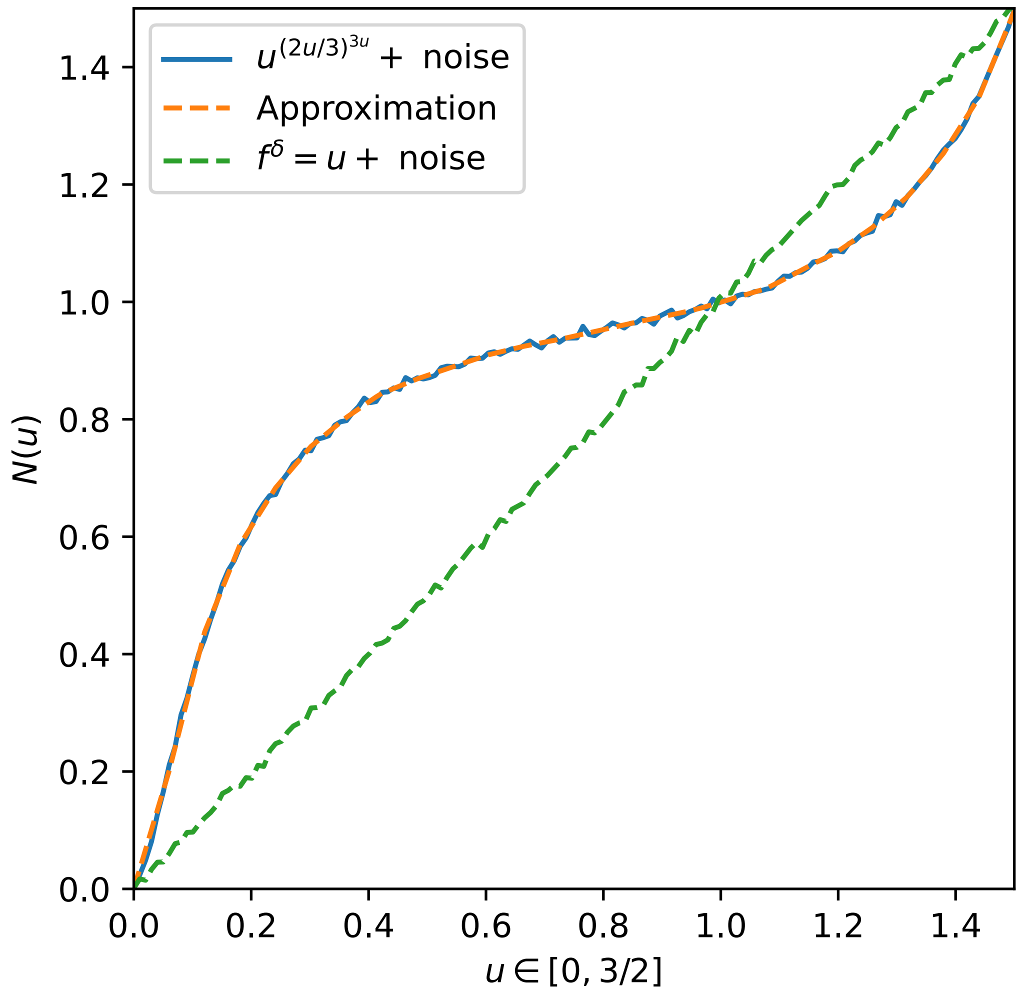
Inverting neural networks

Example:



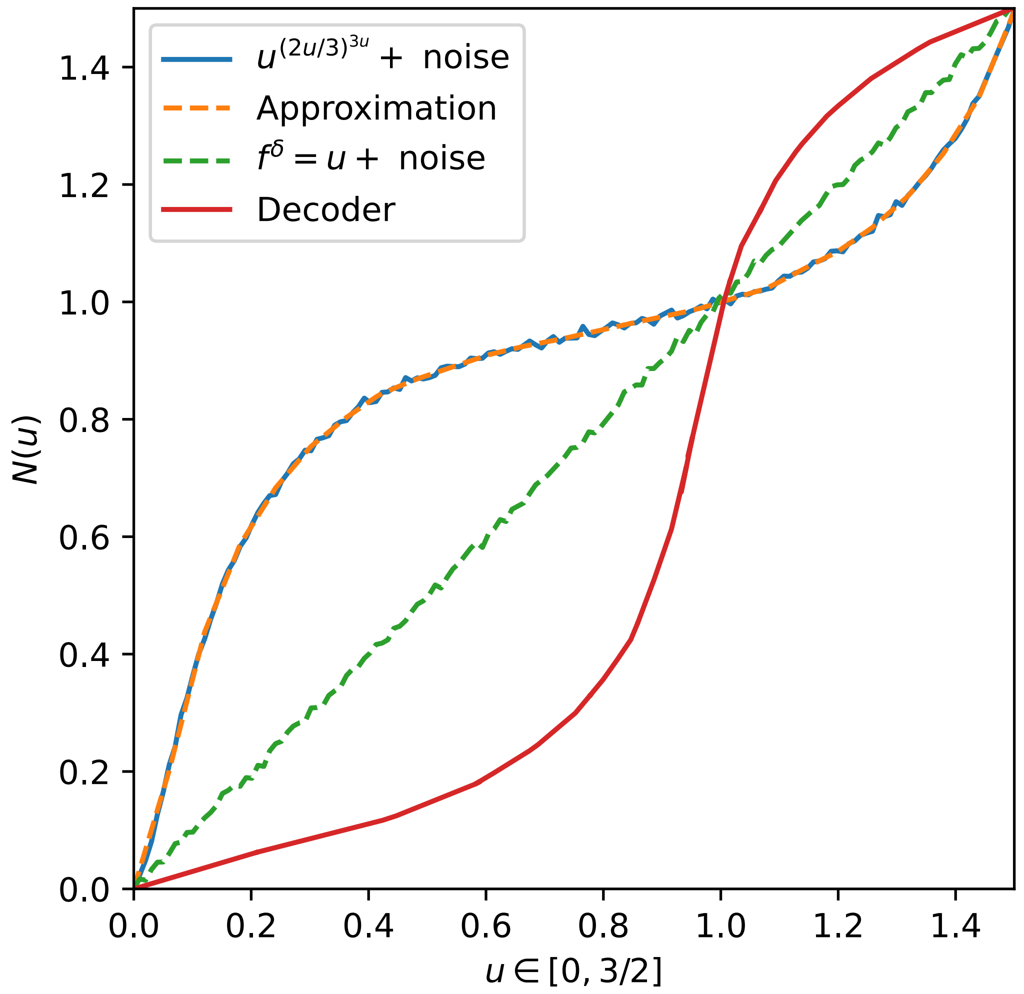
Inverting neural networks

Example:



Inverting neural networks

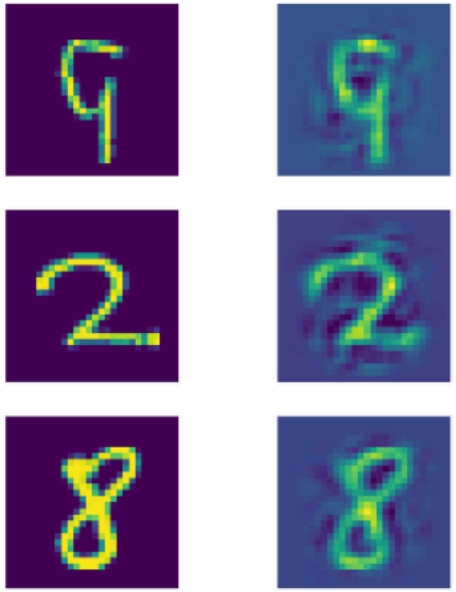
Example:



Why is this interesting?

Example: Simple autoencoder $A(u) = W_2 \max\left(0, W_1 u + b_1\right) + b_2$

Toy problem: Train A such that $A(u_i^\dagger) \approx u_i^\dagger$



$u_i^\dagger$ $A(u_i^\dagger)$

Wang, X., & MB. A Lifted Bregman Formulation for the Inversion of Deep Neural Networks. *Front. Appl. Math. Stat.* 9, (2023).

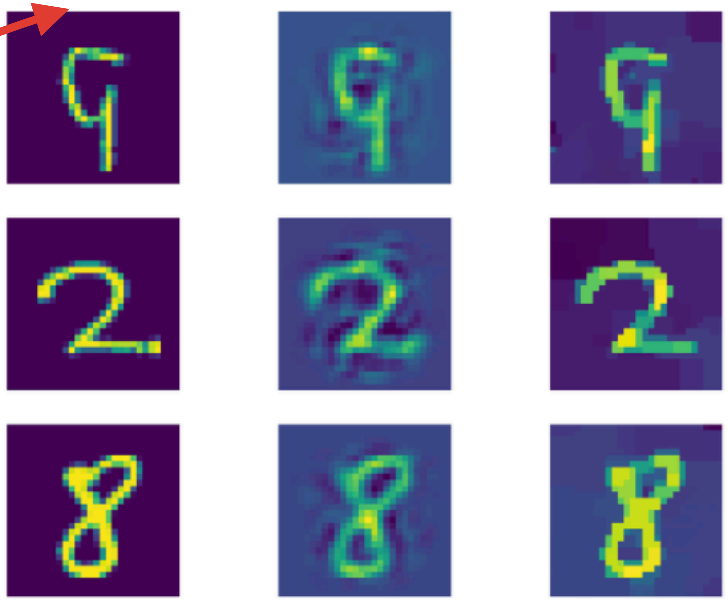
Why is this interesting?

