# How to improve expressivity of convex $\mathrm{ReLU}$ neural networks?

Anne GAGNEUX[1] Mathurin MASSIAS[2] Emmanuel SOUBIES[3] Rémi GRIBONVAL[2]

[1]Ens de Lyon, CNRS, Université Claude Bernard Lyon 1, Inria, LIP, UMR 5668, 69342 Lyon Cedex 07, France

[2]Inria, Ens de Lyon, CNRS, Université Claude Bernard Lyon 1, LIP, UMR 5668, 69342 Lyon Cedex 07, France

[3]Université de Toulouse, CNRS, IRIT, UMR 5505, 31000, Toulouse, France

**Résumé –** Pour implémententer des fonctions convexes avec des réseaux de neurons, l'architecture standard est celle des ICNNs. Cet article présente une étude de leur expressivité, à la fois théorique et expérimentale. En s'appuyant sur la caractérisation des réseaux $\mathrm{ReLU}$ convexes, nous proposons une régularisation favorisant la convexité de la fonction apprise.

**Abstract –** To implement convex functions with neural networks, the standard way is to use Input Convex Neural Networks (ICNNs). This article provides a study of their expressivity, theoretically and experimentally. Leveraging the characterization of convex $\mathrm{ReLU}$ networks, we introduce a new regularisation which softly enforces convexity of the learnt function.

## 1 Introduction

The ability to implement convex functions with neural networks is crucial in many applications, from learning convex regularizers for inverse problems tasks [10] to learning optimal transport maps [8, 3]. The dominant approach, widely adopted by the community, is to use Input Convex Neural Networks (ICNNs) [1]. Their main advantage is that they require only slight changes to standard neural architectures, namely an additional nonnegativity constraint on the weight matrices. While this architecture is straightforward to implement and guarantees convexity by design, ICNNs demonstrate poor expressivity when scaling up [9]. In fact, the nonnegativity constraint of ICNNs can seem somewhat arbitrary, questioning whether it is the only way to enforce convexity or if there exist many more convex neural networks that are not ICNNs. The authors in [4] provide a first answer, showing that any convex function defined over a compact domain and implemented by a ReLU neural network, *i.e.*, a piecewise affine function, can also be implemented by an ICNN. Yet, their constructive proof yields an architecture which has as many layers as affine pieces in the function, and only one neuron per layer, thus far from a practical architecture. Following this work, we study in [5] the expressivity of ICNNs for *a given architecture* (*i.e.*, set width and depth). We fully characterize convex $\mathrm{ReLU}$ neural networks and show that there exist convex functions implemented by a given $\mathrm{ReLU}$ network that are *not implementable* by any ICNN with the same architecture.

Our paper first brings a pedagogical dive into the findings from [5]. Second, while [5] builds a theoretical framework for characterizing convexity in neural networks, this paper provides *empirical evidence* that ICNNs only constitute a small fraction of all convex $\mathrm{ReLU}$ neural networks. Additionally, we propose a new regularisation which acts as a soft constraint while training a $\mathrm{ReLU}$ network, promoting convexity of the learnt function.

**Notations** We denote by $N$ the set of all neurons of a neural network, and $\nu \in N$ is a neuron. We divide the set of all neurons between input neurons $N_{\text{in}}$, the (single) output neuron $N_{\text{out}}$ and the hidden neurons $H$ such that $N = N_{\text{in}} \cup H \cup N_{\text{out}}$. The parameters of a network are denoted $\theta$. The function $z_\nu(x, \theta)$ is the pre-activation of the neuron $\nu \in N$ at $x \in \mathbb{R}^d$.

## 2 Convex $\mathrm{ReLU}$ neural networks

### 2.1 Input Convex Neural Networks

The ICNN architecture, displayed in Figure 1, consists in a standard multilayer perceptron, with the additional constraint that the weight matrices of all hidden layers have nonnegative entries. ICNNs require convex and non-decreasing activation functions; in this work, only $\mathrm{ReLU}$ activations will be considered. Weighted skip connections linking the input to each hidden layer are also allowed. The convexity of ICNNs is based on rules for composition of convex functions: the composition $f \circ g$ of a convex function $g$ with a non-decreasing convex function $f$ is convex, and so is any linear combination of convex functions using nonnegative coefficients.

