

Computing frustration and near-monotonicity in deep neural networks

Joel Wendin,^{*} Erik G. Larsson,[†] and Claudio Altafini^{*‡}

October 6, 2025

Abstract

For the signed graph associated to a deep neural network, one can compute the frustration level, i.e., test how close or distant the graph is to structural balance. For all the pretrained deep convolutional neural networks we consider, we find that the frustration is always less than expected from null models. From a statistical physics point of view, and in particular in reference to an Ising spin glass model, the reduced frustration indicates that the amount of disorder encoded in the network is less than in the null models. From a functional point of view, low frustration (i.e., proximity to structural balance) means that the function representing the network behaves near-monotonically, i.e., more similarly to a monotone function than in the null models. Evidence of near-monotonic behavior along the partial order determined by frustration is observed for all networks we consider. This confirms that the class of deep convolutional neural networks tends to have a more ordered behavior than expected from null models, and suggests a novel form of implicit regularization.

Introduction

Even though current deep neural networks (DNN) can achieve human-like performances in fields such as image classification, speech recognition and language generation, the reasons and properties underpinning these successes remain largely unclear. Factors such as overparametrization and the possibility of training on massive datasets are certainly playing a role, but alone do not provide a justification of the generalization power of DNNs, i.e., of how even elementary training algorithms like gradient descent tend to produce models that generalize well to new data without signs of overfitting, even in absence of any explicit regularization.

Very little is known in general of the organization principles that enable a DNN to function so efficiently. In simpler settings such as deep linear networks [1, 4, 8, 32, 43], it is known for instance that lack of convexity in the loss function does not imply existence

^{*}Division of Automatic Control, Dept. of Electrical Engineering, Linköping University, SE-58183, Linköping, Sweden.

[†]Division of Communication Systems, Dept. of Electrical Engineering, Linköping University, SE-58183, Linköping, Sweden.

[‡]Corresponding author (email: claudio.altafini@liu.se)

of local minima in the loss landscape [21], which guarantees that a reasonably trained linear network typically achieves performances that are optimal within the class of linear models. Nevertheless, even in this simplified setting, it is unclear if there is any principle governing the “inner functioning” of the trained network. In matrix factorization problems, for instance, the models seem to obey a “greedy low-rank” principle, at least when they are initialized near the origin [10, 23]. In the nonlinear case, the theoretical understanding is even more limited. For classification problems with separable data, and binary classification on non-separable data, it appears that a trained model obeys to a maximum-margin principle [30, 37, 20], but how to extend this concept to more general settings is unclear. The paper [9] discusses the loss landscape of general deep neural networks with ReLU activations, using results from random matrix theory applied to spherical spin glasses. In particular, it shows that, under certain assumptions, the low-value critical points of the DNN are located into a narrow band lower bounded by the global minimum, and that the number of local minima outside this band decreases exponentially with the size of the networks. As shown first in [44], current large DNNs can achieve zero training error also on randomized labels, expressing the inadequacy of a criterion based on the loss function as a measure of complexity of a model. This issue is investigated further in e.g. [28, 41]. No satisfactory complexity measure has so far been found for DNNs.

This paper provides a novel ingredient which can shed some light on the internal organization of a trained DNN. To do so, we combine notions from Statistical Physics of complex networks, namely the idea of frustration and structural balance from Ising spin glass theory, with the idea of monotonicity of a function, here the DNN map. The theory of spin glasses is one of the classical theories used to model complex disordered systems in Statistical Physics [27, 6], and has been invoked repeatedly to deal with different aspects of neural networks, see e.g. [2, 9, 12, 26]. Here rather than the spherical spin glass model of [9], our reference is to the Ising spin glass model, in which the spin variables at the nodes can assume only a binary value (± 1), instead of any value on a sphere [6, 27, 26]. Unlike the spherical model, the Ising spin glass model is characterized by the presence of *frustration* [39], a graphical property that is taken as a paradigm for the disorder encoded in the spin glass. Frustration-free networks are disorder-free, and are characterized by graphs that are *structurally balanced*, meaning that all (undirected) cycles have an even number of negative edges [15]. When the graph is a directed acyclic graph (DAG) as in DNNs, the original notion from Ising spin glasses (where edge weights represent ferromagnetic and antiferromagnetic bonds and are typically indirect and binary, ± 1) can be extended to weighted signed digraphs without introducing ambiguities. These broader concepts of frustration and structural balance have been used in the last decade beyond spin glass theory, for general complex networks [13], and applied to other disciplines like Biology [17, 25], Social Sciences [14, 42], etc. In addition, it has been shown that there is a strict relation between the graphical property of structural balance and a property of dynamical systems called monotonicity [35]. The Jacobian of a monotone dynamical system is in fact structurally balanced everywhere in its state space. This relation can be extended from dynamical systems to static functions, like the input-output map of a DNN. Monotone DNN have been studied thoroughly because they have the property of being interpretable: increasing a certain input feature is guaranteed to increase/decrease a certain output feature [7, 24, 31, 33, 34]. Exploiting the Jacobian of the DNN, the extension of monotonicity from an input-output to an internal (hidden) state perspective

is rather straightforward, and allows to verify monotonicity through the structural balance of the DAG associated to the DNN.

Monotone DNN are however very unlikely to appear when training from large-scale real data, just like signed networks of any kind are unlikely to be exactly structurally balanced. A signed graph which is not structurally balanced has some amount of frustration. In Ising spin glass theory, this is quantified by computing the ground state of a Hamiltonian energy functional [27], a problem known to be NP hard [5]. Efficient heuristics, inspired by this notion and based on gradient descent principles, were presented e.g. in [18]. When investigating the amount of frustration present in diverse types of networks from fields such as Biology and Social Sciences, it has often been observed that the networks are less frustrated than expected from null models [14, 18, 17, 25, 40], which can be interpreted as near-monotonicity of the networks [36]. In other words, these networks appear to encode less disorder than expected from randomizations, and to behave more predictably than completely random models.

The task of this paper is to check if something similar may be occurring in DNNs. In order to do so we consider state-of-the-art DNNs, like the modern deep convolutional neural networks (CNN) used for image recognition. In particular, we focus on several pretrained CNNs, such as SqueezeNet, ShuffleNet, Alexnet, GoogLeNet, ResNet, and compute the associated DAGs. On the resulting DAGs, we calculate the level of frustration and compare it with several types of null models, obtained by preserving the convolutional structure at the layers but reshuffling the position of the parameters in the filters, or reshuffling them in a more drastic way that destroys also the convolutional pattern.

All the pretrained CNNs we consider have a certain amount of frustration, which is however always less than that of the null models, and the difference is statistically significant. Consequently, all pretrained CNNs are expected to have a behavior which is closer to monotone than the associated null models. For our CNNs this is indeed the case: the outputs of the CNNs exhibit a marked tendency to respond to input perturbations by aligning themselves with the partial order direction predicted by our ground state calculations. This near-monotone tendency is statistically significant for all the CNNs we consider.

Overall, this suggests that the internal organization of a trained CNN encodes some order, which induces a near-monotone input-output behavior never observed before. We believe it can be considered a novel form of implicit regularization, i.e., of mathematical simplicity not imposed explicitly but emerging from the training process.

Methods

In this section we review the concepts of structural balance and monotonicity, and show how to quantify the distance to structural balance/monotonicity through the notion of frustration.

Structural balance

Consider a signed digraph $\mathcal{G}(A)$, where $A \in \mathbb{R}^{n \times n}$ is the associated weighted adjacency matrix, i.e., $A_{ij} \neq 0$ iff $\mathcal{G}(A)$ has an edge from node j to node i . The edge weight $A_{ij} \neq 0$

can be either a positive or negative number. Given $\mathcal{G}(A)$, we denote $\mathcal{G}_u(A)$ the associated undirected graph obtained by dropping the arrow on the edges: $\mathcal{G}_u(A) = \mathcal{G}(A_u)$, where $A_u = A + A^\top$ is the symmetrized version of A . Since we deal only with DAGs, this operation can always be done without creating ambiguity. The digraph $\mathcal{G}(A)$ is said *structurally balanced* (or *frustration-free*) if in its associated undirected graph $\mathcal{G}_u(A)$ all cycles are positive, i.e., they have an even number of negative edges [15].¹ If $\mathcal{G}(A)$ is structurally balanced, then there exists a partition of its nodes into two sets $\mathcal{V}_1$ and $\mathcal{V}_2$ s.t. $\mathcal{V}_1 \cup \mathcal{V}_2 = \{1, \dots, n\}$ and $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$, for which $A_{ij} \geq 0$ if $i, j \in \mathcal{V}_1$ or $i, j \in \mathcal{V}_2$, and $A_{ij} \leq 0$ if $i \in \mathcal{V}_1$ and $j \in \mathcal{V}_2$, or $i \in \mathcal{V}_2$ and $j \in \mathcal{V}_1$. Another equivalent characterization of structural balance is the existence of a diagonal signature matrix $S = \text{diag}(\mathbf{s})$, where $\mathbf{s} = [s_1 \dots s_n]$ with $s_i = \pm 1$ for all $i = 1, \dots, n$, s.t. $SAS \geq 0$ [14]. S is sometimes referred to as a gauge transformation matrix in the Statistical Physics literature. The sign of the cycles of $\mathcal{G}_u(A) = \mathcal{G}(A_u)$ is invariant to gauge transformations by $S = \text{diag}(\mathbf{s})$. In fact, if $s_i = -1$ and $s_j = +1$ for all $j \neq i$, then the effect of the gauge transformation is to flip the sign of all entries in the i -th row and column of A_u , which leaves the sign of the cycles in which the node i is involved (and of all other cycles as well) invariant, since each (simple) cycle involves 2 edges adjacent to i .

Monotonicity

To introduce the notion of monotonicity we use in this paper, we need to consider the idea of partial ordering induced by an orthant of $\mathbb{R}^n$. Consider the gauge transformation matrix S and denote $\mathbb{S}$ the orthant of $\mathbb{R}^n$ identified by S : $\mathbb{S} = S\mathbb{R}^n = \{\mathbf{z} \in \mathbb{R}^n \text{ s.t. } s_i z_i \geq 0, i = 1, \dots, n\}$. $\mathbb{S}$ determines the following partial ordering in $\mathbb{R}^n$, denoted “ $\leq_{\mathbb{S}}$ ”: $\mathbf{z}_1 \leq_{\mathbb{S}} \mathbf{z}_2$ iff $\mathbf{z}_2 - \mathbf{z}_1 \in \mathbb{S} \forall \mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^n$.

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is said *monotone with respect to the partial order $\mathbb{S}$* (for short $\mathbb{S}$ -monotone, or simply *monotone* when there is no need to specify the partial order $\mathbb{S}$) if $\mathbf{z}_1 \leq_{\mathbb{S}} \mathbf{z}_2$ implies $f(\mathbf{z}_1) \leq_{\mathbb{S}} f(\mathbf{z}_2)$ for all $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^n$. Checking orthant monotonicity is relatively easy if f is differentiable everywhere, as it corresponds to structural balance of the graph of the Jacobian matrix of f at any point $\mathbf{z} \in \mathbb{R}^n$. In particular, as stated in Proposition 1 in the SI, a differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is monotone w.r.t. $\mathbb{S}$ if and only if $S \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}) S \geq 0 \forall \mathbf{z} \in \mathbb{R}^n$, i.e., if and only if $\mathcal{G}(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}))$ is structurally balanced for some gauge matrix S , $\forall \mathbf{z} \in \mathbb{R}^n$. To understand this result, it is enough to notice that monotonicity with respect to the $\mathbb{S}$ orthant corresponds to nonnegativity of the directional derivative of f along the increasing $\mathbb{S}$ direction. The result then follows by a simple application of the chain rule. See SI for a formal proof.

Distance to balance/monotonicity: frustration

When a graph is not structurally balanced, then it is of interest to understand how distant it is from being structurally balanced. One of the measures adopted in the literature for this scope is the so-called *frustration* [14, 3]. This concept derives from Ising spin glass theory [6, 27, 26], and in fact it can be expressed in terms of the “ground state energy”,

¹Self-loops in $\mathcal{G}(A)$ are normally disregarded. Here this detail can be omitted, as there are never self-loops in our DAGs.

i.e., the minimum over the vector of “spins” $\mathbf{s}$ of a (normalized) energy-like functional

$$e(\mathbf{s}) = \frac{\sum_{i,j} [|A| - SAS]_{ij}}{2 \sum_{i,j} [|A|]_{ij}}, \quad (1)$$

where A is the weighted adjacency matrix of the graph, $|A|$ is its absolute value, and $\mathbf{s}$ is the signature vector of the aforementioned gauge diagonal matrix $S = \text{diag}(\mathbf{s})$. By construction it is $0 \leq e(\mathbf{s}) \leq 1$. If A is given, the frustration index is the minimum of this cost functional over all $\mathbf{s}$:

$$\epsilon = \min_{\substack{\mathbf{s}=[s_1 \dots s_n] \\ s_i = \pm 1}} e(\mathbf{s}). \quad (2)$$

For a connected $\mathcal{G}(A)$, $\epsilon = 0$ iff $\mathcal{G}(A)$ is structurally balanced. In fact, $\mathcal{G}(A)$ structurally balanced corresponds to the existence of a S such that $SAS \geq 0$. When instead $\mathcal{G}(A)$ is not structurally balanced, then $\epsilon > 0$ [14].