Example: Simple autoencoder $A(u) = W_2 \max\left(0, W_1 u + b_1\right) + b_2$

Toy problem: Train A such that $A(u_i^\dagger) \approx u_i^\dagger$

Solve $R(N(u_i^\dagger)) \approx u_i^\dagger$ for $N(u) = \max\left(0, W_1 u + b_1\right)$

R does not require training and only depends on pre-trained N



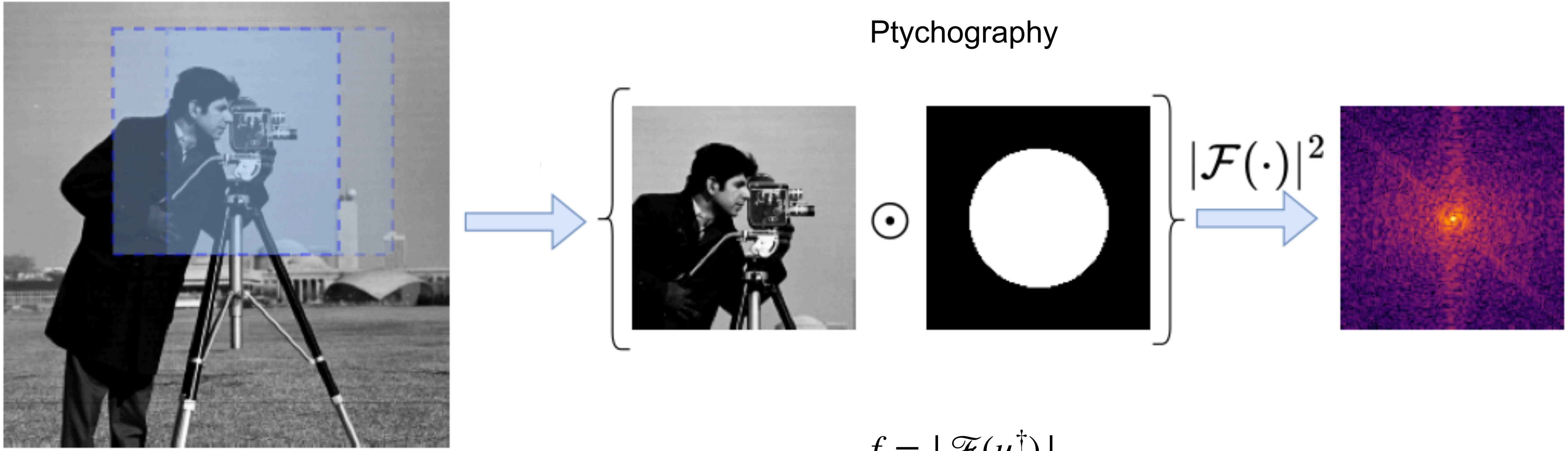
$u_i^\dagger$ $\quad A(u_i^\dagger)$ $\quad R(N(u_i^\dagger))$

We can improve pre-trained decoders by replacing them with reconstruction methods

Wang, X., & MB. A Lifted Bregman Formulation for the Inversion of Deep Neural Networks. *Front. Appl. Math. Stat.* 9, (2023).

Why is this interesting?

Example: nonlinear inverse problems

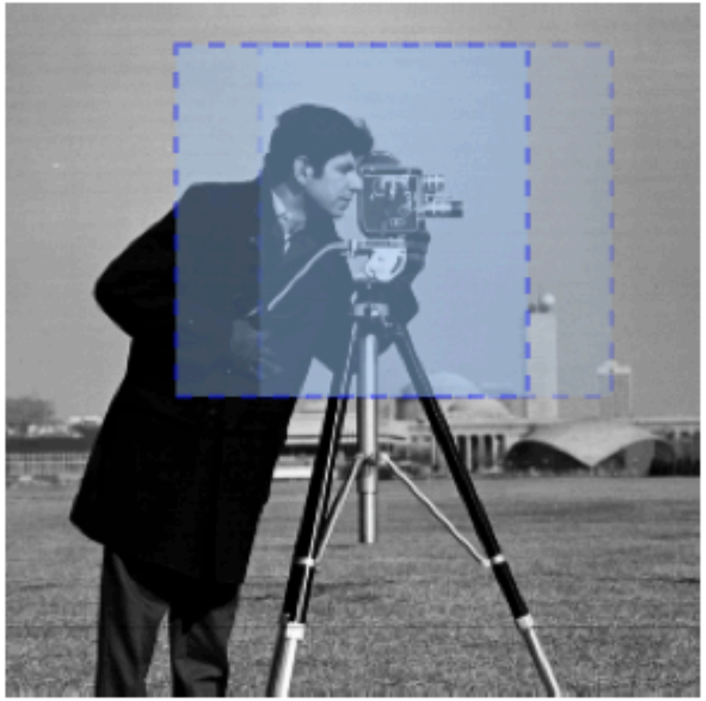


$$f = |\mathscr{F}(u^\dagger)|$$

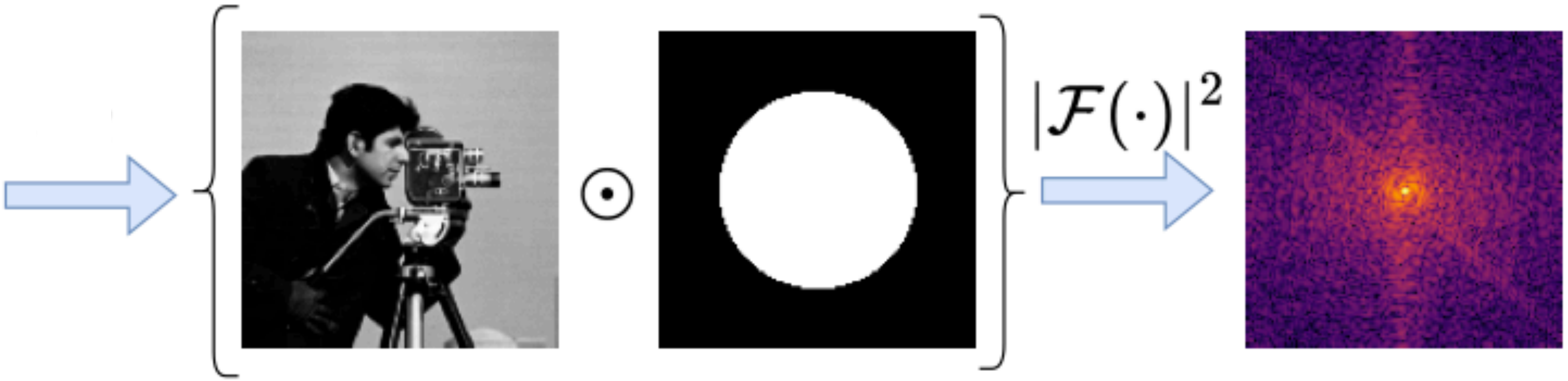
From Alexander Denker, Johannes Hertrich, Zeljko Kereta, Silvia Cipiccia, Ecem Erin, and Simon Arridge. Plug-and-play half-quadratic splitting for ptychography. arXiv preprint arXiv:2412.02548, 2024.

Why is this interesting?

Example: nonlinear inverse problems



Ptychography



$$f = |\mathscr{F}(u^\dagger)|$$

Idea: replace non-linearity $|\cdot|$ with neural network approximation N and solve $f = N(\mathscr{F}u^\dagger)$ instead

From Alexander Denker, Johannes Hertrich, Zeljko Kereta, Silvia Cipiccia, Ecem Erin, and Simon Arridge. Plug-and-play half-quadratic splitting for ptychography. arXiv preprint arXiv:2412.02548, 2024.

Inversion of neural networks

How can we invert neural networks?

We can design another neural network R to approximate the inverse of N:

$$R(N(u^\dagger)) \approx u^\dagger$$

Regularisation (Decorder)

Neural network (Encoder)

Inversion of neural networks

How can we invert neural networks?

We can design another neural network R to approximate the inverse of N:

$$R(N(u^\dagger)) \approx u^\dagger$$

Regularisation (Decorder)

Neural network (Encoder)

We choose neural networks such as

$$N(u) = \text{prox}_\Psi(Wu + b)$$

Perceptron

Proximal map

Inversion of neural networks

How can we invert neural networks?

We can design another neural network R to approximate the inverse of N:

$$R(N(u^{\dagger})) \approx u^{\dagger}$$

Regularisation (Decorder)

Neural network (Encoder)

We choose neural networks such as

$$N(u) = \mathrm{prox}_{\Psi}(Wu + b)$$

Perceptron

Inversion of neural networks

How can we invert neural networks?

We can design another neural network R to approximate the inverse of N:

$$R(N(u^\dagger)) \approx u^\dagger$$

Regularisation (Decorder)

Neural network (Encoder)

We choose neural networks such as

$$N(u) = W_l \, \mathrm{prox}_{\Psi_{l-1}}(W_{l-1} \cdots \; W_2 \mathrm{prox}_{\Psi_1}(W_1 u + b_1) + b_2) \cdots \; b_{l-1}) + b_l$$

Feed-forward networks

Inversion of neural networks

How can we invert neural networks?

We can design another neural network R to approximate the inverse of N:

$$R(N(u^{\dagger})) \approx u^{\dagger}$$

Regularisation (Decorder)

Neural network (Encoder)

We choose neural networks such as

$$N(u) = W_l u_{l-1} + b_l$$

$$u_j = u_{j-1} + V_{j-1}\mathrm{prox}_{\Psi_{j-1}}\left(W_{j-1}u_{j-1} + b_{j-1}\right)$$

Residual neural networks

Inversion of neural networks

How can we invert neural networks?

We can design another neural network R to approximate the inverse of N:

$$R(N(u^\dagger)) \approx u^\dagger$$

Regularisation (Decorder)

Neural network (Encoder)

All previous networks can be written in compact form as

$$N(u) = \mathbf{Ku}$$

$$\mathbf{Mu} = \mathbf{V}\mathbf{prox}_{\Psi}(\mathbf{Wu} + \mathbf{b})$$

tensor representation of all layers in one variable

Inversion of neural networks

How can we invert neural networks?

We can design another neural network R to approximate the inverse of N:

$$R(N(u^\dagger)) \approx u^\dagger$$

or more like

$$R(f^\delta) \to u^\dagger \qquad \text{for} \qquad f^\delta \to f = N(u^\dagger) \qquad \text{when} \qquad \delta \to 0$$

Open questions:

- What architecture should we choose for R ?
- Can we treat N as a black box or do we need to know its architecture and parameters when we construct R ?
- Do we need to train R, possibly from scratch?
- Do we have any mathematical guarantees that R approximates the inverse of N ?

Inversion of neural networks

We can design another neural network R to approximate the inverse of N:

$$R(N(u^\dagger)) \approx u^\dagger$$

Open questions:

- What architecture should we choose for R ? — No idea
- Can we treat N as a black box or do we need to know its architecture and parameters when we construct R ? — Yes
- Do we need to train R, possibly from scratch? — Yes
- Do we have any mathematical guarantees that R approximates the inverse of N ? — No

Example:

Neural network with arbitrary architecture

$$R(f^\delta) = h_l\left(h_{l-1}\left(\cdots\ h_1(f^\delta, p_1), \cdots, p_{l-1}\right), p_l\right)$$

Activation functions $h_1, \ldots, h_l$

Parameters $p_1, \ldots, p_l$

Inversion of neural networks

We can design another neural network R to approximate the inverse of N:

$$R(N(u^{\dagger})) \approx u^{\dagger}$$

Open questions:

- What architecture should we choose for R ? — Some ideas
- Can we treat N as a black box or do we need to know its architecture and parameters when we construct R ? — No*
- Do we need to train R, possibly from scratch? — No
- Do we have any mathematical guarantees that R approximates the inverse of N ? — Possibly

Example:

Variational regularisation with quadratic fidelity

$$R(f^{\delta}) \in \arg\min_{u} \left\{ \frac{1}{2} \|N(u) - f^{\delta}\|^2 + \alpha J(u) \right\}$$

*Usually requires computation of backward-pass $(\nabla N)^*$; and can be as challenging as if one were to use K directly

Inversion of neural networks

We can design another neural network R to approximate the inverse of N:

$$R(N(u^\dagger)) \approx u^\dagger$$

Open questions:

- What architecture should we choose for R ? — Some ideas
- Can we treat N as a black box or do we need to know its architecture and parameters when we construct R ? — No*
- Do we need to train R, possibly from scratch? — No
- Do we have any mathematical guarantees that R approximates the inverse of N ? — Possibly

Example:

Iterative regularisation with quadratic fidelity

$$R(f^\delta) = u^{k*} \qquad \text{for} \qquad u^{k+1} \in \arg\min_u \left\{ \frac{1}{2}\|N(u) - f^\delta\|^2 + \alpha\, D_J(u, u^k) \right\}$$ + stopping criterion

*Usually requires computation of backward-pass $(\nabla N)^*$; and can be as challenging as if one were to use K directly

Inversion of neural networks

We can design another neural network R to approximate the inverse of N:

$$R(N(u^\dagger)) \approx u^\dagger$$

Open questions:

- What architecture should we choose for R ? — Several options
- Can we treat N as a black box or do we need to know its architecture and parameters when we construct R ? — No
- Do we need to train R, possibly from scratch? — No
- Do we have any mathematical guarantees that R approximates the inverse of N ? — Yes

Example:

Variational regularisation with bespoke fidelity

$$R(f^\delta) \in \arg\min_u \left\{ \text{Bespoke}\left(\mathcal{N}(u), f^\delta\right) + \alpha\, J(u) \right\}$$

In the following, we will derive a suitable candidate for this bespoke data fidelity term

Inversion of neural networks

How can we invert neural networks?