### 2.2 $\mathrm{ReLU}$ Neural Networks

Any function implemented by a $\mathrm{ReLU}$ neural network is a continuous and piecewise linear (CPWL) function, as composition of continuous piecewise linear functions. A CPWL function $f$ partitions the input space $\mathbb{R}^d$ into convex polyhedra –called *regions*– on which $f$ is affine.

For $\mathrm{ReLU}$ neural networks, a change in the slope of the implemented function $f$ at $x \in \mathbb{R}^d$ means that at least one neuron's pre-activation is zero at $x$, as zero is the only point of nondifferentiability of the $\mathrm{ReLU}$ function.

In this work, we adopt the definition of *activation regions* [7, Definition 1]. Consider a $\mathrm{ReLU}$ network with fixed parameters $\theta$. For each neuron $\nu \in N$, define its activation as $a_\nu(x, \theta) = \mathbf{1}_{z_\nu(x,\theta)>0} \in \{0, 1\}$. Then, activation regions are sets of input points on which all the activations $a_\nu$ are constant ($\nu \in H$; see Figure 2). [7, Lemma 1] shows that activation regions define a partition into convex regions for which $f$ is affine on each region.

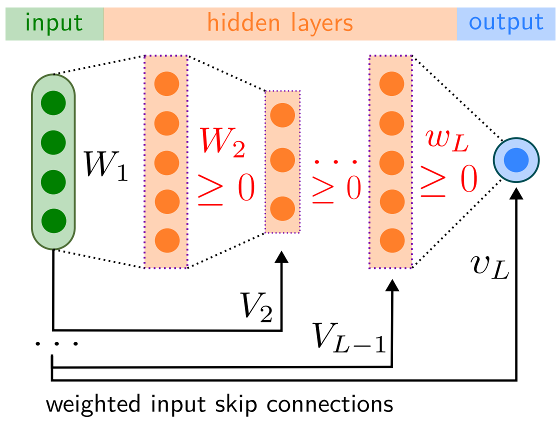


Figure 1 – The ICNN architecture.

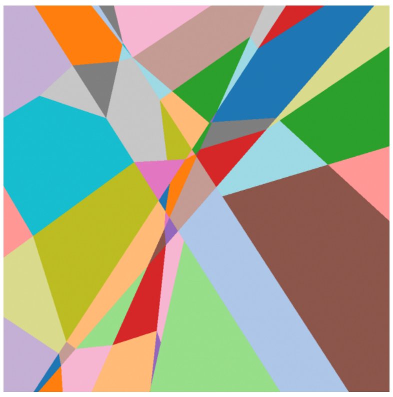


Figure 2 – Activation regions for a ReLU network with input dimension $d = 2$.

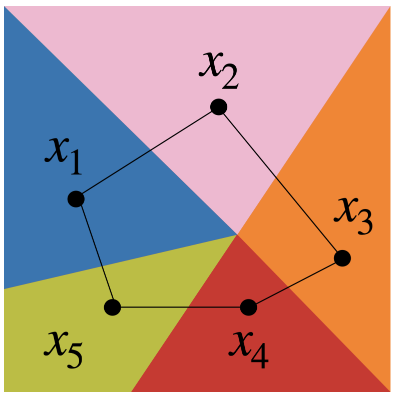


Figure 3 – Convexity must be checked along each black line, *i.e.*, between neigbouring regions only.

## 2.3 Convexity of CPWL functions

Intuitively, convexity of CPWL functions only needs to be checked at boundaries between regions. [5, Proposition 3.7] shows that it is in fact sufficient to check convexity only on boundaries between *neighboring regions*, defined as regions which share a boundary with affine dimension $d-1$. Specifically, this result states that a CPWL function $f$ is convex if and only if for every neighboring regions $R_1 \sim R_2$, there exists a pair of points $(x_1, x_2)$ with $x_1 \in \text{int}(R_1), x_2 \in \text{int}(R_2)$ such that

$$\langle \nabla f(x_1) - \nabla f(x_2), x_1 - x_2 \rangle \geq 0. \quad (1)$$

The key insight is that we identify here a *minimal* set of convexity conditions to be checked, as illustrated on Figure 3. This will be crucial for ReLU neural networks, as we seek the smallest set of constraints that still ensures convexity of the implemented function.