Heuristic algorithm for computing frustration

For structurally unbalanced graphs, computing ϵ is an NP-hard problem, equivalent to solving a MAX-CUT problem [5], or to a MAX-2XORSAT problem [26]. Consider the weighted adjacency matrix A and its symmetrized version $A_u = A + A^\top$. Let us rewrite (1) as

$$e(\mathbf{s}) = \frac{1}{2} \left(1 - \alpha \sum_{i,j} [SA_u S]_{ij} \right) = \frac{1}{2} (1 - \alpha \mathbf{1}^\top SA_u S \mathbf{1}), \quad (3)$$

where $\mathbf{1}$ is the vector of all 1s and $\alpha = 1 / \sum_{i,j} [|A_u|]_{ij}$. To search for the minimum of (3), we use the greedy heuristic described e.g. in [18], adapting it from adjacency matrices with binary entries ± 1 to our weighted A_u . The algorithm amounts essentially to flipping sign to the spins corresponding to rows of negative sum in $SA_u S$, until no negative row sum remains. See Algorithm 1 in Section B of the SI for a pseudocode.

Results

Monotonicity and structural balance for DNNs

In this section we show how to apply the concepts of monotonicity and structural balance to DNNs.

Given an input $\mathbf{x} \in \mathbb{R}^{n_0}$ and an output $\mathbf{y} \in \mathbb{R}^{n_h}$, let $\phi : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_h}$ be a function from input to output. To model ϕ , consider a deep neural network composed of h layers (meaning $h - 1$ hidden layers, or $h + 1$ total layers, including input and output). Each layer ℓ has n_ℓ nodes ($n_0 = \text{n. of inputs}$, and $n_h = \text{n. of outputs}$). Let $W^{(\ell)}$ and $\mathbf{b}^{(\ell)}$ be the weights associated to the ℓ -th layer (of size $n_\ell \times n_{\ell-1}$ and $n_\ell \times 1$ respectively). Denoting $\mathbf{z}_\ell$ the state of the nodes of the ℓ -th layer, with $\mathbf{z}_0 = \mathbf{x}$ the input and $\mathbf{z}_h = \mathbf{y}$ the output, then the input-output function ϕ describing the network can be written as

$$\begin{aligned} \mathbf{z}_\ell &= \sigma(W^{(\ell)} \mathbf{z}_{\ell-1} + \mathbf{b}^{(\ell)}), \quad \ell = 1, \dots, h-1 \\ \mathbf{y} &= W^{(h)} \mathbf{z}_{h-1} + \mathbf{b}^{(h)}, \end{aligned} \quad (4)$$

where $\sigma(\cdot) : \mathbb{R}^{n_\ell} \rightarrow \mathbb{R}^{n_\ell}$ is a componentwise activation function. We shall call $\mathbf{q}_\ell = W^{(\ell)}\mathbf{z}_{\ell-1} + \mathbf{b}^{(\ell)}$ the preactivation state at the ℓ -th layer.

While in (4) each layer is defined as contributing a matrix of weights $W^{(\ell)}$, a vector of biases $\mathbf{b}^{(\ell)}$ and an activation function $\sigma(\cdot)$, in the practice of the machine learning literature each operation that appears in a DNN is typically denoted as a “layer”. In particular, in the DNNs we consider in this study, elements such as dense connections, convolutions, residual connections, but also (max or average) pooling, ReLU, batch normalization, depth concatenation, additions, softmax, etc., are generically referred to as layers. Each of these layers can be mapped into the representation (4), even though this sometimes requires us to overload a bit the notation of (4). See the details in Section D in the SI.

Adjacency matrix of a DNN. Topologically, a DNN like (4) is a DAG, whose adjacency matrix can be constructed from the weights $W^{(\ell)}$, $\mathbf{b}^{(\ell)}$. For that, it is convenient to use a representation in homogeneous coordinates. Denote $\mathbf{z} = [\mathbf{x}^\top \ \mathbf{z}_1^\top \ \dots \ \mathbf{z}_{h-1}^\top \ \mathbf{y}^\top]^\top \in \mathbb{R}^n$ the stack vector of all node states (i.e., the state of all neurons, including inputs and outputs), of dimension $n = \sum_{\ell=0}^h n_\ell$, and $\bar{\mathbf{z}} = [\mathbf{z}^\top \ 1]^\top$ the vector of dimension $n+1$ containing also a single constant state corresponding to the bias terms in (4). For a DNN without multiple branches, where the output of layer ℓ acts as input only to layer $\ell+1$, its adjacency matrix is composed by a block-shift part (subdiagonal blocks in (5)), plus a final column containing the bias terms:

$$\bar{A} = \left[\begin{array}{ccccc|c} 0 & 0 & 0 & \dots & 0 & 0 \\ W^{(1)} & 0 & 0 & \dots & 0 & \mathbf{b}^{(1)} \\ 0 & W^{(2)} & 0 & \dots & 0 & \mathbf{b}^{(2)} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & W^{(h)} & 0 & \mathbf{b}^{(h)} \\ \hline 0 & 0 & \dots & 0 & 0 & 1 \end{array} \right] = \left[\begin{array}{c|c} A & \mathbf{b} \\ \hline 0 & 1 \end{array} \right] \in \mathbb{R}^{(n+1) \times (n+1)} \quad (5)$$

where $A \in \mathbb{R}^{n \times n}$ is the submatrix of $\bar{A}$ obtained dropping the last row and column and $\mathbf{b} = [0 \ (\mathbf{b}^{(1)})^\top \ \dots \ (\mathbf{b}^{(h)})^\top]^\top \in \mathbb{R}^n$ (see sketch in Fig. 1 for the case of a CNN). When the network has multiple branches, as is common when using residual connections, or when concatenating or adding the output of layers, then also blocks in the lower triangular part of A in (5) can be non-zero. For compactness of notation, we omit here these types of connections. All results can be extended to include them at the cost of some extra bookkeeping, see Section D in the SI.

A compact description of the DNN based on the hidden states. On a similar note, to streamline the analysis we can also express the various nonlinearities of the DNN (such as ReLU, max-pooling, batch normalization) as special cases of the activation function $\sigma(\cdot)$ (with a slight abuse of notation, see details in Section D of the SI). Since σ acts componentwise, using A and $\mathbf{b}$ the DNN (4) can be vectorized as follows

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_{h-1} \\ \mathbf{z}_h \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \sigma(W^{(1)}\mathbf{z}_0 + \mathbf{b}^{(1)}) \\ \vdots \\ \sigma(W^{(h-1)}\mathbf{z}_{h-2} + \mathbf{b}^{(h-1)}) \\ W^{(h)}\mathbf{z}_{h-1} + \mathbf{b}^{(h)} \end{bmatrix} = f(A\mathbf{z} + \mathbf{b}) \quad (6)$$

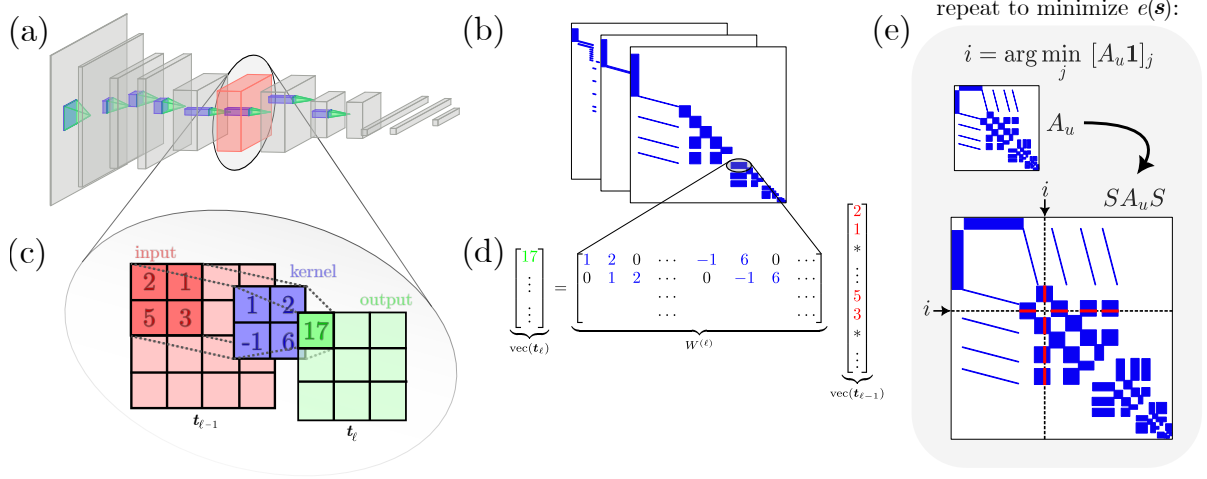


Figure 1: Constructing the adjacency matrix of a CNN. Each layer of the CNN (a) is represented by as sub-diagonal block in the adjacency matrix of the network (b). Convolutional operations (c) are represented as linear transformations (d), which give the weights of the edges in the multibanded Toeplitz adjacency matrix. (e): Heuristic minimization procedure used to compute frustration: choose the row of the adjacency matrix with most negative sum and apply a gauge transformation to it (flipping the sign of all entries in the row and column), until no negative row sum remains.

where $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ indicates a componentwise map gathering all (vectorized) activation functions σ (plus the trivial map on the first n_0 components). If the parameters A and $\mathbf{b}$ are given, then f is a map from $\mathbf{z}$ to itself. For later reference, denote $\boldsymbol{\theta}$ the stack vector of all (vectorized) $W^{(\ell)}$, and $\bar{\boldsymbol{\theta}}$ the vector containing $\boldsymbol{\theta}$ and $\mathbf{b}$.

Structural balance and monotonicity for DNNs. An easy way to investigate monotonicity of the DNN (6) is to consider its Jacobian matrix. Observe first that activation functions $\sigma(\mathbf{q})$ such as ReLU, pooling and normalization are all characterized by $\frac{\partial \sigma(\mathbf{q})}{\partial \mathbf{q}} \geq 0$, even when the derivative should be replaced by a generalized derivative and the inequality by an inclusion because of the non-differentiability of σ .

When we restrict to these activation functions, checking monotonicity of the DNN map f is rather straightforward, as it amounts to checking structural balance of the $n \times n$ subgraph $\mathcal{G}(A)$ of $\mathcal{G}(\bar{A})$. As shown in Theorem 1 in the SI, the DNN map f is monotone if and only if $\mathcal{G}(A)$ is structurally balanced. In fact, nonnegativity and “separability” (i.e., diagonal structure) of the partial derivatives of σ imply that the sign pattern of the Jacobian of f reflects that of A . More precisely, if we denote $F(\mathbf{z}) = \frac{\partial f}{\partial \mathbf{z}}(A\mathbf{z} + \mathbf{b})$ the Jacobian of f at $\mathbf{z}$, it is

$$\text{sgn}(F(\mathbf{z})) = \text{sgn}(\text{diag}(\mathbb{I}(\mathbf{z}))A), \quad (7)$$

where $\text{sgn}(\cdot)$ is the sign function, $\mathbb{I}(\mathbf{z})$ is a vectorized indicator function (here equal to 1 at a node if the associated z_i is active, equal to 0 otherwise), and $\text{diag}(\cdot)$ is the diagonal matrix, see Lemma 1 in the SI. In words, the adjacency matrix A determines the signature of the Jacobian of the DNN, modulo some vanishing rows in correspondence of hidden

nodes that are inactive. The value of $\mathbb{I}(\mathbf{z})$ depends on the input $\mathbf{x}$ and on the type of activation function $\sigma(\cdot)$ present at a node, see Section D.3 in the SI for more details.