We can design another neural network R to approximate the inverse of N:

$$R(N(u^\dagger)) \approx u^\dagger$$

Regularisation (Decorder)

Neural network (Encoder)

One possible choice for R:

$$(\mathbf{u}_\rho, \mathbf{z}_\rho) \in \arg\min_{\mathbf{u},\mathbf{z}} \left\{ E_\Psi^\rho(\mathbf{u}, \mathbf{z}) + J(\mathbf{u}) \right\}$$

(variational regularisation)

Inversion of neural networks

One possible choice for R:

$$(\mathbf{u}_\rho, \mathbf{z}_\rho) \in \arg\min_{\mathbf{u},\mathbf{z}} \left\{ E^\rho_{\boldsymbol{\Psi}}(\mathbf{u}, \mathbf{z}) + J(\mathbf{u}) \right\}$$ (variational regularisation)

with regularisation function J and data fidelity $E^\rho_{\boldsymbol{\Psi}}$ defined as

$$E^\rho_{\boldsymbol{\Psi}}(\mathbf{u}, \mathbf{z}) = \frac{\lambda}{2}\|\mathbf{K}\mathbf{u} - f^\delta\|^2 + B_{\boldsymbol{\Psi}}(\mathbf{z}, \mathbf{W}\mathbf{u} + \mathbf{b}) + \chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) + \frac{\rho}{2}\|\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}\|^2$$

with Fenchel / Bregman penalty function

$$B_{\boldsymbol{\Psi}}(\mathbf{z}, \mathbf{x}) = \left(\frac{1}{2}\|\cdot\|^2 + \boldsymbol{\Psi}\right)(\mathbf{z}) + \left(\frac{1}{2}\|\cdot\|^2 + \boldsymbol{\Psi}\right)^*(\mathbf{x}) - \langle \mathbf{z}, \mathbf{x}\rangle$$

Inversion of neural networks

One possible choice for R:

$$(\mathbf{u}_\rho, \mathbf{z}_\rho) \in \arg\min_{\mathbf{u},\mathbf{z}} \left\{ E_\Psi^\rho(\mathbf{u}, \mathbf{z}) + J(\mathbf{u}) \right\} \qquad \text{(variational regularisation)}$$

with regularisation function J and data fidelity E_Ψ^ρ defined as

$$E_\Psi^\rho(\mathbf{u}, \mathbf{z}) = \frac{\lambda}{2}\|\mathbf{K}\mathbf{u} - f^\delta\|^2 + B_\Psi(\mathbf{z}, \mathbf{W}\mathbf{u} + \mathbf{b}) + \chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) + \frac{\rho}{2}\|\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}\|^2$$

with Fenchel / Bregman penalty function

$$B_\Psi(\mathbf{z}, \mathbf{x}) = \left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)(\mathbf{z}) + \left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)^*(\mathbf{x}) - \langle \mathbf{z}, \mathbf{x} \rangle$$

$$\chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) = 0 \quad \Longleftrightarrow \quad \mathbf{M}\mathbf{u} = \mathbf{V}\mathbf{z}$$

$$B_\Psi(\mathbf{z}, \mathbf{W}\mathbf{x} + \mathbf{b}) = 0 \quad \Longleftrightarrow \quad \mathbf{z} = \mathrm{prox}_\Psi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Inversion of neural networks

Example: Shallow two-layer neural networks (or linear combinations of 1d perceptrons)

$$N(u) = \sum_{j=1}^{m} c_j \, \mathrm{prox}_{\Psi_j}(w_j u + b_j) \qquad u, w_j, b_j, c_j \in \mathbb{R}$$

Inversion of neural networks

Example: Shallow two-layer neural networks (or linear combinations of 1d perceptrons)

$$N(u) = \sum_{j=1}^{m} c_j \, u_j \qquad u, w_j, b_j, c_j \in \mathbb{R}$$

$$u_j = \operatorname{prox}_{\Psi_j}(w_j \, u + b_j) \qquad \forall j \in \{1, \ldots, m\}$$

Inversion of neural networks

Example: Shallow two-layer neural networks (or linear combinations of 1d perceptrons)

$$N(u) = \sum_{j=1}^{m} c_j\, u_j \qquad u, w_j, b_j, c_j \in \mathbb{R}$$

$$u_j = \text{prox}_{\Psi_j}(w_j\, u + b_j) \qquad \forall j \in \{1,\ldots,m\}$$

Corresponding variational regularisation framework:

$$u_\alpha \in \arg\min_{u\in\mathbb{R}^{1+m}} \left\{ \frac{1}{2}\left| f^\delta - \sum_{j=1}^{m} c_j u_j \right|^2 + \sum_{j=1}^{m} B_{\Psi_j}(u_j, w_j u_0 + b_j) + \alpha J(u_0, u_1, \ldots, u_m) \right\}$$

Implicit/explicit coordinate descent implementation for choice $J(u_0, u_1, \ldots, u_m) = \frac{1}{2}|u_0|^2$

Inversion of neural networks

Corresponding variational regularisation framework:

$$u_\alpha \in \arg\min_{u \in \mathbb{R}^{1+m}} \left\{ \frac{1}{2} \left| f^\delta - \sum_{j=1}^m c_j u_j \right|^2 + \sum_{j=1}^m B_{\Psi_j}(u_j, w_j u_0 + b_j) + \alpha J(u_0, u_1, \ldots, u_m) \right\}$$

Implicit/explicit coordinate descent implementation for choice $J(u_0, u_1, \ldots, u_m) = \frac{1}{2} | u_0 |^2$

$$u_l^{k+1} = \text{prox}_{(1+c_l^2)^{-1}\Psi_l} \left(\frac{c_l \left(f^\delta - \sum_{j=1}^{l-1} c_j u_j^{k+1} - \sum_{j=l+1}^m c_j u_j^k \right) + w_l u_0^k + b_l}{1 + c_l^2} \right) \qquad \forall l \in \{1, \ldots, m\}$$

$$u_0^{k+1} = (1 + \alpha / \|w\|^2)^{-1} \left(u_0^k - \|w\|^{-2} \sum_{j=1}^m w_j \left(\text{prox}_{\Psi_j}(w_j u_0^k + b_j) - u_j^{k+1} \right) \right)$$

Inversion of neural networks

Encoder:

$$N(u) = \sum_{j=1}^{m} c_j \; \mathrm{prox}_j (w_j u + b_j)$$

Decoder: $R(f^\delta) = \begin{pmatrix} u_0^* & u_1^* & \cdots & u_m^* \end{pmatrix}^\top$ where $u_0^*, u_1^*, \cdots, u_m^*$ are solutions of the fixed-point iteration

$$u_l^{k+1} = \mathrm{prox}_{(1+c_l^2)^{-1}\Psi_l}\left(\frac{c_l\left(f^\delta - \sum_{j=1}^{l-1} c_j u_j^{k+1} - \sum_{j=l+1}^{m} c_j u_j^k\right) + w_l u_0^k + b_l}{1 + c_l^2} \right) \qquad \forall l \in \{1,\ldots,m\}$$

$$u_0^{k+1} = (1 + \alpha/\|w\|^2)^{-1}\left(u_0^k - \|w\|^{-2} \sum_{j=1}^{m} w_j \left(\mathrm{prox}_{\Psi_j}(w_j u_0^k + b_j) - u_j^{k+1} \right) \right)$$

Inversion of neural networks

Example:

$$N(u) = \sum_{j=1}^{m} c_j \, \text{prox}_j (w_j u + b_j)$$

$$\Psi_j(v) = \begin{cases} 0 & v \geq 0 \\ \infty & v < 0 \end{cases} \quad \Rightarrow \quad \text{prox}_{\Psi_j}(z) = \text{ReLU}(z) = \max(0,z)$$

$$m = 25 \qquad w_j = 1, \forall j \in \{1,\ldots,25\} \qquad b_j = -(j-1)h, \quad j \in \{1,\ldots,25\}, \quad h = 3/50$$

$$\alpha = 10^{-4}$$

Inversion of neural networks

Example:

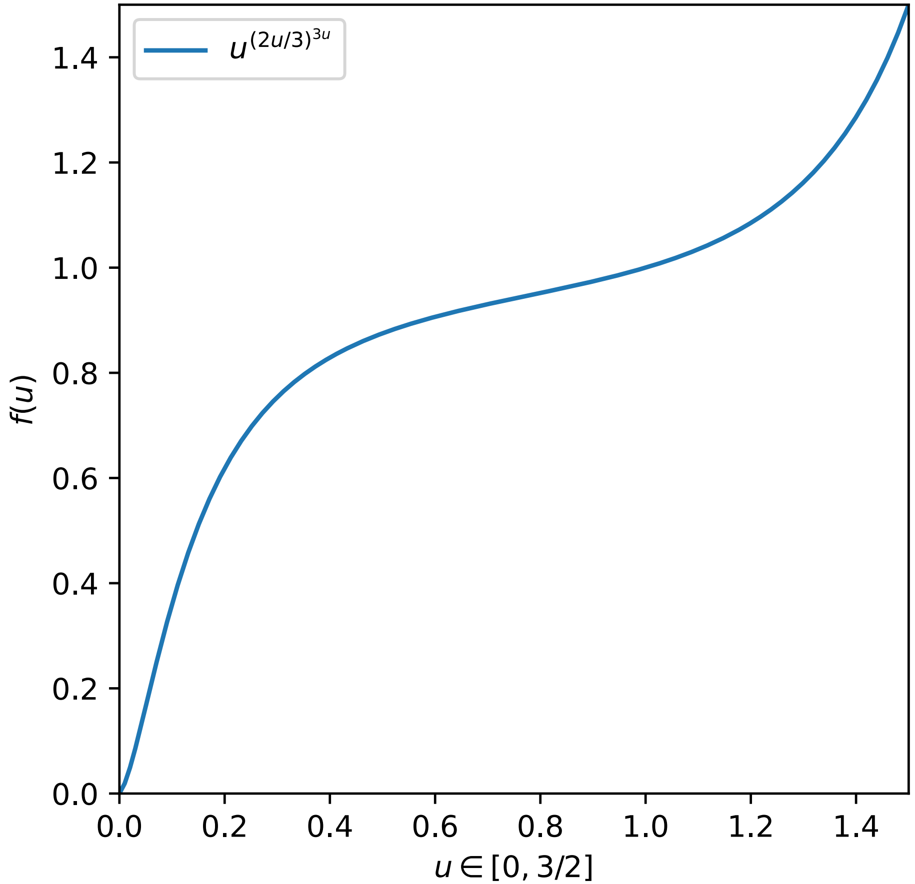
$$N(u) = \sum_{j=1}^{m} c_j \ \max(0, w_j u + b_j)$$