## 2.4 Convexity of ReLU neural networks

To go from characterization of convex CPWL functions to characterization of convex ReLU networks, a first step is to identify *neighboring activation regions*. Between two neighboring regions $R_1 \sim R_2$, at least one neuron's activation switches from active to unactive state. That is, there exists at least one hidden neuron $\nu \in H$ such that $a_\nu(x_1, \theta) = 1 - a_\nu(x_2, \theta)$ for $x_1 \in \text{int}(R_1), x_2 \in \text{int}(R_2)$. We will more specifically focus on points where only the activation of one specific neuron changes.

**Definition 2.1 (Isolated neurons [5])** *Given parameters $\theta$, for each hidden neuron $\nu \in H$, $\mathfrak{X}_\nu$ is the set of input points for which in every small enough neighborhood, only the activation of neuron $\nu$ switches:*

$$\begin{aligned} \mathfrak{X}_\nu := \{x \in \mathbb{R}^d : \exists \epsilon_0 > 0 : \forall 0 < \epsilon \leq \epsilon_0, \\ \forall \mu \neq \nu \quad a_\mu(\cdot, \theta) \textit{ is constant on } B(x, \epsilon), \\ a_\nu(\cdot, \theta) \textit{ is not constant on } B(x, \epsilon)\}. \end{aligned}$$

*A neuron $\nu \in H$ is isolated if $\mathfrak{X}_\nu \neq \emptyset$.*

In principle, this set $\mathfrak{X}_\nu$ should match the *frontiers*, *i.e.*, the relative interior of the boundaries between two regions with dimension $d-1$, which are points where convexity needs to be checked. Indeed, for input points at the intersection with more than two regions, at least two neurons' activations switch. However, there may be some cases where frontiers also involve multiple neurons switching simultaneously: we consider such cases as degenerate cases and exclude them from our analysis. We refer to [5, Section 7.3] for a discussion on the genericity of such an assumption.

### 2.4.1 The one-hidden-layer case

For one-hidden-layer networks, [5, Section 4] shows that convex ReLU networks and ICNNs implement the same set of functions.

**Proposition 2.2 (ICNNs are all you need)** *Any convex function implemented by a one-hidden-layer* ReLU *network with weighted input skip connections can also be implemented by a one-hidden-layer ICNN with the same width.*

### 2.4.2 The two-hidden-layer case

For two-hidden-layer networks, the previous results no longer hold: [5, Proposition 2.1] gives an example of a convex function implemented by a two-hidden-layer ReLU network which *is not implementable by any ICNN with the same architecture*. In fact, the characterization of two-hidden-layer convex ReLU networks shows that a network is convex if and only if some *sum of products of weights* are nonnegative: this condition is less restrictive than the ICNN constraint enforcing every hidden layer weight to be nonnegative.

### 2.4.3 General architectures

Consider a neural network architecture with parameters $\theta$, implementing a function $f : \mathbb{R}^d \to \mathbb{R}$, and described by a Directed Acyclic Graph (DAG) $G = (N, E)$ with vertices $N$ representing neurons and edges $E$ representing weights. To study convexity of $f$, we use the path-norm toolkit [6], which allows us to rewrite the output of the network $f(x, \theta)$ for any input $x \in \mathbb{R}^d$ as

$$f(x, \theta) = \left\langle \Phi(\theta), A(x, \theta) \begin{pmatrix} x \\ 1 \end{pmatrix} \right\rangle. \quad (2)$$

The vector $\Phi(\theta)$ is called the *path-lifting* [6]; it has as many rows as there are paths in $G$ connecting some neuron $\nu$ (either an input neuron or a hidden neuron) to the (unique) output neuron. The value for each row is given by the product of weights along the path. The matrix $A(x,\theta)$ corresponds to *the path-activations*. It has as many rows as paths and $d+1$ columns where $d$ is the input dimension. For each path *$p$ starting at an input neuron* $\mu \in N_{\text{in}}$, $A(x,\theta)_{p,\mu}$ is $1$ if the path is activated, $0$ otherwise – a path $p$ being activated if each neuron along the path is activated, *i.e.*, $A(x,\theta)_p = \prod_{\nu\in p} a_\nu(x,\theta)$. For each path *$p$ starting at an hidden neuron* $\mu \in H$, $A(x,\theta)_{p,d+1}$ is $1$ if the path is activated, $0$ otherwise. Precise definitions are given both in [6, 5], as well as how these tools can take into account convolutional layers, max-pooling, *etc*. Figure 4 illustrates the construction of these objects for a simple one-hidden-layer network with $d=1$ ($x\in\mathbb{R}$).