Active subnetwork for an input. Notice that when a node is inactive then also its outgoing edges become irrelevant for the DNN map. Hence, for a specific input $\mathbf{x}$, in place of (7) one can consider the more compact active sub-adjacency matrix

$$A_{\text{act}}(\mathbf{x}) = \text{diag}(\mathbb{I}(\mathbf{z})) A \text{diag}(\mathbb{I}(\mathbf{z})).$$

Similarly to (2), we can compute the frustration index of A_{act} as

$$\epsilon_{\text{act}}(\mathbf{x}) = \min_{\substack{\mathbf{s}=[s_1 \dots s_n] \\ s_i=\pm 1}} \frac{\sum_{i,j} [|A_{\text{act}}(\mathbf{x})| - S A_{\text{act}}(\mathbf{x}) S]_{ij}}{2 \sum_{i,j} [|A_{\text{act}}(\mathbf{x})|]_{ij}}. \quad (8)$$

Relation with IO monotonicity. The concept of monotonicity considered in this paper differs from the one normally used in the DNN literature [7, 24, 31, 33, 34], which in our setting could be called Input-Output (IO) monotonicity. A DNN, represented as an IO map $\mathbf{y} = \phi(\mathbf{x}, \bar{\boldsymbol{\theta}})$, is IO monotone with respect to the input order $\mathbb{S}_{\mathbf{x}}$ and the output order $\mathbb{S}_{\mathbf{y}}$ if $\forall \mathbf{x}_1, \mathbf{x}_2$

$$\mathbf{x}_1 \leq_{\mathbb{S}_{\mathbf{x}}} \mathbf{x}_2 \Rightarrow \phi(\mathbf{x}_1, \bar{\boldsymbol{\theta}}) \leq_{\mathbb{S}_{\mathbf{y}}} \phi(\mathbf{x}_2, \bar{\boldsymbol{\theta}}). \quad (9)$$

Notice that since in this case the input and output spaces are different, two distinct partial orders $\mathbb{S}_{\mathbf{x}}$ and $\mathbb{S}_{\mathbf{y}}$ must be specified in the definition of IO monotone. Since $\mathbf{x}$ and $\mathbf{y}$ are parts of the “complete state” $\mathbf{z}$, both signature vectors $\mathbf{s}_{\mathbf{x}}$ and $\mathbf{s}_{\mathbf{y}}$ associated to $\mathbb{S}_{\mathbf{x}}$ and $\mathbb{S}_{\mathbf{y}}$ are subvectors of the signature $\mathbf{s}$ associated to $\mathbb{S}$.

A key result that we rely upon in this paper is that structural balance of the DAG is a sufficient condition for IO monotonicity. In fact, from (7), the sign pattern of the Jacobian of ϕ is inherited from that of A , hence if the graph $\mathcal{G}(A)$ is structurally balanced then the IO map $\mathbf{y} = \phi(\mathbf{x}, \bar{\boldsymbol{\theta}})$ must be IO monotone, see Theorem 2 in the SI for a detailed proof. Consequently, if the DNN map f is monotone then the IO map $\mathbf{y} = \phi(\mathbf{x}, \bar{\boldsymbol{\theta}})$ is IO monotone (Corollary 1 in the SI).

The crucial advantage of dealing with “hidden state monotonicity”, i.e., with the map f instead of the IO map ϕ , is that we obtain a notion which gives insight into the organization of the DNN induced by the training algorithm, which the simpler IO perspective cannot account for.

Quantifying near-monotonicity on DNNs

If a DNN is not monotone, then (9) cannot be obeyed rigorously for all $\mathbf{x}_1, \mathbf{x}_2$, and we also know from Theorem 1 in the SI that the graph $\mathcal{G}(A)$ associated to the DNN cannot be structurally balanced, i.e., $\epsilon > 0$. As mentioned in the Methods, the frustration ϵ , a measure of distance to structural balance, is often taken as a proxy for the distance to monotonicity of a system whose Jacobian has graph $\mathcal{G}(A)$. However, ϵ provides only an indirect measure. In this section we aim to compute a more direct measure of near-monotonicity. The rationale is that when the frustration ϵ is low, one would expect that to some extent the partial order determined by $\mathbb{S}_{\mathbf{x}}$ and $\mathbb{S}_{\mathbf{y}}$ in the IO map $\phi(\cdot)$ is still respected. One possible way to quantify this is to count how many output nodes

$y_i = \phi_i(\mathbf{x}, \bar{\theta})$, $i = 1, \dots, n_h$, still respect the output order $\mathbb{S}_y$ when $\mathbf{x}_1$ and $\mathbf{x}_2$ respect the input order, i.e., for $\mathbf{x}_1 \leq_{\mathbb{S}_x} \mathbf{x}_2$. Before embarking in this quantification, in next section we explain why for a non-monotone function, even a near-monotone one, it is a priori impossible to disambiguate increasing vs decreasing near-monotonicity w.r.t. the partial order, just by looking at the partial order pair $\{\mathbb{S}_x, \mathbb{S}_y\}$ and at ϵ .

Decreasing/increasing trends in near-monotonicity: an heuristic explanation of an intrinsic ambiguity. Lack of exact monotonicity introduces an ambiguity in the near-monotonicity pattern, which can be understood by looking at IO paths, i.e., at directed paths on $\mathcal{G}(A)$ connecting an input i to an output j . Once we drop the edge direction, any two equal-ends IO paths, say p_{ij}^1 and p_{ij}^2 , form an undirected cycle, which is necessarily positive when the DNN is monotone, even though it could be $p_{ij}^k > 0$ or $p_{ij}^k < 0$, $k = 1, 2$ (all equal-ends IO paths between i and j have to have the same sign in the monotone case, otherwise negative undirected cycles would appear). However, when the DNN is not monotone, then some negative undirected cycles exist, implying that equal-ends IO paths with opposite signs are present. IO paths of opposite signs have opposite effects on the output order: if a positive IO path reflects an expression like (9) (“increasing output” in the partial order $\mathbb{S}_y$), a negative IO path reflects instead its opposite $\phi(\mathbf{x}_1, \bar{\theta}) \geq_{\mathbb{S}_y} \phi(\mathbf{x}_2, \bar{\theta})$ (“decreasing output”). For given $\mathbf{x}_1$ and $\mathbf{x}_2$ obeying $\mathbf{x}_1 \leq_{\mathbb{S}_x} \mathbf{x}_2$, the two cases coexist when a network is not monotone. Frustration alone cannot predict the prevalence of positive or negative IO paths, meaning that predicting whether the near-monotonicity is of increasing or decreasing type cannot be done a priori, and for a given DNN it can vary with $\mathbf{x}_1$ and $\mathbf{x}_2$.

Quantifying near-monotonicity in DNNs: a direct test. Let Ω be the output alignment fraction for increasing monotonicity: $\Omega := \frac{1}{n_h} \mathbb{I}(\phi(\mathbf{x}_1, \bar{\theta}) \leq_{\mathbb{S}_y} \phi(\mathbf{x}_2, \bar{\theta}))$, where n_h is the output size, i.e., Ω is the fraction of outputs for which $\phi(\mathbf{x}_1, \bar{\theta}) \leq_{\mathbb{S}_y} \phi(\mathbf{x}_2, \bar{\theta})$ holds. Then we say that ϕ is λ -monotone with respect to the IO partial order pair $\{\mathbb{S}_x, \mathbb{S}_y\}$ if

$$\mathbf{x}_1 \leq_{\mathbb{S}_x} \mathbf{x}_2 \Rightarrow P(|\Omega - 0.5| \geq \lambda) \geq 2\lambda. \quad (10)$$

The two (mutually exclusive) inequalities, $\Omega > 0.5 + \lambda$ and $\Omega < 0.5 - \lambda$, specify the size of Ω in the two possible cases resulting from the increasing/decreasing ambiguity just described. Eq. (10) expresses the probability at which these two inequalities are expected to occur when an image $\mathbf{x}_1$ is perturbed in an increasing direction with respect to $\mathbb{S}_x$. The value $\lambda \in [0, 0.5]$ can be taken as a measure of (increasing or decreasing) IO monotonicity associated to an input $\mathbf{x}_1$: the larger λ is, the closer to IO monotone the DNN is. $\lambda = 0$ means no monotonicity at all, while $\lambda = 0.5$ means exact IO monotonicity, and corresponds to the “lower” (decreasing) condition in (10) becoming void (no increasing/decreasing ambiguity is present in the exact monotone case, which is always increasing as in (9)).

Numerical experiments

To quantify the frustration and the degree of monotonicity in state-of-the-art DNNs, we consider seven pre-trained networks available in MATLAB’s Deep Learning Toolbox (ShuffleNet [45], SqueezeNet [19], ResNet18 [16], AlexNet [22], and GoogLeNet [38]) and

from PyTorch Hub (AlexNet and GoogLeNet), which are all convolutional-type neural networks trained on the ImageNet dataset [11].

Frustration. We construct the adjacency matrix of the CNNs as described above, restricting to the homogeneous matrix A and disregarding the bias terms $\mathbf{b}$, see details in Section D and Fig. 6 of the SI. We then run Algorithm 1 (see Section B of the SI) to estimate the frustration index (2) in correspondence of the real weights of the networks. To explore the energy landscape associated to these weights, we repeat the calculation of frustration 80 times, initializing the sign pattern matrix S differently each time, in particular applying 1 million random single-neuron gauge transformations at the start of the algorithm (we explore in this way 80 different replicas of the same “quenched” spin glass). The least among these 80 values of frustration is our best estimate of the frustration of a CNN. To assess if this frustration value is below or above our expectations, we compare it with the frustration obtained from several null models, in which the network weights have been reshuffled or randomized. The first null model (N1) is constructed by randomly reordering the position of the parameters in each convolutional layer and in each dense layer. The reshuffling occurs within each layer and, for the convolutional layers, all repeated instances of a parameter in the Toeplitz matrix $W^{(\ell)}$ are replaced with another parameter but they remain identical. Accordingly, the recomputed adjacency matrix has the same topology and convolutional structure as the original one. The second null model (N2) is obtained by shuffling the weights of the adjacency network across all convolutional and dense layers, so that the new adjacency matrix has entries in the same positions as the real network and N1 model but it no longer represents convolutions. A third null model (N3) is constructed by drawing new weights for the convolutional layers and dense layers from either uniform (Xavier) or normal (He) distributions to simulate the state of a CNN before training starts. The pooling layer weights are not modified in any null model. We create 80 null models of each kind and explore their energy landscape by applying our algorithm to two instances of each null model, corresponding to different initial S (the second instance is obtained randomizing the initial S through 1 million random single-neuron gauge transformations).

The results of these computations are shown in Fig. 2. Denote ϵ_R the frustration for the network with the real parameters, and ϵ_{N1} , ϵ_{N2} and ϵ_{N3} the frustration of the three null models. We always find $\epsilon_R < \epsilon_{N1} < \epsilon_{N2}$ and the difference, though small, is statistically significant: Welch’s t-test shows significant difference in distribution means between ϵ_R , ϵ_{N1} and ϵ_{N2} at arbitrarily low p-value ($< 10^{-50}$). Recall that $0 \leq \epsilon \leq 1$. A completely random network, with random weights, would corresponds to $\epsilon \approx 0.5$. This is roughly what happens to ϵ_{N3} in the reinitialized model N3, where the two distinct peaks appearing in some of the networks correspond to the two different initializations mentioned above, with He initialization tending to give slightly lower frustration. For most CNNs, some decrease in frustration is already visible in ϵ_{N2} where the reshuffling of the real weights destroys the convolutional structure, but in A many identical weights are repeated, which already favors a reduction in disorder. When the convolution structure is preserved, as in the N1 null model, this effect is enhanced, and in fact ϵ_{N1} is significantly smaller than ϵ_{N2} and ϵ_{N3} . The optimized real weights reduce even further the frustration: in all cases ϵ_R is significantly smaller than the frustration of the null models.