$m = 25 \qquad w_j = 1, \forall j \in \{1,\ldots,25\}$

$b_j = -(j-1)h, \quad j \in \{1,\ldots,25\}, \quad h = 3/50$

$\alpha = 10^{-4}$

Function $K(u) = u^{\left(\frac{2}{3}u\right)^{3u}}$



Inversion of neural networks

Example:

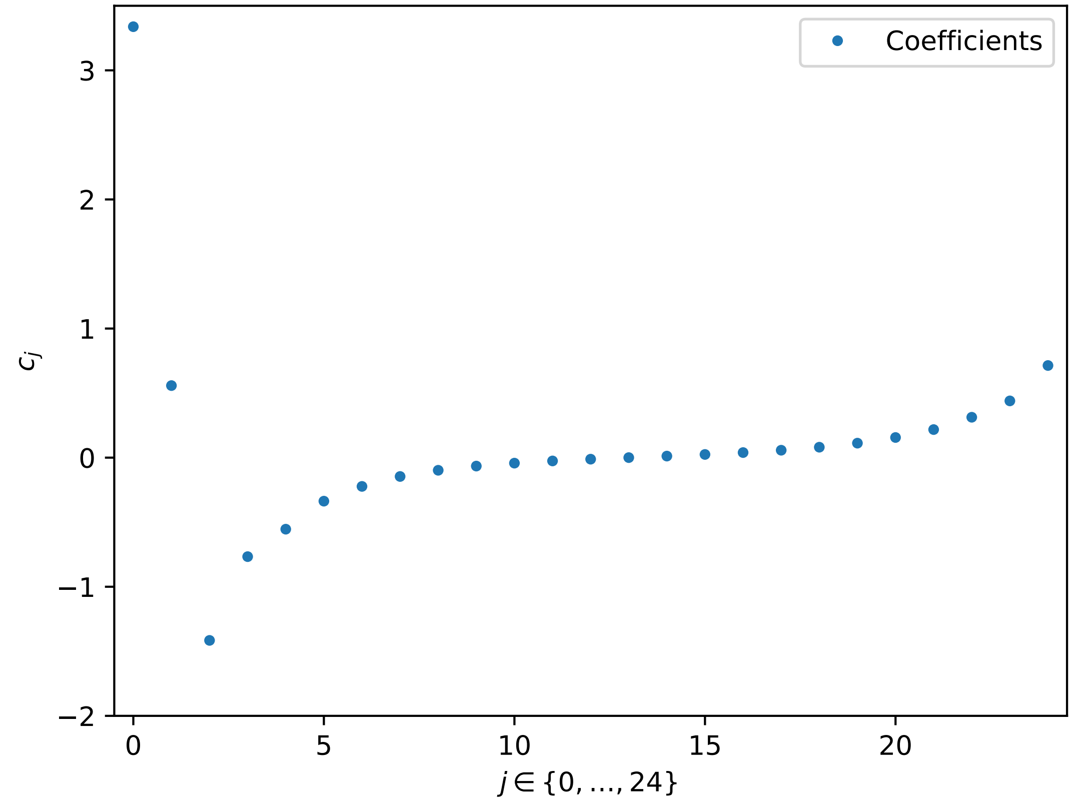
$$N(u) = \sum_{j=1}^{m} c_j \ \max(0, w_j u + b_j)$$

$m = 25 \qquad w_j = 1, \forall j \in \{1,\ldots,25\}$

$b_j = -(j-1)h, \quad j \in \{1,\ldots,25\}, \quad h = 3/50$

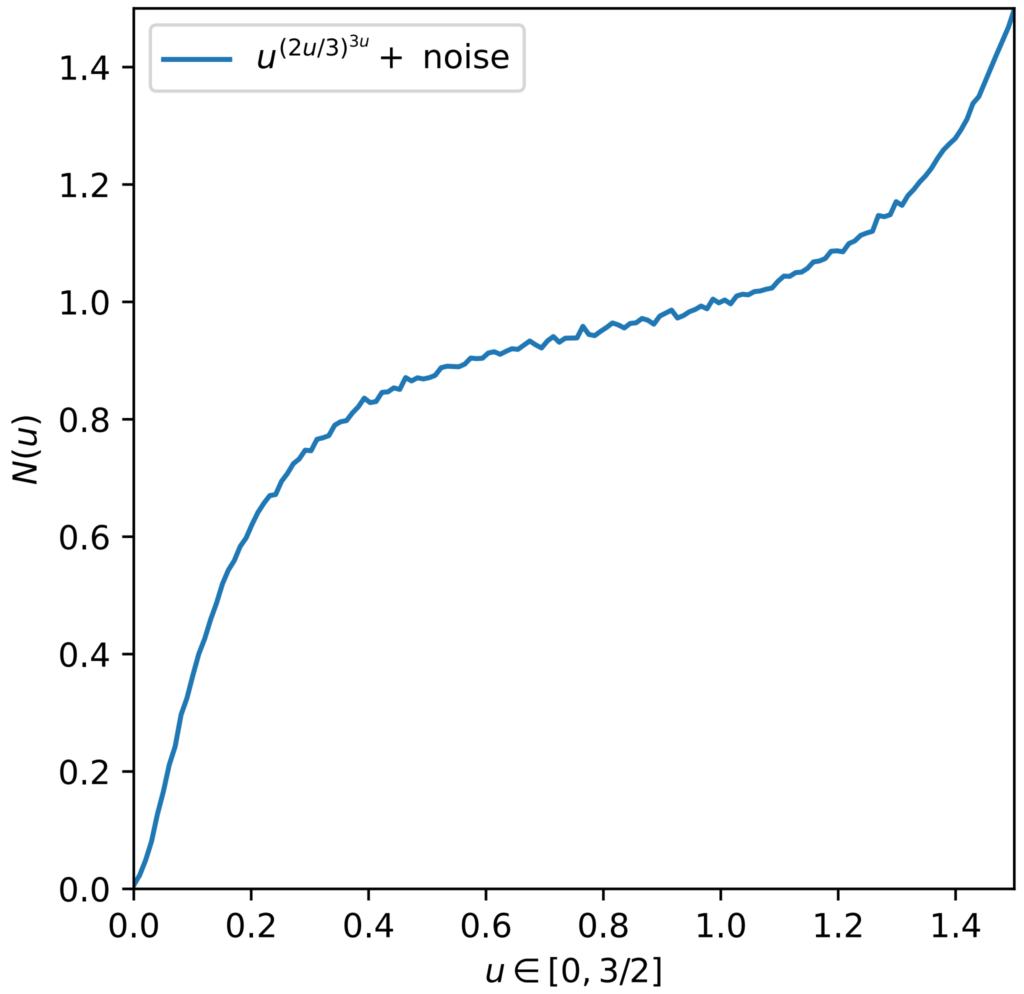
$\alpha = 10^{-4}$

Coefficients



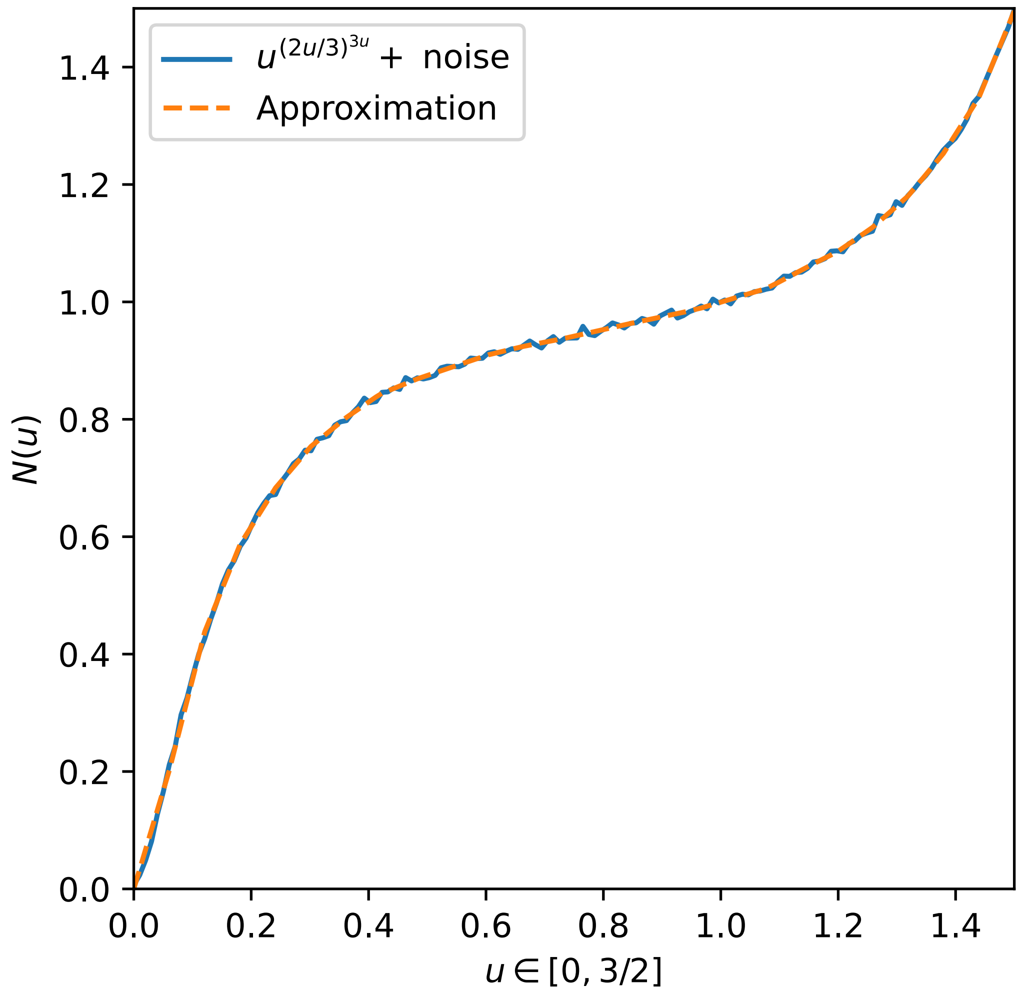
Inversion of neural networks

Example:



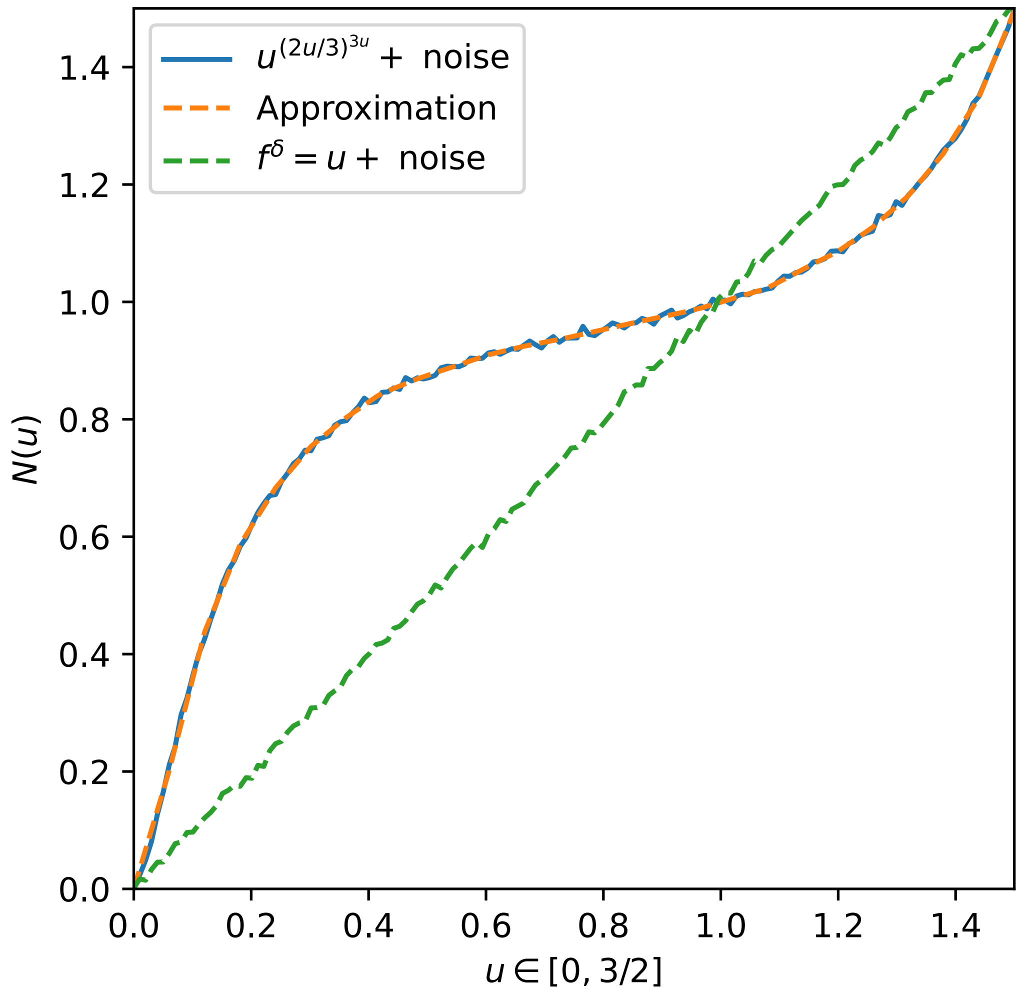
Inversion of neural networks

Example:



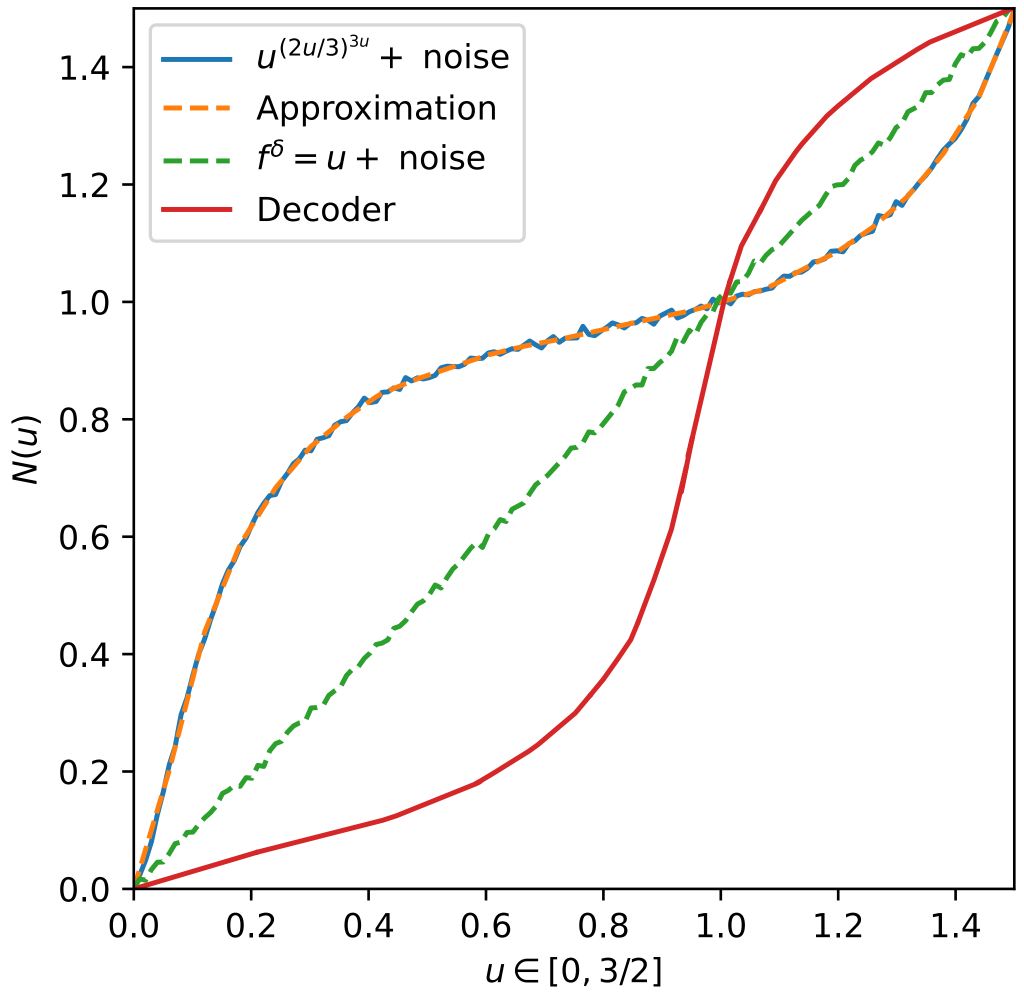
Inversion of neural networks

Example:



Inversion of neural networks

Example:



Inversion of neural networks

Example: Residual neural networks

$$N(u) = u^l$$

$$u^k = u^{k-1} + W_k^\top \mathrm{prox}_{\Psi_k}(W_k u_{k-1} + b_k) \qquad \forall\, k \in \{1, \ldots, l\}$$

Corresponding variational regularisation framework:

$$u_\alpha \in \arg\min_u \left\{ \frac{\lambda}{2}\|Ku - f^\delta\|^2 + B_\Psi(z, Wu + b) + J(u) \right\} \quad \text{subject to} \quad Mu = W^\top z$$

for

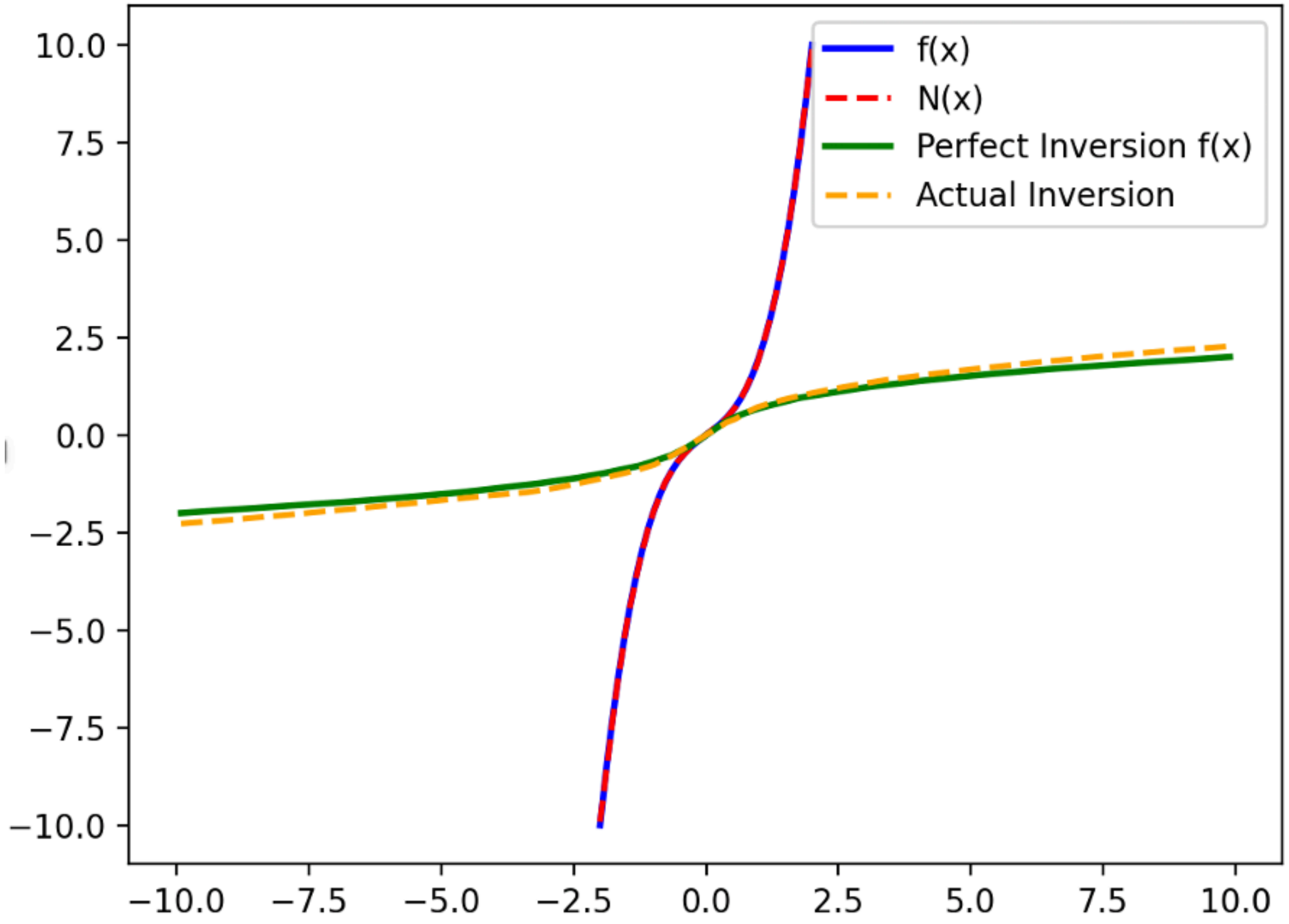
$$M = \begin{pmatrix} -I & I & 0 & \cdots & 0 \\ 0 & -I & I & & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ \vdots & & & -I & I \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \quad W = \begin{pmatrix} W_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & W_2 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & W_l & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_l \\ 0 \end{pmatrix} \quad \Psi(z_0, \ldots, z_l) = \sum_{k=0}^{l} \Psi_k(z_k)$$

Inversion of neural networks

Example:

$$l = 20$$

Function $K(u) = u^3 + u$



Inversion with theoretical guarantees?