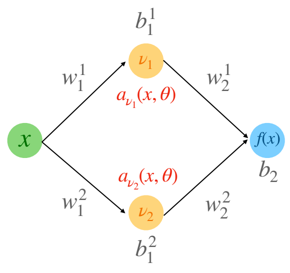


Figure 4 – Replacing the $\mathrm{ReLU}$ by the neurons activation, $f(x,\theta) = w_2^1 a_{\nu_1}(x,\theta)(w_1^1 x + b_1^1) + w_2^2 a_{\nu_2}(x,\theta)(w_1^2 x + b_1^2) + b_2$. So $f$ rewrites as the sum of 5 terms. The path-lifting thus writes as $\Phi(\theta) = (w_1^1 w_2^1, b_1^1 w_2^1, w_1^2 w_2^2, b_1^2 w_2^2, b_2)^\top$.

Similar objects can be built for sub-graphs of $G$. We denote $\Phi^{\nu\rightarrow}(\theta)$ the path-lifting associated with the sub-graph $G^{\nu\rightarrow}$ extracted from $G$ with $\nu$ as input neuron and the same output neuron as $G$ (see Figure 5).

The path-norm toolkit is useful to study convexity as the convexity conditions given by Equation 1 involve the slopes of $f(\cdot,\theta)$ for each activation regions, on which $A(\cdot,\theta)$ is constant. For $x_1 \in R_1$ where $R_1$ is an activation region with constant path-activations matrix $A_1$, we deduce from (2) that

$$\nabla f(x_1) = A_1^\top \Phi(\theta). \tag{3}$$

Under some non-degeneracy assumption [5, Assumption 7.1], convex $\mathrm{ReLU}$ neural networks are characterized by [5, Theorem 7.4], which we recall below.

**Theorem 2.3 (Characterization of convex $\mathrm{ReLU}$ networks)** *Consider a $\mathrm{ReLU}$ network with parameters $\theta$, implementing a function $f_\theta : \mathbb{R}^d \to \mathbb{R}$ and described by a DAG. The function $f_\theta$ is convex if and only if for every hidden neuron $\nu \in H$, it holds*

$$\min_{a\in \mathbf{a}^{\nu\rightarrow}} \langle a, \Phi^{\nu\rightarrow}(\theta)\rangle \geq 0, \tag{4}$$

*where* $\mathbf{a}^{\nu\rightarrow} := \{a^{\nu\rightarrow}(x,\theta) : x \in \mathfrak{X}_\nu\}$ [1].

Intuitively, each binary vector $a \in \mathbf{a}^{\nu\rightarrow}$ corresponds to an activation pattern for a $d-1$ dimensional boundary between *neighboring activations regions* for which the activation of $\nu$ switches. Because each vector $a$ is binary, these convexity conditions involve checking nonnegativity of *sums of product of weights*. If all elementary vectors $(0,\dots,0,1,0,\dots,0)^\top$ are in $\mathbf{a}^{\nu\rightarrow}$ for all $\nu \in H$, then these constraints reduce to the ICNN constraints as the scalar product only involve one term. On the contrary, when enough elementary vectors are missing, the constraints (4) become less restrictive than the ICNN ones.

1. The notation $^{\nu\rightarrow}$ relates to the largest sub-graph extracted from $G$ with $\nu$ as single input.

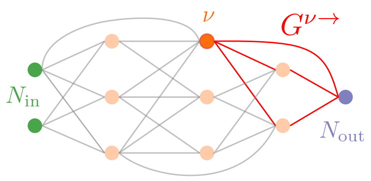


Figure 5 – Sub-graphs extracted from $G$.

## 3 Algorithm

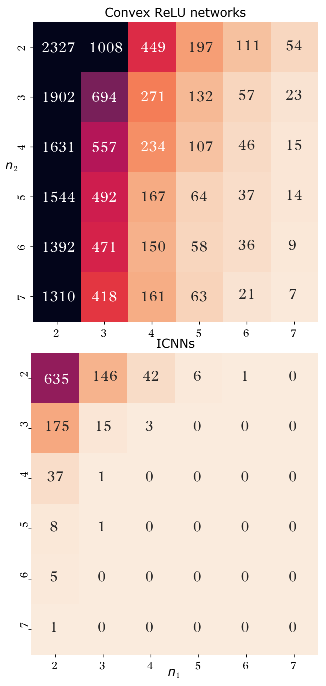
Given a $\mathrm{ReLU}$ neural network with architecture $G = (N,E)$, the two main bottlenecks to check the convexity conditions of Theorem 2.3 are:

1. **Handling the dimension of the vectors $\Phi^{\nu\rightarrow}$ and $a^{\nu\rightarrow}$** involved in the scalar product. Considering *feedforward* networks, the number of paths grows exponentially with the number of layers, and in general with the depth of the network, so this is *a priori* a daunting task. [5, Section 8] explains how this can be easily addressed. Each condition can be evaluated with a forward pass on a slightly modified neural network, circumventing the need to explicitly compute and store the path-lifting vectors $\Phi^{\nu\rightarrow}(\theta)$.
2. **Computing the set of activation patterns $\mathbf{a}^{\nu\rightarrow}$** for every hidden neuron $\nu \in H$. This requires identifying all input points $x \in \mathfrak{X}_\nu$ where only the activation of $\nu$ changes. In practice, it amounts to identifying all frontiers and regions of the CPWL function, whose number grows exponentially with the number of neurons. We rely on [2] which provides an efficient algorithm to identify the frontiers of the CPWL function implemented by a feedforward fully-connected $\mathrm{ReLU}$ network.

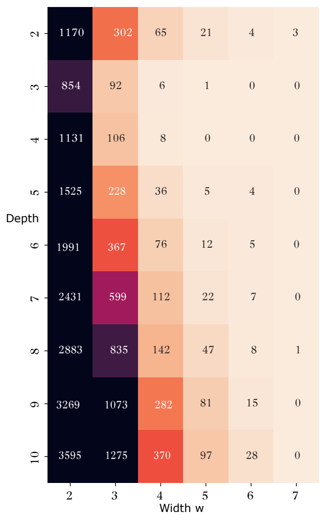
## 4 Experiments

### 4.1 Number of convex networks

To assess the limitations of ICNNs, we compare the number of convex $\mathrm{ReLU}$ networks to the number of ICNNs when sampling random networks. We provide several experiments, varying the width (Figure 6a), the depth (Figure 6b) and the input dimension of the network. The weights of the last layer are sampled independently from a folded normal distribution, ensuring their nonnegativity (as it is a necessary condition for convexity). The weights of the other hidden layers are sampled from a normal distribution, and may thus have negative entries. We observe (not reported) that the input dimension has no impact on the number of sampled convex $\mathrm{ReLU}$ networks or ICNNs. This is expected for ICNNs, as the nonnegativity constraint is applied to the hidden layers, making it independent of the input dimension. Figure 6a compares the number of random two-hidden-layer $\mathrm{ReLU}$ networks that are convex with the number of ones that are ICNNs, when varying the widths of each hidden layer. Likewise, Figure 6b displays the number of random networks that are convex when increasing the depth, *i.e.*, the number of hidden layers. We do not report the number of ICNNs among these convex $\mathrm{ReLU}$ networks as it is consistently zero beyond three hidden layers. In both cases, we observe that convex $\mathrm{ReLU}$ networks largely outnumber ICNNs. The number of convex $\mathrm{ReLU}$ networks varies in assymetric manners. Somewhat unexpectedly, it increases



(a) **Width varies.** Convex ReLU networks among $10k$ draws. Top: convex ReLU networks. Bottom: ICNNs. Architecture $\mathbf{n} = (n_1, n_2)$. Input dimension $d = 2$.



(b) **Depth varies.** Convex ReLU networks among $5k$ draws. $\mathbf{n} = (w, w, \dots, w)$. Input dimension $d = 3$.

Figure 6 – Comparison of convex ReLU networks by varying width and depth.

with depth for a given width. Besides, there are more convex networks in "decoder-like" architectures ($n_2 > n_1$) than in "encoder-like" architectures.

## 4.2 Convex regularisation

For a ReLU network with parameters $\theta$, a natural regularisation that pushes towards convex networks is obtained by penalizing the convexity conditions' values which are negative as follows

$$\mathcal{R}(\theta) := \lambda \sum_{\nu \in H} \sum_{a \in \mathbf{a}^{\nu\rightarrow}} \max(0, -\langle a, \Phi^{\nu\rightarrow}(\theta) \rangle). \qquad (5)$$

To assess the potential of such regularisation, we test it on a 2D CPWL function which we approximate by sampling uniform points on the square $[-10, 10]^2$ and training the ReLU networks with an $\ell_1$ loss. We compare an unconstrained ReLU network (which may not be convex), an ICNN and a regularised ReLU network, all with architecture $\mathbf{n} = (4, 8)$ 2. These networks share the same random initialisation. Figure 7 display the learnt functions while Table 1 gives the mean square error on $N_{\text{val}} = 20K$ uniformly sampled points.

| Model | Unconstrained | ICNN | Convex Reg |
|---|---|---|---|
| MSE | $0.90 \pm 0.53$ | $1.71 \pm 0.90$ | $0.84 \pm 0.73$ |

Table 1 – Approximation results. The standard deviation is given on 10 initialisations.