In addition, Fig. 2 shows that all 80 values of ϵ_R are always rather close to each other:

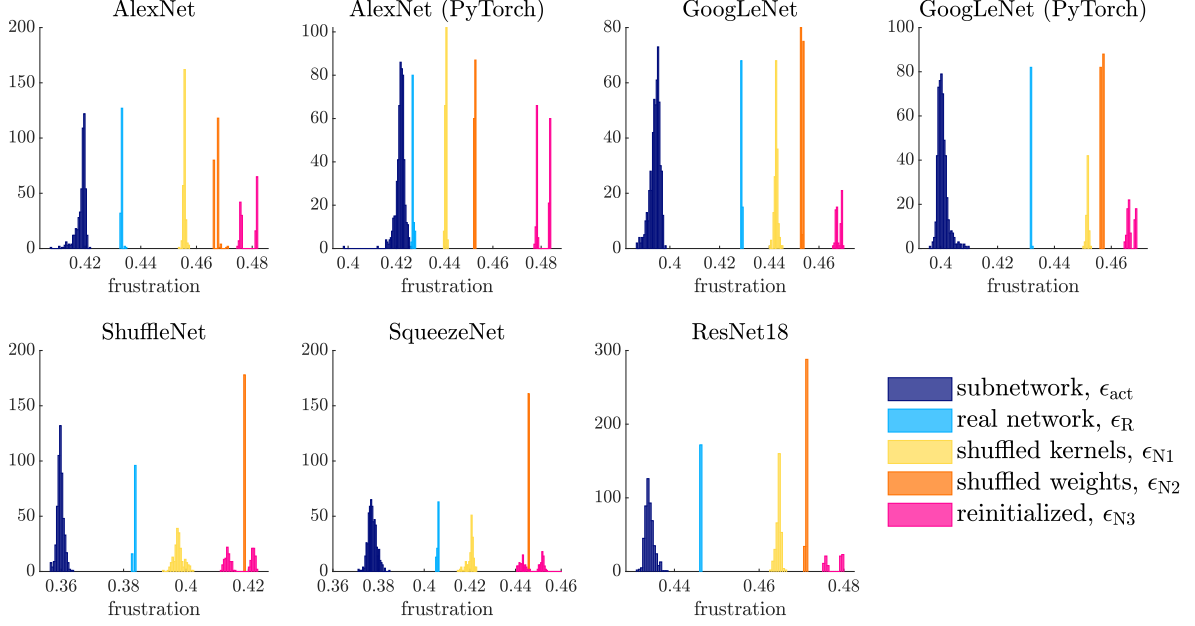


Figure 2: Frustration of the seven pretrained CNNs (ϵ_R), their active subnetworks (ϵ_{act}) and their null models (ϵ_{N1} , ϵ_{N2} and ϵ_{N3}).

the histograms of ϵ_R have always a limited dispersion. This means that the local minima reached by the algorithm are all close to each other in value, i.e., that the energy landscape of the spin glass is not particularly rugged around its ground states.

As the results are similar for all the CNNs we consider, even for those implemented and trained in a different platform, it appears that they hold regardless of specific details such as the values of the hyperparameters chosen for the training and other implementation technicalities.

Also for the active subnetwork $A_{act}(\mathbf{x})$ associated to each image $\mathbf{x}$, it is possible to compute the frustration ϵ_{act} using (8). For this purpose, we use images from the Imagenette subset of the ImageNet dataset (see Section E in the SI), which contains 10 classes of images. For around 500 images from these 10 classes the frustration ϵ_{act} is shown in Fig. 2. In all networks it is nearly always $\epsilon_{act} < \epsilon_R$, i.e., the adjacency matrix A is organized in such a manner that each active subnetwork $A_{act}(\mathbf{x})$ is even less frustrated than the entire A .

Near-monotonicity. To check whether the reduced frustration results in a CNN which is near-monotone, i.e. in an IO function $\mathbf{y} = \phi(\mathbf{x}, \bar{\boldsymbol{\theta}})$ which is closer to monotone than expected, given an image $\mathbf{x}_1$, we perturb it along the direction indicated by the input gauge transformation $S_{\mathbf{x}}$, i.e., we add a randomly chosen perturbation $\boldsymbol{\delta}_{S_{\mathbf{x}}} = S_{\mathbf{x}}\boldsymbol{\delta} \in \mathbb{S}_{\mathbf{x}}$, with $\boldsymbol{\delta} \in \mathbb{R}^{n_0}$, $\boldsymbol{\delta} > 0$, so that $\mathbf{x}_2 = \mathbf{x}_1 + \boldsymbol{\delta}_{S_{\mathbf{x}}}$ is s.t. $\mathbf{x}_2 \geq_{S_{\mathbf{x}}} \mathbf{x}_1$. Each element of $\boldsymbol{\delta}$ is drawn from a uniform distribution. We then compute the associated logit (pre-softmax) outputs $\mathbf{y}_k = \phi(\mathbf{x}_k, \bar{\boldsymbol{\theta}})$, $k = 1, 2$, and calculate the output alignment fraction Ω . For each image we repeat this calculation 100 times, changing the magnitude $\|\boldsymbol{\delta}\|$ of the random perturbation. The procedure is repeated for 1000 images on each of the seven CNNs.

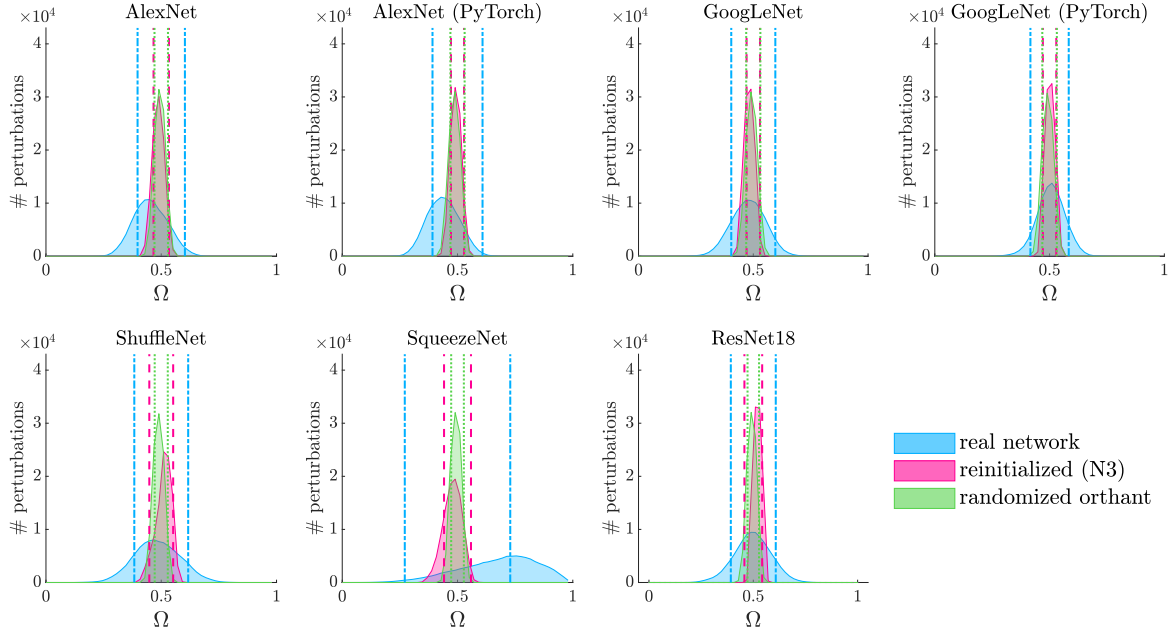


Figure 3: Histograms of the output alignment fraction Ω and associated values of λ , for the real networks (azure) and for two null models (violet and green). For each of the seven CNNs, the histogram of the real network differ significantly from the other two: it is much broader on both sides of 0.5, showing that $|\Omega - 0.5|$ is much larger than expected from null models. For each histogram, the two vertical bars identify the interval of width 2λ described in Eq. (10). See Fig. 7 in the SI for details on how to compute λ .

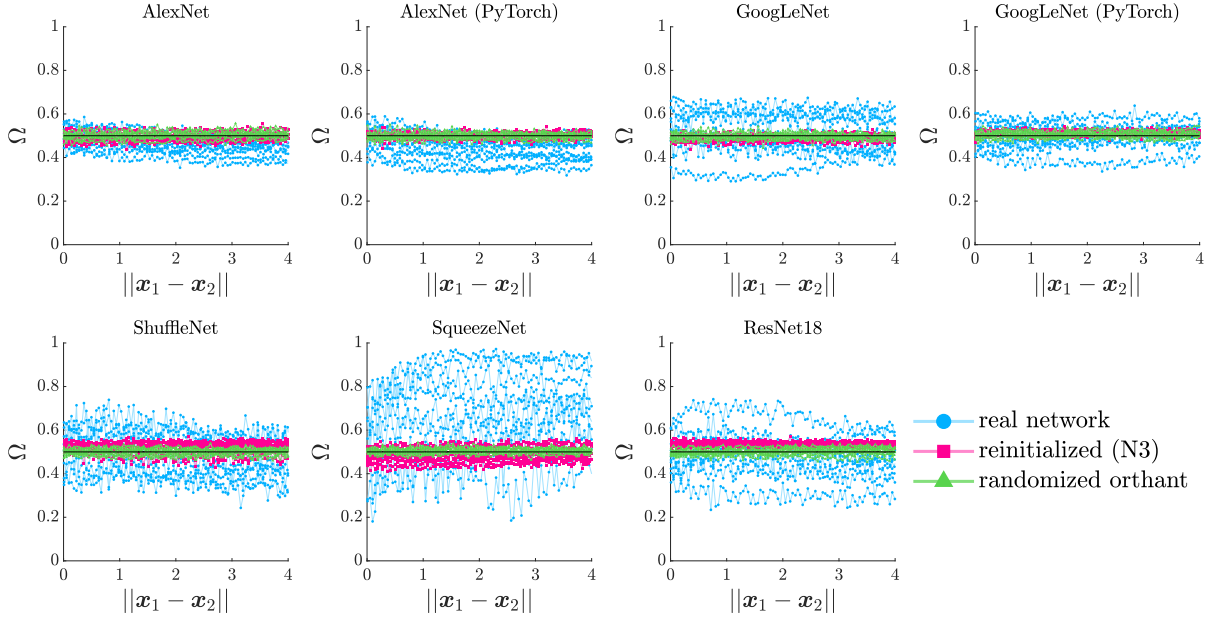


Figure 4: Output alignment fraction Ω in response to 100 perturbations $\delta_{\mathbf{s}_x}$ of different amplitude for a few images $\mathbf{x}_1$, as a function of the amplitude $\|\delta_{\mathbf{s}_x}\|$, for each of the seven CNNs. For the real network (azure) the alignment is normally larger than for the two null models (violet and green), and all perturbations for an image (linked by a continuous line) tend to be on the same side of 0.5, i.e., always increasing or always decreasing.

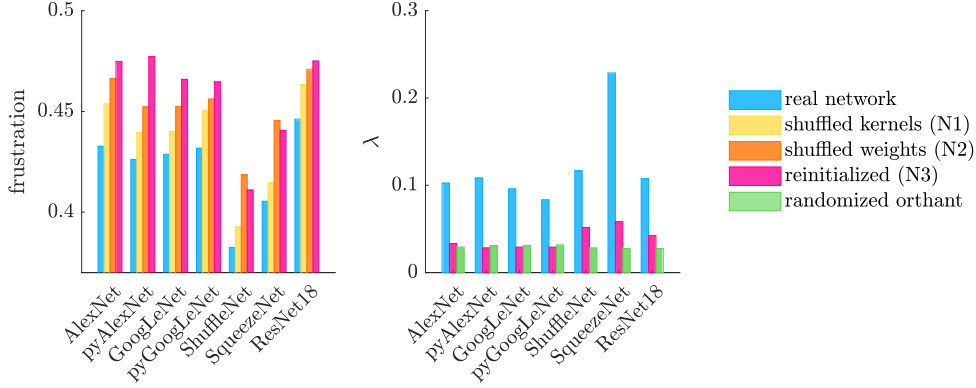


Figure 5: Summary of the results. (a) Frustration computed for all real networks and null models. (b) λ -monotonicity values for all networks with real weights, re-initialized weights (N3) and real weights but with input perturbation in a random direction.

The resulting histograms for Ω are shown in Fig. 3. We use them to compute λ that satisfies (10), see Fig. 7 in the SI. For all CNNs, λ is in the range $0.1 \div 0.25$, and always significantly larger than when computed on null models. In this case we consider as null models N3² and a randomly chosen partial order in place of the $\{\mathbb{S}_x, \mathbb{S}_y\}$ determined by Algorithm 1. For all seven CNNs, the histograms of output alignment Ω associated to these two null models are much more concentrated around 0.5, see Fig. 3, and the corresponding λ almost never exceeds 0.05, see summary in Fig. 5. As can be seen in Fig. 3, for the majority of CNNs (except SqueezeNet and the two AlexNet), both the cases $\Omega > 0.5 + \lambda$ (increasing monotonicity) and $\Omega < 0.5 - \lambda$ (decreasing monotonicity) are present. What is remarkable is that for a given image all perturbations we apply tend to induce always increasing monotonicity or always decreasing monotonicity even when the amplitude of δ varies, see Fig. 4 and Fig. 8 in the SI. In addition, when for instance $\|\delta\| = 4$, the perturbed image is predicted to belong to the same class as the original image in about 90% of the cases. Both observations confirm that the partial order direction $\{\mathbb{S}_x, \mathbb{S}_y\}$ determined by our computation of frustration is indeed special for the CNN, and hence that the CNNs have a near-monotone behavior along them.