Inversion with theoretical guarantees?

Can we provide some theoretical properties for the objective function

$$E_{\Psi}^{\rho}(\mathbf{u},\mathbf{z}) = \frac{\lambda}{2}\|\mathbf{K}\mathbf{u} - f^{\delta}\|^2 + B_{\Psi}(\mathbf{z}, \mathbf{W}\mathbf{u} + \mathbf{b}) + \chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) + \frac{\rho}{2}\|\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}\|^2$$

or the regularisation operator?

$$R(f^{\delta}) \in \arg\min_{u} \left\{ \frac{\lambda}{2}\|\mathbf{K}\mathbf{u} - f^{\delta}\|^2 + B_{\Psi}(\mathbf{z}, \mathbf{W}\mathbf{u} + \mathbf{b}) + \chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) + \frac{\rho}{2}\|\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}\|^2 + J(\mathbf{u}) \right\}$$

Inversion with theoretical guarantees?

Can we provide some theoretical properties for the objective function

$$E^{\rho}_{\Psi}(\mathbf{u},\mathbf{z}) = \frac{\lambda}{2}\|\mathbf{K}\mathbf{u} - f^{\delta}\|^2 + B_{\Psi}(\mathbf{z}, \mathbf{W}\mathbf{u} + \mathbf{b}) + \chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) + \frac{\rho}{2}\|\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}\|^2$$

$$R(f^{\delta}) \in \arg\min_{u} \left\{ \frac{\lambda}{2}\|\mathbf{K}\mathbf{u} - f^{\delta}\|^2 + B_{\Psi}(\mathbf{z}, \mathbf{W}\mathbf{u} + \mathbf{b}) + \chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) + \frac{\rho}{2}\|\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}\|^2 + J(\mathbf{u}) \right\}$$

Inversion with theoretical guarantees?

Can we provide some theoretical properties for the objective function

$$E_{\Psi}^{\rho}(\mathbf{u},\mathbf{z}) = \frac{\lambda}{2}\|\mathbf{K}\mathbf{u} - f^{\delta}\|^2 + B_{\Psi}(\mathbf{z}, \mathbf{W}\mathbf{u} + \mathbf{b}) + \chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) + \frac{\rho}{2}\|\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}\|^2$$

A sufficient condition for convexity is $\left\langle \partial E_{\Psi}^{\rho}(\mathbf{u}_1,\mathbf{z}_2) - \partial E_{\Psi}^{\rho}(\mathbf{u}_2,\mathbf{z}_2), \begin{pmatrix}\mathbf{u}_1\\ \mathbf{z}_1\end{pmatrix} - \begin{pmatrix}\mathbf{u}_2\\ \mathbf{z}_2\end{pmatrix} \right\rangle \geq 0.$

It can be shown that for $\mathbf{V} = \mathbf{W}^{\top}$ a sufficient condition for achieving this inequality for all $\mathbf{u}_1, \mathbf{u}_2, \mathbf{z}_1, \mathbf{z}_2$ is

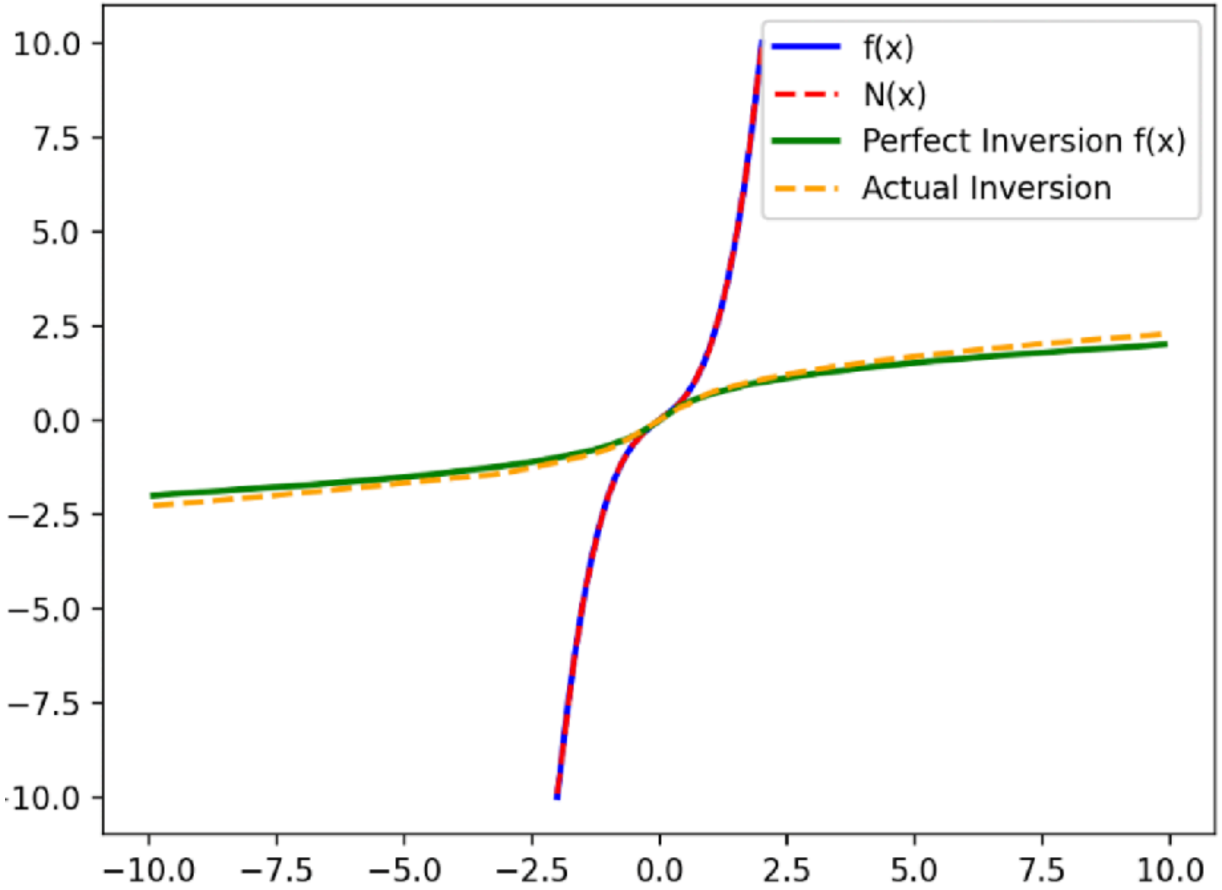
$$Q := \lambda K^{\top}K - \rho^{-1}I - M - M^{\top} \geq 0$$

Example: $u_j = u_{j-1} + W_{j-1}^{\top}\text{prox}_{\Psi_{j-1}}\left(W_{j-1}u_{j-1} + b_{j-1}\right) \implies$ Q is positive semi-definite

Inversion with theoretical guarantees?

Example: $$u_j = u_{j-1} - W_{j-1}^{\top} \mathrm{prox}_{\Psi_{j-1}} \left(W_{j-1} u_{j-1} + b_{j-1} \right)$$

$$f(x) = x + x^3$$



Inversion with theoretical guarantees?

$$E_{\Psi}^{\rho}(\mathbf{u},\mathbf{z}) = \frac{\lambda}{2}\|\mathbf{K}\mathbf{u} - f^{\delta}\|^2 + B_{\mathbf{\Psi}}(\mathbf{z},\mathbf{W}\mathbf{u}+\mathbf{b}) + \chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) + \frac{\rho}{2}\|\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}\|^2$$

or the regularisation operator?

$$R(f^{\delta}) \in \arg\min_{u} \left\{ \frac{\lambda}{2}\|\mathbf{K}\mathbf{u} - f^{\delta}\|^2 + B_{\mathbf{\Psi}}(\mathbf{z},\mathbf{W}\mathbf{u}+\mathbf{b}) + \chi_{=0}(\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}) + \frac{\rho}{2}\|\mathbf{M}\mathbf{u} - \mathbf{V}\mathbf{z}\|^2 + J(\mathbf{u}) \right\}$$

Inversion with theoretical guarantees?

$$E^{\rho}_{\Psi}(\mathbf{u},\mathbf{z}) = \frac{\lambda}{2}\|\mathbf{Ku} - f^{\delta}\|^2 + B_{\mathbf{\Psi}}(\mathbf{z}, \mathbf{Wu} + \mathbf{b}) + \chi_{=0}(\mathbf{Mu} - \mathbf{Vz}) + \frac{\rho}{2}\|\mathbf{Mu} - \mathbf{Vz}\|^2$$

or the regularisation operator?

$$R(f^{\delta}) \in \arg\min_{u} \left\{ \frac{\lambda}{2}\|\mathbf{Ku} - f^{\delta}\|^2 + B_{\mathbf{\Psi}}(\mathbf{z}, \mathbf{Wu} + \mathbf{b}) + \chi_{=0}(\mathbf{Mu} - \mathbf{Vz}) + \frac{\rho}{2}\|\mathbf{Mu} - \mathbf{Vz}\|^2 + J(\mathbf{u}) \right\}$$

No general results yet, but for

$$R(f^{\delta}) \in \arg\min_{u} \left\{ B_{\Psi}(f^{\delta}, Wu + b) + \alpha J(u) \right\}$$

we can show the following

Inversion with theoretical guarantees?