2. $\mathbf{n}$ denotes the tuple of widths of hidden layers.

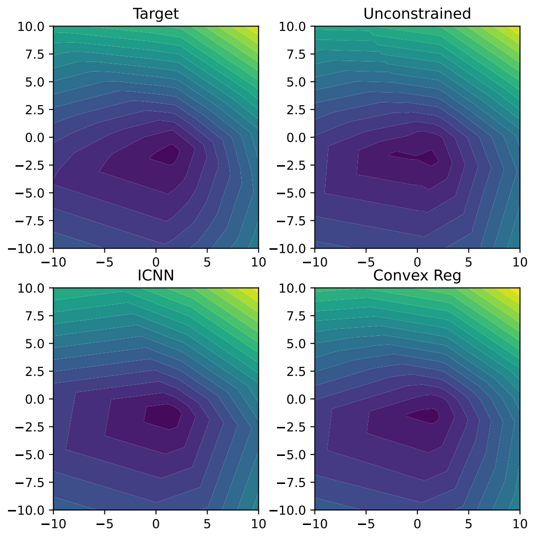


Figure 7 – Target and learnt functions. The regularisation is a compromise between the lack of expressivity of ICNNs and the convexity requirement.

## 5 Conclusion

Both the theoretical characterization of convex ReLU networks and our numerical experiments confirm there is room for improvements upon the ICNN architecture. In this paper, we suggest one way to do so, by using the characterization of convex networks as a soft regularisation to promote convexity of the network. A computational challenge is to scale it to practical networks, as this would open new ways to train more expressive convex networks.

**Acknowledgements** R. Gribonval is grateful to Antonin Chambolle for enlightening discussions on related topics during the SMAI-MODE 2024 conference. This project was supported in part by the AllegroAssai ANR project ANR-19-CHIA0009 and by the SHARP ANR project ANR-23-PEIA-0008 funded in the context of the France 2030 program. A. Gagneux thanks the RT IASIS for supporting her work through the PROSSIMO grant.

## References

[1] Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. 2017.

[2] Arturs Berzins. Polyhedral Complex Extraction from ReLU Networks using Edge Subdivision. 2023.

[3] Charlotte Bunne, Stefan G Stark, Gabriele Gut, Jacobo Sarabia Del Castillo, Mitch Levesque, Kjong-Van Lehmann, Lucas Pelkmans, Andreas Krause, and Gunnar Rätsch. Learning single-cell perturbation responses using neural optimal transport. *Nature methods*, 20(11):1759–1768, 2023.

[4] Yize Chen, Yuanyuan Shi, and Baosen Zhang. Optimal Control Via Neural Networks: A Convex Approach. In *ICLR*, 2019.

[5] Anne Gagneux, Mathurin Massias, Emmanuel Soubies, and Rémi Gribonval. Convexity in ReLU Neural Networks: beyond ICNNs?, 2025.

[6] Antoine Gonon, Nicolas Brisebarre, Elisa Riccietti, and Rémi Gribonval. A path-norm toolkit for modern networks: consequences, promises and challenges. *ICLR*, 2023.

[7] Boris Hanin and David Rolnick. Deep ReLu Networks Have Surprisingly Few Activation Patterns. *Neurips*, 2019.

[8] Alexander Korotin, Vage Egiazarian, Arip Asadulaev, Alexander Safin, and Evgeny Burnaev. Wasserstein-2 generative networks. *arXiv preprint arXiv:1909.13082*, 2019.

[9] Alexander Korotin, Daniil Selikhanovych, and Evgeny Burnaev. Neural optimal transport. 2023.

[10] S. Mukherjee, S. Dittmer, Z. Shumaylov, S. Lunz, O. Öktem, and C.-B. Schönlieb. Data-driven convex regularizers for inverse problems. 2024.