Discussion

For all pretrained CNNs we consider in this study, the frustration ϵ_R is significantly less than expected from any kind of null model associated to the network. This can be interpreted by saying that a trained DNN has some degree of order and it is not a completely disordered system. Having some degree of order means that in a trained CNN the function f is closer to being a monotone function than a random function.

This does not mean that the CNN is monotone. In fact a value of $\epsilon_R \approx 0.4$ is rather far from structural balance, and hence the CNN is rather far from behaving like a “trivial” (i.e., completely predictable) function as an exactly monotone function would be. Yet, a trained

²N1 and N2 sometimes lead to vanishing – or exploding – hidden states, hence they are not suitable to compute outputs.

network has some amount of predictability which a non-trained network does not have, and the partial order we compute indeed provides a direction in which this predictability becomes manifest. It is tempting to speculate that this reduction in frustration (or increase in monotonicity) reflects the inference power of the CNN.

An interesting observation is that also the null model N1 is less frustrated than a completely random model: $\epsilon_{N1} < \epsilon_{N2}, \epsilon_{N3}$. This suggests that part of the order in the CNN is due to its organization into convolutional structures, and in fact, CNNs are a *de facto* standard in modern DNNs. If we look at what happens in the inner organization of A , the Toeplitz structure associated to a convolutional layer $W^{(\ell)}$ is such that many rows and columns of $W^{(\ell)}$ have identical row/column sum. It is not yet clear to us why this should favor order and be associated to lower frustration, especially in view of the fact that the reinitialized null model N3 also consists of convolutional layers and hence shares all features due to the Toeplitz structure.

What is clear is that for the set of pretrained parameters θ the convolutional structure leads to a less rugged landscape in the energy functional $e(\mathbf{s})$. In fact, when applying Algorithm 1, the rows/columns of A with identical sum all flip sign simultaneously, simplifying the search for a ground state in the energy $e(\mathbf{s})$. While this per se does not provide any information on the training dynamics, it may help explaining why even simple training algorithms are typically successful.

The computation of frustration is instrumental for the calculation of IO monotonicity, since it provides as a byproduct the partial order pair $\{\mathbb{S}_x, \mathbb{S}_y\}$. The presence of a preferential partial order direction sheds light on the internal organization of the CNNs: images perturbed along the input partial order direction $\mathbb{S}_x$ are consistently still classified in the same class as the original image. This suggests that the partial order direction determined by the pair $\{\mathbb{S}_x, \mathbb{S}_y\}$ is robust to perturbations, hence that the CNN is unlikely to be affected by practices like adversarial training along it. Notice that the perturbation δ can have different amplitude on different pixels, provided that it $\delta_i > 0$ for all i (or $\delta_i < 0$ for all i), so that $\delta_{\mathbb{S}_x}$ is increasing (or decreasing) in the desired input partial order. As soon as the direction of the perturbation departs from $\mathbb{S}_x$, then any sign of monotonicity quickly disappears.

For DNNs, the search for implicit regularization principles, i.e., for mathematical rules underpinning the inherent bias of DNNs towards “low-complexity” solutions that generalize well is characterized by only limited successes, in spite of a intense research [20, 28, 30, 37, 41].

One strong form of organization that has been recently detected, the so-called neural collapse, occurs in the terminal phase of training, i.e., in the extra training phase that follows the achievement of the zero training error [29]. Relating this form of order with our near-monotonicity is at the moment difficult, as our networks are not trained in the terminal phase, and our output (pre softmax, but post dense layer) does not coincide with the (pre dense layer) last layer studied in [29]. We leave investigating the monotonicity of the terminal phase of training to the future research.

To summarize, what we are suggesting in this paper is that near-monotonicity could be one mathematical notion characterizing implicit regularization in DNNs, especially since it is grounded on well-established ideas like the notion of disorder of Ising spin glasses. Furthermore, since frustration is representable by the valleys of an energy functional which is different from the loss function used in the training, a question that can be posed is

whether frustration can be taken as a measure of complexity, to be used to understand generalizability [28, 44]. Also in this case, a further analysis is needed before drawing any conclusion.

References

- [1] E. M. Achour, F. Malgouyres, and S. Gerchinovitz. The loss landscape of deep linear neural networks: a second-order analysis. *arXiv preprint arXiv:2107.13289*, 2021.
- [2] D. J. Amit, H. Gutfreund, and H. Sompolinsky. Spin-glass models of neural networks. *Physical Review A*, 32(2):1007, 1985.
- [3] S. Aref and M. C. Wilson. Measuring partial balance in signed networks. *Journal of Complex Networks*, 6(4):566–595, 2018.
- [4] S. Arora, N. Cohen, and E. Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *International conference on machine learning*, pages 244–253. PMLR, 2018.
- [5] F. Barahona. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.
- [6] K. Binder and A. P. Young. Spin glasses: Experimental facts, theoretical concepts, and open questions. *Reviews of Modern physics*, 58(4):801, 1986.
- [7] J.-R. Cano, P. A. Gutiérrez, B. Krawczyk, M. Woźniak, and S. García. Monotonic classification: An overview on algorithms, performance measures and data sets. *Neurocomputing*, 341:168–182, 2019.
- [8] Y. Chitour, Z. Liao, and R. Couillet. A geometric approach of gradient descent algorithms in linear neural networks. *arXiv preprint arXiv:1811.03568*, 2018.
- [9] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.
- [10] N. Cohen and N. Razin. Lecture notes on linear neural networks: A tale of optimization and generalization in deep learning. *arXiv preprint arXiv:2408.13767*, 2024.
- [11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [12] V. Dotsenko. *An introduction to the theory of spin glasses and neural networks*, volume 54. World Scientific, 1995.
- [13] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets. Reasoning About a Highly Connected World*. Cambridge Univ. Press, Cambridge, 2010.

- [14] G. Facchetti, G. Iacono, and C. Altafini. Computing global structural balance in large-scale signed social networks. *Proceedings of the National Academy of Sciences*, 108(52):20953–20958, 2011.
- [15] F. Harary. A matrix criterion for structural balance. *Naval Research Logistics Quarterly*, 7(2):195–199, 1960.
- [16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [17] G. Iacono and C. Altafini. Monotonicity, frustration, and ordered response: an analysis of the energy landscape of perturbed large-scale biological networks. *BMC systems biology*, 4:1–14, 2010.
- [18] G. Iacono, F. Ramezani, N. Soranzo, and C. Altafini. Determining the distance to monotonicity of a biological network: a graph-theoretical approach. *IET Systems Biology*, 4(3):223–235, 2010.
- [19] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.
- [20] Z. Ji and M. Telgarsky. The implicit bias of gradient descent on nonseparable data. In *Conference on learning theory*, pages 1772–1798. PMLR, 2019.
- [21] K. Kawaguchi. Deep learning without poor local minima. *Advances in neural information processing systems*, 29, 2016.
- [22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
- [23] Z. Li, Y. Luo, and K. Lyu. Towards resolving the implicit bias of gradient descent for matrix factorization: Greedy low-rank learning. *arXiv preprint arXiv:2012.09839*, 2020.
- [24] X. Liu, X. Han, N. Zhang, and Q. Liu. Certified monotonic neural networks. *Advances in Neural Information Processing Systems*, 33:15427–15438, 2020.
- [25] A. Ma’ayan, A. Lipshtat, R. Iyengar, and E. D. Sontag. Proximity of intracellular regulatory networks to monotone systems. *IET Systems Biology*, 2(3):103–112, 2008.
- [26] M. Mezard and A. Montanari. *Information, physics, and computation*. Oxford University Press, 2009.
- [27] M. Mézard, G. Parisi, and M. A. Virasoro. *Spin glass theory and beyond: An Introduction to the Replica Method and Its Applications*, volume 9. World Scientific Publishing Company, 1987.
- [28] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. *Advances in neural information processing systems*, 30, 2017.

- [29] V. Pappayan, X. Han, and D. L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.
- [30] H. Ravi, C. Scott, D. Soudry, and Y. Wang. The implicit bias of gradient descent on separable multiclass data. *Advances in Neural Information Processing Systems*, 37:81324–81359, 2024.
- [31] D. Runje and S. M. Shankaranarayana. Constrained monotonic neural networks. In *International Conference on Machine Learning*, pages 29338–29353. PMLR, 2023.
- [32] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.
- [33] A. Sharma and H. Wehrheim. Testing monotonicity of machine learning models. *arXiv preprint arXiv:2002.12278*, 2020.
- [34] J. Sill. Monotonic networks. *Advances in neural information processing systems*, 10, 1997.
- [35] H. L. Smith. *Monotone dynamical systems: an introduction to the theory of competitive and cooperative systems: an introduction to the theory of competitive and cooperative systems*. Number 41. American Mathematical Soc., 1995.
- [36] E. D. Sontag. Monotone and near-monotone biochemical networks. *Systems and synthetic biology*, 1(2):59–87, 2007.
- [37] D. Soudry, E. Hoffer, M. S. Nacson, S. Gunasekar, and N. Srebro. The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, 19(70):1–57, 2018.
- [38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [39] G. Toulouse et al. Theory of the frustration effect in spin glasses: I. *Spin Glass Theory and Beyond: An Introduction to the Replica Method and Its Applications*, 9:99, 1987.
- [40] S. Tripathi, D. A. Kessler, and H. Levine. Minimal frustration underlies the usefulness of incomplete regulatory network models in biology. *Proceedings of the National Academy of Sciences*, 120(1):e2216109120, 2023.
- [41] G. Vardi. On the implicit bias in deep-learning algorithms. *Communications of the ACM*, 66(6):86–93, 2023.
- [42] S. Wasserman and K. Faust. *Social Network Analysis: methods and applications*. Cambridge Univ. Press, 1994.

- [43] C. Yun, S. Sra, and A. Jadbabaie. Global optimality conditions for deep neural networks. *arXiv preprint arXiv:1707.02444*, 2017.
- [44] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.
- [45] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

Supplementary Information

A Mathematical results

In this section we provide mathematical formulations and formal proofs of the properties mentioned and exploited in the main text of the paper. In the section “Monotonicity” of the Methods we use the following.

Proposition 1 *A differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is monotone w.r.t. $\mathbb{S}$ iff $S \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}) S \geq 0 \ \forall \mathbf{z} \in \mathbb{R}^n$, i.e., iff $\mathcal{G}(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}))$ is structurally balanced with gauge matrix S , $\forall \mathbf{z} \in \mathbb{R}^n$.*

Proof of Proposition 1. If f is $\mathbb{S}$ -monotone, the partial order $\leq_{\mathbb{S}}$ can be simplified by a gauge transformation. Namely, there exists S such that $S\mathbf{z}_1 \leq S\mathbf{z}_2 \implies f(\mathbf{z}_1) \leq_{\mathbb{S}} f(\mathbf{z}_2)$, or, equivalently, $Sf(\mathbf{z}_1) \leq Sf(\mathbf{z}_2)$. Consider the change of basis $\mathbf{r} = S\mathbf{z}$, which is simply a flip of the order in some orthants. The partial order in the new basis is the usual $\mathbb{R}^n$ order: $\mathbf{r}_1 \leq \mathbf{r}_2 \implies f(\mathbf{r}_1) \leq f(\mathbf{r}_2) \ \forall \mathbf{r}_1, \mathbf{r}_2 \in \mathbb{R}^n$. In the new basis, f is monotone in this usual sense. When we differentiate it along the direction $\mathbf{r}_2 - \mathbf{r}_1 \geq 0$, we get, using the chain rule,

$$\frac{d}{dt}f(\mathbf{r}_1 + t(\mathbf{r}_2 - \mathbf{r}_1)) = \frac{\partial f}{\partial \mathbf{r}}(\mathbf{r}_1 + t(\mathbf{r}_2 - \mathbf{r}_1))(\mathbf{r}_2 - \mathbf{r}_1) \geq 0 \quad \forall t \in [0, 1] \quad (11)$$

By contradiction, assume that $[\frac{\partial f}{\partial \mathbf{r}}(\mathbf{r})]_{ij} < 0$ for some index i, j and for some $\mathbf{r} \in \mathbb{R}^n$. Then there exists $\mathbf{r}_1, \mathbf{r}_2$ and $t \in [0, 1]$, with $\mathbf{r}_2 - \mathbf{r}_1 \geq 0$, s.t. $\mathbf{r} = \mathbf{r}_1 + t(\mathbf{r}_2 - \mathbf{r}_1)$ and (11) is violated. One example is $\mathbf{r}_2 - \mathbf{r}_1 = \mathbf{e}_j$, the elementary vector having 1 on the j -th slot. In fact, $[\frac{d}{dt}f(\mathbf{r}_1 + t\mathbf{e}_j)]_i = [\frac{\partial f}{\partial \mathbf{r}}(\mathbf{r}_1 + t\mathbf{e}_j)\mathbf{e}_j]_i < 0$, which contradicts the monotonicity assumption. Hence it must be $\frac{\partial f}{\partial \mathbf{r}}(\mathbf{r}) \geq 0 \ \forall \mathbf{r} \in \mathbb{R}^n$. Since $S^{-1} = S$, in the original basis this corresponds to $S \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}) S \geq 0$, which is an equivalent condition to $\mathcal{G}(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}))$ being structurally balanced.