Theorem: suppose we have $f = \text{prox}_\Psi(Wu^\dagger + b)$ and $B_\Psi(f^\delta, Wu^\dagger + b) \leq \delta^2$ and $u^\dagger$ satisfies the source condition $W^\top v^\dagger \in \partial J(u^\dagger)$. Then, a solution $u_\alpha \in \arg\min_u \left\{ B_\Psi(f^\delta, Wu + b) + \alpha J(u) \right\}$ satisfies

$$D_J(u^\dagger, R(f^\delta)) \leq \underbrace{\frac{2\delta^2}{\alpha} + \alpha \|v^\dagger\|^2}_{\text{Classical error estimate}} + \frac{1}{\alpha} \underbrace{\left(\Psi\left(f^\delta + \alpha v^\dagger\right) + \Psi\left(f^\delta - \alpha v^\dagger\right) - 2\Psi(f^\delta) \right)}_{\text{Burbea Rao divergence between } f^\delta + \alpha v^\dagger \text{ and } f^\delta - \alpha v^\dagger}$$

Classical error estimate

Burbea Rao divergence between $f^\delta + \alpha v^\dagger$ and $f^\delta - \alpha v^\dagger$

Here D_J denotes the Bregman distance w.r.t. J.

Wang, X., & MB. A Lifted Bregman Formulation for the Inversion of Deep Neural Networks. *Front. Appl. Math. Stat.* 9, (2023).

Inversion with theoretical guarantees?

Example

$$\Psi(z) = \begin{cases} 0 & z \geq 0 \\ \infty & \text{else} \end{cases} \quad \Rightarrow \quad f = \max(0, Wu^\dagger + b)$$

$$\Rightarrow \quad \Psi\left(f^\delta + \alpha v^\dagger\right) + \Psi\left(f^\delta - \alpha v^\dagger\right) - 2\Psi(f^\delta) = 0 \qquad \text{if } v_j^\dagger \in \left[-\frac{f_j^\delta}{\alpha}, \frac{f_j^\delta}{\alpha}\right]$$

If we choose $\alpha(\delta) = \delta\sqrt{2}/\|v^\dagger\|$, then we observe

$$D_J\left(u^\dagger, u_{\alpha(\delta)}\right) \leq \underbrace{2\sqrt{2}\,\|v^\dagger\|}_{=C}\,\delta \quad \longrightarrow_{\delta \to 0} 0$$

Wang, X., & MB. A Lifted Bregman Formulation for the Inversion of Deep Neural Networks. *Front. Appl. Math. Stat.* 9, (2023).

Inversion with theoretical guarantees?

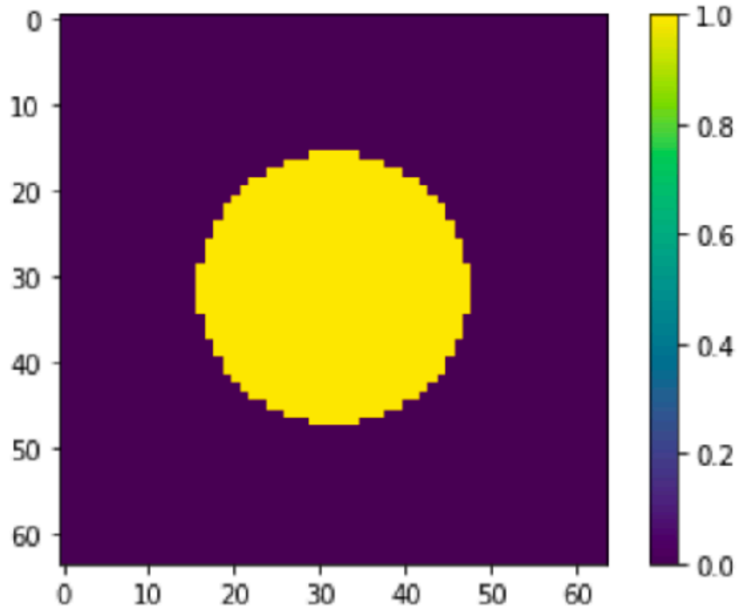
Example: ReLU Perceptron

$$N(u) = \max(0, Wu^\dagger + b)$$

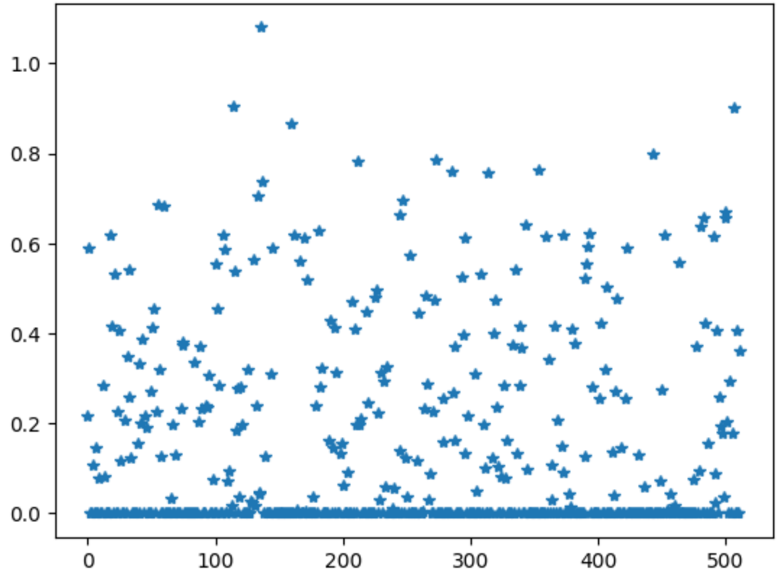
$W : \mathbb{R}^{64\times 64} \to \mathbb{R}^{512}$

$b \in \mathbb{R}^{512}$

Random entries
with zero mean and std one



Ground truth $u^\dagger$



Data f

Wang, X., & MB. A Lifted Bregman Formulation for the Inversion of Deep Neural Networks. *Front. Appl. Math. Stat.* 9, (2023).

Inversion with theoretical guarantees?

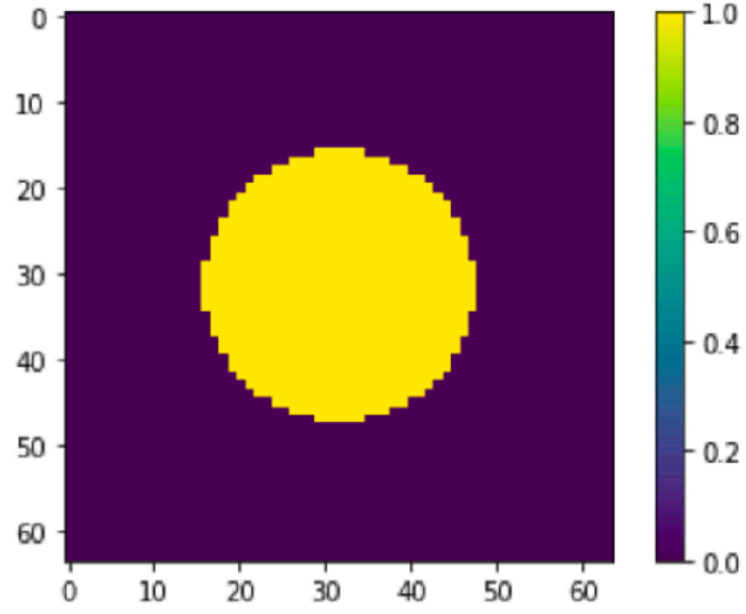
Example: ReLU Perceptron

$$N(u) = \max(0, Wu^\dagger + b)$$

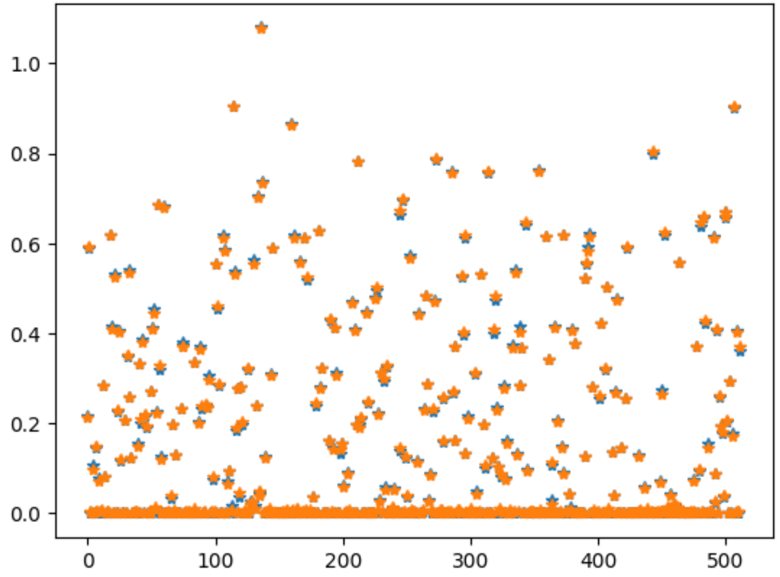
$W : \mathbb{R}^{64\times64} \to \mathbb{R}^{512}$

$b \in \mathbb{R}^{512}$

Random entries
with zero mean and std one



Ground truth $u^\dagger$



Data f^δ

Wang, X., & MB. A Lifted Bregman Formulation for the Inversion of Deep Neural Networks. *Front. Appl. Math. Stat.* 9, (2023).

Inversion with theoretical guarantees?

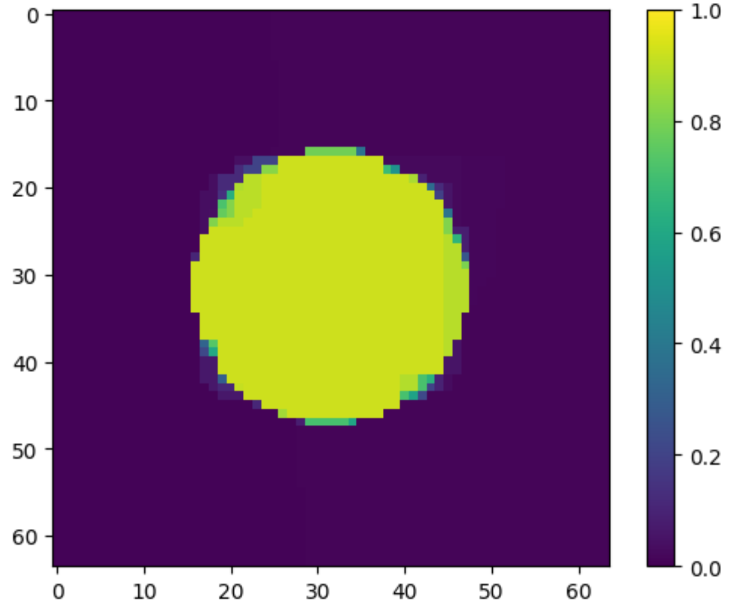
Example: ReLU Perceptron

$$N(u) = \max(0, Wu^\dagger + b)$$

$W : \mathbb{R}^{64\times 64} \to \mathbb{R}^{512}$

$b \in \mathbb{R}^{512}$

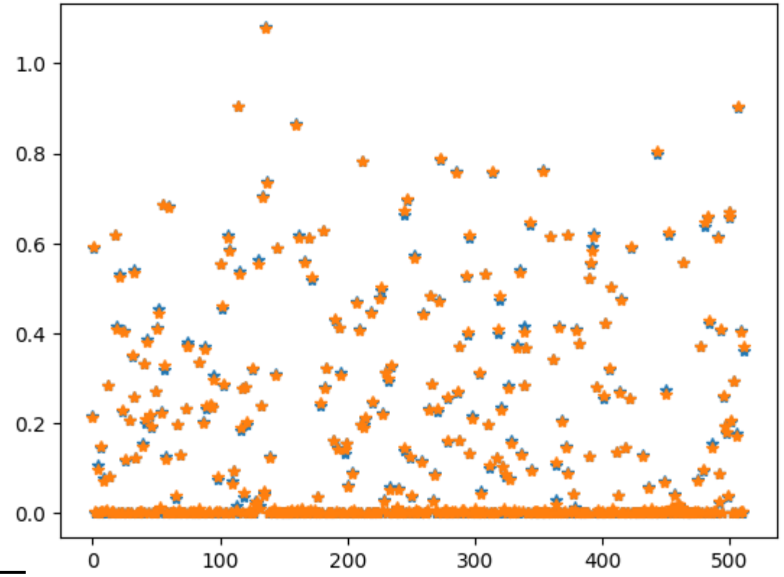
Random entries
with zero mean and std one