Viceversa, if $\mathcal{G}(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}))$ is structurally balanced, then there exists a gauge transformation S such that, in the new basis $\mathbf{r} = S\mathbf{z}$, $\frac{\partial f}{\partial \mathbf{r}}(\mathbf{r}) \geq 0 \ \forall \mathbf{r} \in \mathbb{R}^n$. But then according to (11) $\frac{d}{dt}f(\mathbf{r}_1 + t(\mathbf{r}_2 - \mathbf{r}_1)) \geq 0$ whenever $\mathbf{r}_2 - \mathbf{r}_1 \geq 0$, which is the definition of monotonicity w.r.t. the usual orthant order, and implies that f is $\mathbb{S}$ -monotone. ■

The following theorem and lemma are instead used in the section “Structural balance and monotonicity for DNNs” of the Results.

Theorem 1 *Consider a DNN (6) in which all the activation functions have nonnegative (generalized) derivative. The DNN map $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is monotone iff $\mathcal{G}(A)$ is structurally balanced.*

Proof of Theorem 1. In the DNN (4), using the chain rule for differentiation, at the ℓ -th layer we have

$$\frac{\partial \mathbf{z}_\ell}{\partial \mathbf{z}_{\ell-1}} = \frac{\partial \sigma(\mathbf{q}_\ell)}{\partial \mathbf{q}_\ell} \frac{\partial \mathbf{q}_\ell}{\partial \mathbf{z}_{\ell-1}}$$

where $\mathbf{q}_\ell = W^{(\ell)}\mathbf{z}_{\ell-1} + \mathbf{b}^{(\ell)}$ is the pre-activation state. Since by assumption $\frac{\partial\sigma(\mathbf{q}_\ell)}{\partial\mathbf{q}_\ell} = \text{diag}\left(\frac{\partial\sigma(\mathbf{q}_{\ell,i})}{\partial\mathbf{q}_{\ell,i}}\right) \geq 0$ (elementwise inequality) holds for all $\mathbf{q}_\ell$ (when the partial derivative is not uniquely defined, like in 0 for ReLU, it can be replaced by a differential inclusion, but the inequality still holds) and since $\frac{\partial\mathbf{q}_\ell}{\partial\mathbf{z}_{\ell-1}} = W^{(\ell)}$, then it must be $\text{sgn}\left(\frac{\partial\mathbf{z}_\ell}{\partial\mathbf{z}_{\ell-1}}\right) = \text{sgn}(W^{(\ell)})$ wherever $\frac{\partial\mathbf{z}_\ell}{\partial\mathbf{z}_{\ell-1}}$ has a nonvanishing entry. Extra zeros in $\frac{\partial\mathbf{z}_\ell}{\partial\mathbf{z}_{\ell-1}}$ can happen because of $\frac{\partial\sigma(\mathbf{q}_\ell)}{\partial\mathbf{q}_\ell}$: $\left[\frac{\partial\mathbf{z}_\ell}{\partial\mathbf{z}_{\ell-1}}\right]_{ij} \neq 0$ implies $[W^{(\ell)}]_{ij} \neq 0$ but not viceversa.

Assembling the layers as in (5) and computing the $n \times n$ Jacobian matrix of the DNN (6), call it $F(\mathbf{z}) = \frac{\partial f}{\partial \mathbf{z}}(A\mathbf{z} + \mathbf{b})$, then from the previous argument we get

$$\begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ \frac{\partial\mathbf{z}_1}{\partial\mathbf{x}} & 0 & 0 & \dots & 0 \\ 0 & \frac{\partial\mathbf{z}_2}{\partial\mathbf{z}_1} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{\partial\mathbf{y}}{\partial\mathbf{z}_{h-1}} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ \frac{\partial\sigma(\mathbf{q}_1)}{\partial\mathbf{q}_1}W^{(1)} & 0 & 0 & \dots & 0 & 0 \\ & \frac{\partial\sigma(\mathbf{q}_2)}{\partial\mathbf{q}_2}W^{(2)} & 0 & \dots & 0 & \\ \vdots & \vdots & \ddots & & & \\ \vdots & \vdots & & \frac{\partial\sigma(\mathbf{q}_{h-1})}{\partial\mathbf{q}_{h-1}}W^{(h-1)} & \vdots & \\ 0 & 0 & & \dots & W^{(h)} & 0 \end{bmatrix}$$

i.e.,

$$F(\mathbf{z}) = \underbrace{\text{diag}\left(0, \frac{\partial\sigma(\mathbf{q}_1)}{\partial\mathbf{q}_1}, \frac{\partial\sigma(\mathbf{q}_2)}{\partial\mathbf{q}_2}, \dots, \frac{\partial\sigma(\mathbf{q}_{h-1})}{\partial\mathbf{q}_{h-1}}, I\right)}_{\geq 0} A, \quad \forall \mathbf{z} \in \mathbb{R}^n \quad (12)$$

Recall from Proposition 1 that f is monotone with respect to some orthant order iff $\mathcal{G}(F(\mathbf{z}))$ is structurally balanced. Combining with (12), we have that f is monotone iff $\mathcal{G}(A)$ is structurally balanced. ■

From Eq. (12) in the proof of Theorem 1, it is worth highlighting the following property of a DNN f , monotone or not. Denote $F(\mathbf{z}) = \frac{\partial f}{\partial \mathbf{z}}(A\mathbf{z} + \mathbf{b})$ the Jacobian of f at $\mathbf{z}$.

Lemma 1 *In a DNN (6) in which all the activation functions have nonnegative (generalized) derivative, it is $\text{sgn}(F(\mathbf{z})) = \text{sgn}(\text{diag}(\mathbb{I}(\mathbf{z}))A)$.*

The sufficient but not necessary condition linking the structural balance of the graph $\mathcal{G}(A)$ to the monotonicity of the IO map of the DNN used in Section “Relation with IO monotonicity” of the Results is formally proven in the following theorem and corollary.

Theorem 2 *Consider a DNN (6) in which all the activation functions have nonnegative (generalized) derivative. If the subgraph $\mathcal{G}(A)$ is structurally balanced then the IO map $\mathbf{y} = \phi(\mathbf{x}, \bar{\boldsymbol{\theta}})$ is IO monotone.*

Proof of Theorem 2. Let A^h be the h -th power of A . If $\mathcal{G}(A)$ is structurally balanced, then so is $\mathcal{G}(A^h)$ and with the same gauge transformation. In fact, if $SA^hS \geq 0$, then $(SAS)^h = SA^hS \geq 0$, since $S^{-1} = S$ and $S^2 = I$. In absence of residual connections, the

gauge transformed adjacency matrix SAS has lower diagonal blocks $S^{(\ell)}W^{(\ell)}S^{(\ell-1)} \geq 0$, where $S^{(\ell)}$ is the diagonal block of S associated with the ℓ -th layer. Furthermore, since

$$A^h = \begin{bmatrix} & 0 & & 0 & \dots & 0 \\ & \vdots & & & & \\ W^{(h)}W^{(h-1)} & \dots & W^{(2)}W^{(1)} & 0 & \dots & 0 \end{bmatrix},$$

from $SA^hS \geq 0$ it must be $S^{(h)}W^{(h)}W^{(h-1)} \dots W^{(2)}W^{(1)}S^{(0)} \geq 0$, where $S^{(0)}$ and $S^{(h)}$ are the diagonal subblocks of S associated to input and output layers, $S^{(0)} = \text{diag}(\mathbf{s}_x)$, $S^{(h)} = \text{diag}(\mathbf{s}_y)$.

Following arguments similar to those used in the proof of Theorem 1, let us apply the gauge transformation in input and output space. Denote $\mathbf{r} = S\mathbf{z}$ and $\mathbf{p} = S\mathbf{q}$. If $\mathbf{u} = S^{(0)}\mathbf{x} = \mathbf{r}_0$ and $\mathbf{v} = S^{(h)}\mathbf{y} = \mathbf{r}_h$, then after the gauge transformation the input-output map becomes $\mathbf{v} = \phi_S(\mathbf{u}, \bar{\boldsymbol{\theta}})$, or extensively,

$$\begin{aligned} \mathbf{v} &= S^{(h)}W^{(h)}S^{(h-1)}\mathbf{r}_{h-1} + S^{(h)}\mathbf{b}^{(h)} \\ \mathbf{r}_\ell &= \sigma(\mathbf{p}_\ell) = \sigma(S^{(\ell)}W^{(\ell)}S^{(\ell-1)}\mathbf{r}_{\ell-1} + S^{(\ell)}\mathbf{b}^{(\ell)}), \quad \ell = 1, \dots, h-1. \end{aligned}$$

At a point $\mathbf{u}_1$ let us consider an increment $\mathbf{u}_2 - \mathbf{u}_1 > 0$ and compute the directional derivative along $\mathbf{u}_2 - \mathbf{u}_1$:

$$\frac{d}{dt}\phi_S(\mathbf{u}_1 + t(\mathbf{u}_2 - \mathbf{u}_1), \bar{\boldsymbol{\theta}}) = \frac{\partial \phi_S((\mathbf{u}_1 + t(\mathbf{u}_2 - \mathbf{u}_1)), \bar{\boldsymbol{\theta}})}{\partial \mathbf{u}}(\mathbf{u}_2 - \mathbf{u}_1) \quad (13)$$

The Jacobian matrix can be expressed as

$$\begin{aligned} \frac{\partial \phi_S}{\partial \mathbf{u}} &= \frac{\partial \mathbf{r}_h}{\partial \mathbf{p}_h} \frac{\partial \mathbf{p}_h}{\partial \mathbf{r}_{h-1}} \frac{\partial \mathbf{r}_{h-1}}{\partial \mathbf{p}_{h-1}} \frac{\partial \mathbf{p}_{h-1}}{\partial \mathbf{r}_{h-2}} \dots \frac{\partial \mathbf{p}_2}{\partial \mathbf{r}_1} \frac{\partial \mathbf{r}_1}{\partial \mathbf{p}_1} \frac{\partial \mathbf{p}_1}{\partial \mathbf{u}} \\ &= S^{(h)}W^{(h)}S^{(h-1)} \frac{\partial \sigma(\mathbf{p}_{h-1})}{\partial \mathbf{p}_{h-1}} S^{(h-1)}W^{(h-1)}S^{(h-2)} \dots S^{(2)}W^{(2)}S^{(1)} \frac{\partial \sigma(\mathbf{p}_1)}{\partial \mathbf{p}_1} S^{(1)}W^{(1)}S^{(0)} \end{aligned}$$

Since $\frac{\partial \sigma(\mathbf{p}_\ell)}{\partial \mathbf{p}_\ell} = \text{diag}\left(\frac{\partial \sigma(\mathbf{p}_{\ell,i})}{\partial \mathbf{p}_{\ell,i}}\right) \geq 0$ and $(S^{(\ell)})^2 = I$, $\frac{\partial \phi_S}{\partial \mathbf{u}}$ has the same sign pattern as $S^{(h)}W^{(h)}W^{(h-1)} \dots W^{(2)}W^{(1)}S^{(0)}$, which we have shown above to be nonnegative. Hence in (13), since $\mathbf{u}_2 - \mathbf{u}_1 \geq 0$, it must be $\frac{d}{dt}\phi_S(\mathbf{u}_1 + t(\mathbf{u}_2 - \mathbf{u}_1), \bar{\boldsymbol{\theta}}) \geq 0$, i.e., along any growing direction of the input the output grows. The argument implies that the map ϕ_S is IO monotone with input and output orders which are the usual ones in $\mathbb{R}^{n_0}$ and $\mathbb{R}^{n_h}$. In the original basis, ϕ is therefore IO monotone with orders $\mathbb{S}_x$ and $\mathbb{S}_y$. ■

The proof holds also when there are multiple branches in the DNN (e.g. in presence of residual connections). With the modifications made to (4) for this case (see last paragraph of Section D.1 below), more terms will appear in the expression for $\frac{\partial \phi_S}{\partial \mathbf{u}}$, all of which are non-negative due to $SAS \geq 0$ and $\frac{\partial \sigma(\mathbf{p}_\ell)}{\partial \mathbf{p}_\ell} \geq 0$.

Combining Theorem 1 with Theorem 2 we have the following corollary.

Corollary 1 *Under the assumptions of Theorem 2, if the DNN map $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is monotone then the IO map $\mathbf{y} = \phi(\mathbf{x}, \bar{\boldsymbol{\theta}})$ is IO monotone.*

B Heuristic algorithm for computing frustration: pseudocode

Algorithm 1 provides the pseudocode for the heuristic procedure mentioned in Section “Heuristic algorithm for computing frustration” of the Methods.

Consider the weighted adjacency matrix A and its symmetrized version $A_u = A + A^\top$. In terms of (3) of the paper, the algorithm is initialized at $S = \text{diag}(\mathbf{1})$ and follows a basic gradient descent rule in S . It works by computing the row/column sums in SA_uS and by flipping the sign to the most negative such row/column sum. If this corresponds to the i th row/column, then the sign flipping is accomplished by flipping the sign of the i -th entry in the diagonal of S . In this way the sum $\mathbf{1}^\top SA_u S \mathbf{1}$ increases monotonically (i.e., $e(\mathbf{s})$ in (3) decreases monotonically) as long as there are negative rows/columns in SA_uS . It stops when SA_uS no longer has negative rows/columns. As the algorithm is heuristic, this is a local minimum in the energy landscape functional $e(\mathbf{s})$. The local search can be repeated from different initializations, corresponding to choosing different initial $S = \text{diag}(\mathbf{s})$. The best among such local minima is our approximation of the frustration of A .

Algorithm 1 Frustration heuristic

Input: Adjacency matrix $A \in \mathbb{R}^{n \times n}$ of DNN

Input: Number of initial random sign flips, $\nu \geq 0$

Input: Maximum number of iterations, M

$A_u \leftarrow A + A^\top$

$S \leftarrow$ gauge transf. with ν negative entries chosen at random

rowsum $\leftarrow SA_u S \mathbf{1}$

for M iterations **do**

if $\nexists k : \text{rowsum}[k] < 0$ **then**

break

end if

$i \leftarrow \arg \min_k \text{rowsum}[k]$

$\triangleright$ Optionally: choose any i s.t. $\text{rowsum}[i] < 0$

$S_{ii} \leftarrow -S_{ii}$

 rowsum $\leftarrow SA_u S \mathbf{1}$

end for

$\alpha \leftarrow 1 / \sum_{i,j} [|A_u|]_{ij}$

Return: $\hat{e} \leftarrow \frac{1}{2} (1 - \alpha \mathbf{1}^\top SA_u S \mathbf{1})$

Notice that the algorithm provides a simple way to localize edges and nodes involved in frustrated cycles, as they correspond to the residual negative edges remaining in the best local minimum matrix SA_uS . In fact, the optimal spin assignment in such local minimum is simply $\mathbf{1}$. Reconstructing the original optimal spin assignment in the original basis is also straightforward, as it corresponds to the diagonal entries of the matrix S of all “accumulated” single-spin gauge transformations.

C Pretrained models

The pretrained models we use are retrieved from PyTorch Hub and MATLAB’s Deep Learning Toolbox.³ From MATLAB we get ShuffleNet, SqueezeNet, ResNet18, AlexNet and GoogLeNet, and from PyTorch we get versions of AlexNet and GoogLeNet with distinct weights and slightly different architecture. The version of ResNet18 available from PyTorch is identical to its counterpart in MATLAB, and we therefore disregard it. The networks have learnable parameters in the span of 1-60 million, but when constructing the adjacency matrices for the networks, keeping in mind that the weights are repeatedly used in convolutions, the number of nodes is in the order of millions, and the number of edges is in the order of billions. See Table 1 for the exact numbers.

Table 1: Sizes of neural networks used in the experiments. Number of nodes in the adjacency matrix A , number of edges (entries in A), and number of learnable parameters of the neural networks.

Network	n. nodes	n. edges	n. param.	n. edges	n. edges
	($\times 10^6$)	($\times 10^9$)	($\times 10^6$)	n. nodes	n. param.
ShuffleNet	3.91	0.28	1.4	72	200.9
SqueezeNet	3.27	0.38	1.2	116	306.4
ResNet18	3.01	2.80	11.7	928	239.3
AlexNet	0.94	0.67	61.0	718	11.0
AlexNet (PyTorch)	0.73	0.66	61.1	899	10.8
GoogLeNet	4.80	1.52	7.0	316	216.7
GoogLeNet (PyTorch)	4.80	1.45	6.6	302	218.1

D Construction of the adjacency matrices of the DNNs

In (4) a layer is defined as contributing a matrix of weights $W^{(\ell)}$, a vector of biases $\mathbf{b}^{(\ell)}$ and an activation function $\sigma(\cdot)$. In practice, however, it is common in the literature to refer to each operation that appears in the DNN as a “layer”. In particular, the DNNs we consider in this study consist of the following “layers”: (group) convolutional, dense, max-pooling, average pooling, ReLU, softmax, batch normalization, channel normalization, dropout, and also “layers” which are ways of joining parallel branches of the network, such as depth concatenation and addition layers. We now give some detail on how each of them can be mapped into the representation (4) of the paper. The aim is to provide a precise and reproducible way to construct an adjacency matrix for the DNN, accounting for all layers of the DNNs we use in the experiments. Notice that this sometimes requires us to overload a bit the notation of (4).

³<https://www.mathworks.com/help/deeplearning/ref/imagepretrainednetwork.html>, https://pytorch.org/hub/pytorch_vision_alexnet, and https://pytorch.org/hub/pytorch_vision_googlenet

D.1 Construction of the weight matrices

In this section we detail how the weight matrices of dense, convolutional and pooling layers are constructed for the real network. In our setting, layers such as ReLU, normalization and softmax do not contribute any weights to the representation (5), see Section D.2 for how activation functions are treated. For instance, the normalization layers only scale preactivations with a positive number (and possibly add an offset term), and they do not change the size of the layer input like pooling layers can. The pooling layers contribute both (non-trainable) weight matrices, discussed in this section, and an operation represented as an activation, discussed in Section D.2.

In dense (i.e., fully connected) layers, the weight matrices are obtained directly from their usual representation as matrix-vector multiplication and no activation function is present, i.e., $\mathbf{z}_\ell = \mathbf{q}_\ell = W^{(\ell)}\mathbf{z}_{\ell-1} + \mathbf{b}^{(\ell)}$ with $W^{(\ell)}$ full.

Convolutional layers can also be represented in this way, with $W^{(\ell)}$ which are generalized Toeplitz matrices with sparse structure characterized by repeated entries patterned along shifting rows/columns according to the input size and the stride of each convolutional filter. More in detail, consider a convolutional layer (index ℓ) and denote its weight matrix $W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$. For this layer, denote the input and output (without activation) tensors $\mathbf{t}_{\ell-1}$ and $\mathbf{t}_\ell$, and let the number of elements in these tensors be $n_{\ell-1}$ and n_ℓ respectively. In the adjacency matrix of the entire network each element corresponds to a node in the graph. To obtain $W^{(\ell)}$ we vectorize the input and output tensors, and put on each row of $W^{(\ell)}$ the entries of the kernel such that $\text{vec}(\mathbf{t}_\ell) = W^{(\ell)}\text{vec}(\mathbf{t}_{\ell-1})$, see the sketch in Fig. 1.

A more general convolutional layer is the so-called grouped convolutional layer, which separates the input into even groups along the channel/depth dimension, and convolves each group with different filters (allowing for parallel computation) whereafter these output groups are concatenated. Constructing the weight matrix for group convolutions is otherwise entirely analogous to a regular convolution. One can shuffle the output channels of a group convolutional layer, as in ShuffleNet, to avoid disentangling the channel groups entirely. Such a channel shuffling layer reorders the $g \cdot m$ channels into g groups of m channels, and it outputs m groups of g channels, where the i -th channel in output group j is the j -th channel of input group i [45]. When constructing the weight matrix of a layer followed by channel shuffling we reorder the rows of $W^{(\ell)}$ to account for this.

Since pooling operations are convolutions of the feature map with a (max or average) pooling window, the adjacency matrix for a pooling layer is constructed analogously to that of a convolution layer. Denote $\mathcal{N}_{\ell,i}$ the pool of node i at the ℓ -th layer. If the pool is a window of size $q \times q$, then its cardinality is $|\mathcal{N}_{\ell,i}| = q^2$. We choose to associate to a pooling layer a matrix $W^{(\ell)}$ whose rows contain up to $|\mathcal{N}_{\ell,i}|$ nonzero fixed (i.e., not trainable) and identical entries in correspondence of the group of nodes which form the pool of node i , and $\mathbf{b}^{(\ell)} = 0$. No distinction is made between average and max-pooling when we consider the entire real networks A (which are independent of the input $\mathbf{x}$), while their behavior differ when we look at active subnetworks $A_{\text{act}}(\mathbf{x})$ in response to a specific input $\mathbf{x}$, see Section D.3 of this SI. For a pooling window of size $p \times p$, the pooling weights are set to $0.01/p$ to be roughly the same order of magnitude as the neural network weights.

If a convolutional or pooling layer uses zero padding, we do not consider the padded elements as part of the input tensor, as they do not contribute extra edges and they do not appear in the adjacency matrix. Since zero padding edges are not present, in practice

some rows of $W^{(\ell)}$ have fewer edges than the kernel size.

The weight matrices are then positioned in the adjacency matrix A according to the topology of the neural network. When there are no residual connections, then the structure of A is the one shown in (5) of the paper. When residual connections are present, i.e., feedforward connections from layers $k_1, \dots, k_r$ to layer ℓ , $k_i < \ell - 1$, in place of (4) in the paper, the output of layer ℓ is expressed as $\mathbf{z}_\ell = \sigma \left(W^{(\ell)} \mathbf{z}_{\ell-1} + \mathbf{b}^{(\ell)} + \sum_{i=1}^r \mathbf{z}_{k_i} \right)$. In the adjacency matrix, this means that the weight matrices of layers $k_1, \dots, k_r$ appear in the block-row corresponding to layer ℓ , that is, non-zero blocks of weights appear below the lower diagonal blocks in A , see visualizations of the adjacency matrices in Fig. 6, especially items (a) ShuffleNet and (e) ResNet18, where many residual connections are present. Similarly, if the input to layer ℓ is a concatenation of the outputs of layers $k_1, \dots, k_r$, we have $\mathbf{z}_\ell = \sigma \left(W^{(\ell)} \begin{bmatrix} \mathbf{z}_{k_1}^\top & \dots & \mathbf{z}_{k_r}^\top \end{bmatrix}^\top + \mathbf{b}^{(\ell)} \right)$, and again the weight matrices appear on the block-row of layer ℓ , but with each in its own sub-block since now $W^{(\ell)} \in \mathbb{R}^{n_\ell \times (n_{k_1} + \dots + n_{k_r})}$. In comparison, for addition layer it is $W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $n_{k_1} = \dots = n_{k_r} = n_{\ell-1}$.

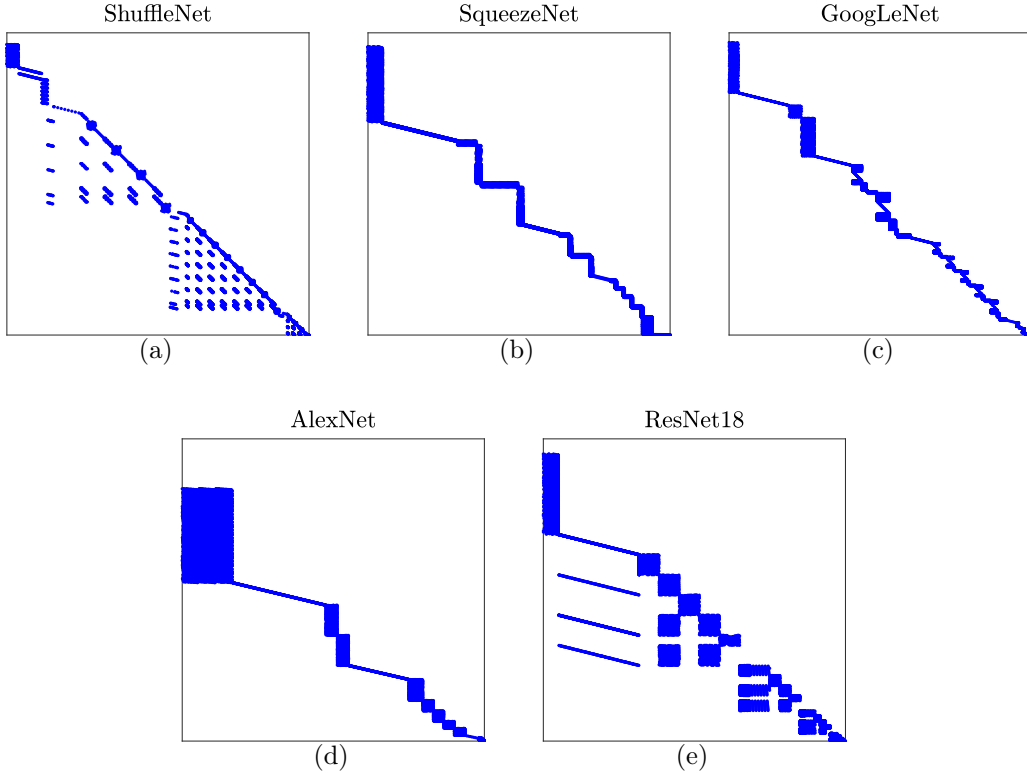


Figure 6: Adjacency matrix layout for (a) ShuffleNet, (b) SqueezeNet, (c) GoogLeNet, (d) AlexNet, and (e) ResNet18, slightly downsampled for rendering.