Reconstruction $R(f^\delta)$

$\alpha = 0.015$

$$J(u) = \sum_{i=1} \sum_{j=1} \sqrt{|\nabla u|^2_{i,j,1} + |\nabla u|^2_{i,j,2}}$$

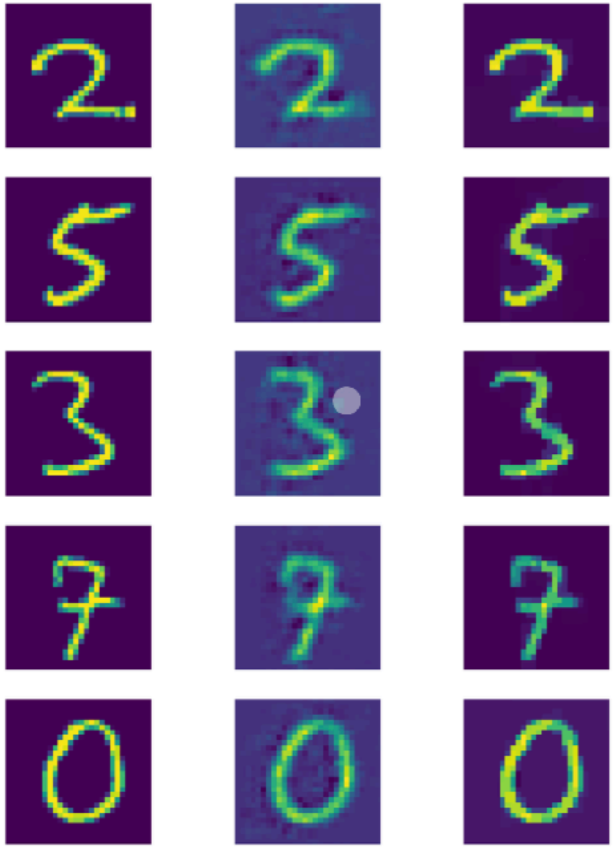


Data f^δ

Wang, X., & MB. A Lifted Bregman Formulation for the Inversion of Deep Neural Networks. *Front. Appl. Math. Stat.* 9, (2023).

Inversion of neural networks

Example: Six-layer convolutional autoencoder. Invert code with TV-based variational regularisation



$u^\dagger$ $\quad A(u^\dagger)$ $\quad R(f^\delta)$

$$J(u) = \sum_{i=1} \sum_{j=1} \sqrt{|\nabla u|^2_{i,j,1} + |\nabla u|^2_{i,j,2}}$$

$$\alpha = 9 \times 10^{-3}$$

Dimension of code is 300

Wang, X., & MB. A Lifted Bregman Formulation for the Inversion of Deep Neural Networks. *Front. Appl. Math. Stat.* 9, (2023).

Conclusions & outlook

Conclusions & outlook

Conclusions: we have

- introduced a novel lifted training approach for feed-forward networks
- shown that novel approach avoids differentiating activation functions
- shown that approach can be used for inversion of neural networks (decoder without training!)
- demonstrated that approach works empirically with numerical experiments
- proven that for one layer we have a convergent regularisation method

Outlook:

- Apply approach to real-world scenarios (blind deconvolution etc.)
- Extend concepts to different architectures
- Prove convergence results for architectures more complex than perceptrons
- Explore parallel or distributed computing frameworks

Thank you for your attention!

Acknowledgements: The Alan Turing Institute; CCIMI Cantab Capital Institute for the Mathematics of Information; Queen Mary University of London Digital Environment Research Institute; Isaac Newton Institute for Mathematical Sciences

Relevant open access research papers (more to come)

JMLR 24(232) 2023

Lifted Bregman Training

Front. Appl. Math. Stat. 9 2013

Lifted Bregman Inversion

Implementation

We minimise

$$E(\Theta, X) = \frac{1}{2s} \sum_{i=1}^{s} \sum_{l=1}^{L} B_{\Psi}\left(x_i^l, W_l x_i^{l-1} + b_l\right)$$

Implementation

We minimise

$$E(\Theta, X) = \frac{1}{2s} \sum_{i=1}^{s} \sum_{l=1}^{L} B_{\Psi}\left(x_i^l, W_l\, x_i^{l-1}\right)$$

Implementation

We minimise

$$E(\Theta, X) = \frac{1}{2s} \sum_{i=1}^{s} \sum_{l=1}^{L} B_\Psi\left(x_i^l, W_l\, x_i^{l-1}\right)$$

via a combination of an implicit stochastic gradient method*

$$(\Theta^{k+1}, X^{k+1}) = \arg\min_{\Theta, X} \left\{ \frac{1}{|S_p|} \sum_{i \in S_p} \left[\sum_{l=1}^{L} B_\Psi\left(x_l^i, W_l x_{l-1}^i\right) + \frac{1}{2\tau^k} \|W_l - W_l^k\|^2 \right] \right\}$$

with random batch S_p and proximal gradient descent** for the inner problem:

$$W_l^{j+1} = W_l^j - \frac{\gamma_l^j}{|S_p|} \left(\sum_{i \in S_p} \left[\sigma\left(W_l^j \left(x_{l-1}^i\right)_j\right) \left(x_{l-1}^i\right)_j^\top \right] + \frac{1}{\tau^k}\left(W_l^j - W_l^k\right) \right)$$

$$\left(x_l^i\right)^j = \mathrm{prox}_{\mu_l^j\left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)} \left(\left(x_l^i\right)^j - \mu_l^j \left(\left(W_l^j\right)^\top \left(\sigma\left(W_l^j \left(x_l^i\right)^j\right) - \left(x_{l+1}^i\right)^j \right) - W_l^j \left(x_{l-1}^i\right)^j \right) \right)$$

Implementation

We minimise E via a combination of an implicit stochastic gradient method*

$$(\Theta^{k+1}, X^{k+1}) = \arg\min_{\Theta, X} \left\{ \frac{1}{|S_p|} \sum_{i \in S_p} \left[\sum_{l=1}^{L} B_\Psi \left(x_l^i, W_l x_{l-1}^i \right) + \frac{1}{2\tau^k} \| W_l - W_l^k \|^2 \right] \right\}$$

with random batch S_p and proximal gradient descent** for the inner problem:

$$W_l^{j+1} = W_l^j - \frac{\gamma_l^j}{|S_p|} \left(\sum_{i \in S_p} \left[\sigma \left(W_l^j \left(x_{l-1}^i \right)_j \right) \left(x_{l-1}^i \right)_j^\top \right] + \frac{1}{\tau^k} \left(W_l^j - W_l^k \right) \right)$$

$$\left(x_l^i \right)^j = \text{prox}_{\mu_l^j \left(\frac{1}{2} \| \cdot \|^2 + \Psi \right)} \left(\left(x_l^i \right)^j - \mu_l^j \left(\left(W_l^j \right)^\top \left(\sigma \left(W_l^j \left(x_l^i \right)^j \right) - \left(x_{l+1}^i \right)^j \right) - W_l^j \left(x_{l-1}^i \right)^j \right) \right)$$

*Toulis, P., & Airoldi, E. M. (2017). Asymptotic and finite-sample properties of estimators based on stochastic gradients. *The Annals of Statistics*, 45(4), 1694–1727.
**Lions, P. L., & Mercier, B. (1979). Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, *16*(6), 964-979.

Implementation

We solve

$$x^\alpha \in \arg\min_x \left\{ B_\Psi(y^\delta, Wx + b) + \alpha R(x) \right\}$$

Implementation

We solve

$$x^\alpha \in \arg\min_x \left\{ B_\Psi(y^\delta, Wx + b) + \alpha \sum_{p=1} \sum_{q=1} \sqrt{\left|(\nabla x)_{p,q,1}\right|^2 + \left|(\nabla x)_{p,q,2}\right|^2} \right\}$$

Here, ∇x is a forward finite-difference discretisation of the gradient operator

We replace the regularisation function by its convex conjugate

$$x^\alpha \in \arg\min_x \left\{ B_\Psi(y^\delta, Wx + b) + \alpha \sup_z \left(\langle \nabla x, z \rangle - \left(\sum_{p=1} \sum_{q=1} \sqrt{|\cdot_{p,q,1}|^2 + |\cdot_{p,q,2}|^2} \right)^\star (z) \right) \right.$$

Implementation

We replace the regularisation function by its convex conjugate

$$x^\alpha \in \arg\min_x \left\{ B_\Psi(y^\delta, Wx + b) + \alpha \sup_z \left(\langle \nabla x, z \rangle - \left(\sum_{p=1} \sum_{q=1} \sqrt{|\cdot_{p,q,1}|^2 + |\cdot_{p,q,2}|^2} \right)^\star (z) \right) \right.$$

and solve this saddle-point problem with a generalised PDHG method*

$$x^{k+1} = x^k - \tau_x \left(W^\top \sigma \left(Wx^k + b \right) - y^\delta \right) - \alpha \mathrm{div} z^k \Big)$$

$$\tilde{z}^k = z^k + \tau_z \left(2\alpha \nabla x^{k+1} - \alpha \nabla x^k \right)$$

$$z^{k+1}_{p,q,d} = \tilde{z}^k_{p,q,d} / \max \left(1, \sqrt{\left| \tilde{z}^k_{p,q,1} \right|^2 + \left| \tilde{z}^k_{p,q,2} \right|^2} \right) \qquad \text{for } d \in \{1,2\}$$

*Chambolle, A., & Pock, T. (2016). An introduction to continuous optimization for imaging. *Acta Numerica*, *25*, 161-319.