D.2 A detailed expression for the nonlinearities at the activation functions

Operations occurring in layers such as pooling, ReLU, normalization and softmax can all be interpreted in terms of the activation function $\sigma(\cdot)$.

A pooling layer consists of a weight matrix, capturing the neighborhood of a node, and composed as described above, plus a pooling operation which we express as a special case of $\sigma(\cdot)$. In max-pooling layers, the max operation is:

$$z_{\ell,i} = \sigma(\mathbf{z}_{\ell-1}) = \max_{j \in \mathcal{N}_{\ell,i}} z_{\ell-1,j}$$

where $\mathcal{N}_{\ell,i}$ is the pool of node i . When the pooling operation is averaging, we have instead

$$z_{\ell,i} = \sigma(\mathbf{z}_{\ell-1}) = \frac{1}{|\mathcal{N}_{\ell,i}|} \sum_{j \in \mathcal{N}_{\ell,i}} z_{\ell-1,j}.$$

Notice that for max-pooling, the overall expression is at odds with the formulation (4), in the sense that no matrix-vector multiplication is performed when taking max, and the term $W^{(\ell)}\mathbf{z}_{\ell-1}$ should be replaced by a nonlinear dependence $W^{(\ell)}(\mathbf{z}_{\ell-1})$.

In a ReLU layer, for each component of σ we can represent the activation function as

$$z_{\ell,i} = \sigma(q_{\ell,i}) = \begin{cases} q_{\ell,i} & \text{if } q_{\ell,i} > 0 \\ 0 & \text{if } q_{\ell,i} \leq 0 \end{cases}$$

where $q_{\ell,i}$ is the preactivation state of the layer that precedes the ReLU. As a ReLU operation happens on a node, there is no edge weight associated to it.

A batch normalization layer can be represented as

$$z_{\ell,i} = \sigma(q_{\ell,i}) = \frac{\gamma(q_{\ell,i} - \mu_{\mathcal{B}}(\mathbf{q}_{\ell}))}{\sqrt{\varsigma_{\mathcal{B}}^2(\mathbf{q}_{\ell}) + \varepsilon}} + \beta$$

where $\mu_{\mathcal{B}}(\mathbf{q}_{\ell})$ and $\varsigma_{\mathcal{B}}^2(\mathbf{q}_{\ell})$ are the mean and variance of $\mathbf{q}_{\ell}$ in a batch during training (replaced by statistics of the dataset during inference), and $\gamma > 0$, and $\beta \in \mathbb{R}$ are parameters learnt during training.

A local response normalization, or cross-channel normalization, was used in [22] to promote generalization, and it is part of the MATLAB versions of AlexNet and GoogLeNet architectures that we consider in this paper. The layer normalizes the input locally along the channel (third) dimension and is represented by

$$z_{\ell,i} = \sigma(\mathbf{z}_{\ell-1})_i = \frac{z_{\ell-1,i}}{K_{\ell-1,i}^{\beta}}, \quad K_{\ell-1,i} = \left(K + \frac{\alpha}{w} \sum_{j \in \mathcal{N}_{\ell,i}} z_{\ell-1,j}^2 \right),$$

where α, β, K are positive parameters, $\mathcal{N}_{\ell,i}$ are the node indices in the local normalization window around and including i , and $w = |\mathcal{N}_{\ell,i}|$ is the normalization window size. It should be noted that

$$\begin{aligned} \frac{\partial \sigma(\mathbf{z}_{\ell-1})_i}{z_{\ell-1,i}} &= \frac{1}{K_{\ell-1,i}^{\beta}} \left(1 - \frac{2\alpha\beta}{wK_{\ell-1,i}} z_{\ell-1,i}^2 \right) \\ \frac{\partial \sigma(\mathbf{z}_{\ell-1})_i}{z_{\ell-1,j}} &= -\frac{2\alpha\beta}{wK_{\ell-1,i}^{\beta+1}} z_{\ell-1,i} z_{\ell-1,j}, \quad j \in \mathcal{N}_{\ell,i} \setminus \{i\} \end{aligned}$$

which can be negative quantities. However, in practice, the elements of $\mathbf{z}_{\ell-1}$ (which have been passed through ReLU activations) have magnitudes in the order $10^0 \sim 10^2$,

and with the parameters $\alpha = 10^{-4}$, $\beta = 0.75$, $w = 5$, $K = 1$, it usually means $\frac{2\alpha\beta z_{\ell-1,i}^2}{wK_{\ell-1,i}}, \frac{2\alpha\beta z_{\ell-1,i}z_{\ell-1,j}}{wK_{\ell-1,i}} \ll 1$, i.e., that $1 \approx \frac{\partial\sigma(\mathbf{z}_{\ell-1})_i}{\partial z_{\ell-1,i}} > 0$, and $\frac{\partial\sigma(\mathbf{z}_{\ell-1})_i}{\partial z_{\ell-1,i}} \gg \frac{\partial\sigma(\mathbf{z}_{\ell-1})_i}{\partial z_{\ell-1,j}} \approx 0$. Hence, $\frac{\partial\sigma(\mathbf{z}_{\ell-1})}{\partial \mathbf{z}_{\ell-1}} \approx \text{diag}\left(\frac{\partial\sigma(\mathbf{z}_{\ell-1})_i}{\partial z_{\ell-1,i}}\right) \geq 0$ is a reasonable approximation.

When training a DNN on classification tasks, also the final layer is endowed with an activation function which is usually the softmax function. In Eq. (4) and (6) of the paper and in the computation of A this layer is ignored, but it is needed when computing the activation adjacency submatrix $A_{\text{act}}(\mathbf{x})$ of a specific input. In terms of σ , the softmax function is

$$\mathbf{y} = \sigma(\mathbf{q}_h) = \frac{e^{\mathbf{q}_h}}{\sum_{k=1}^{n_h} e^{q_{h,k}}}$$

which normalizes the output to $\mathbf{1}^\top \mathbf{y} = 1$, $\mathbf{y} \geq 0$, so that it can be interpreted as class probabilities.

In the paper, all these types of operations are indistinctively denoted with the activation function symbol σ .

D.3 Some details for the construction of the activation adjacency submatrix

To construct the adjacency matrix of the active subnetwork induced by a specific input, we give the DNN an input image $\mathbf{x}$ and record the hidden states $\mathbf{z}$. Starting from the real network, we drop all nodes that are set to zero by ReLU activations and also all edges adjacent to these nodes. All edges that are not active in max-pooling operations are also dropped, i.e., for max-pooling at node i in layer ℓ we have $z_{\ell,i} = \max_{j \in \mathcal{N}_{\ell,i}} z_{\ell-1,j}$, and we retain only the edges that connect node i in layer ℓ to nodes $k \in \arg \max_{j \in \mathcal{N}_{\ell,i}} z_{\ell-1,j}$ in layers connecting to layer ℓ . Removing edges between a max-pooling layer and its input layers may lead to nodes in previous layers that are disconnected from the output. We remove such dead-end nodes (and adjacent edges) from the network. For the final layer, we keep only the one node corresponding to the predicted class.

In the paper, the activation pattern in response to an image $\mathbf{x}$ is indicated as $\mathbb{I}(\mathbf{z})$. In practice, this is a bit of an oversimplification. To be more precise, the pattern $\mathbb{I}(\cdot)$ depends on the layer type. For convolutional, dense, and average pooling layers (and no residual connections) we get

$$W_{\text{act}}^{(\ell)}(\mathbf{x}) = \text{diag}(\mathbb{I}(W^{(\ell)}\mathbf{z}_{\ell-1} + \mathbf{b}^{(\ell)})) W^{(\ell)} \text{diag}(\mathbb{I}(W^{(\ell-1)}\mathbf{z}_{\ell-2} + \mathbf{b}^{(\ell-1)})).$$

In presence of residual connections, the argument of $\mathbb{I}(\cdot)$ must be modified accordingly. As mentioned above, for max-pooling layers we retain only the edges between the output node i and the input node(s) with the maximum value in the pooling window $\mathcal{N}_{\ell,i}$, i.e., the edge(s) which pass on the state. Recall that on the full network specifying this properly requires to replace a linear matrix-vector product like $W^{(\ell)}\mathbf{z}_{\ell-1}$ with a nonlinear dependence $W^{(\ell)}(\mathbf{z}_{\ell-1})$. In the same spirit, the activation submatrix of max-pooling layers can be written as

$$[W_{\text{act}}^{(\ell)}(\mathbf{z}_\ell)]_{ij} = \begin{cases} [W^{(\ell)}]_{ij} & \text{if } j = \arg \max_{k \in \mathcal{N}_i} z_{\ell,k} \\ 0, & \text{otherwise} \end{cases}$$

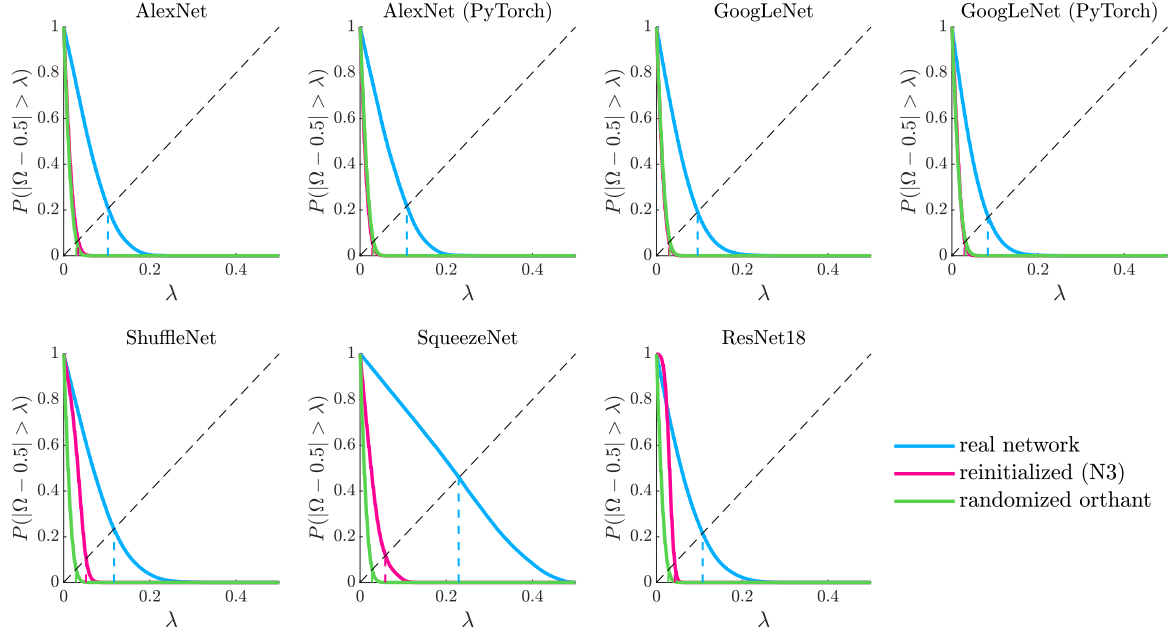


Figure 7: Quantification of the λ coefficient in (10), for the seven CNNs. In order to compute the output alignment fraction Ω , for each network we consider 100 perturbations for each of 1000 different images. For any value of λ , the colored line shows the cumulative count of the fraction of cases in which Ω deviates from 0.5 by more than λ , i.e., $P(|\Omega - 0.5| > \lambda)$. The IO-function of the network is λ -monotone if $P(|\Omega - 0.5| > \lambda) \geq 2\lambda$, where the 2λ line is the dashed black line in the figure. The intersection of this line with the cumulative curve gives the value of λ we seek.

If ReLU activations are present after the max-pooling, the non-activated nodes are dropped just as for other weight layers.

All networks we consider in this paper have softmax activation after the final layer. When constructing the final layer of the active subnetwork we keep only the edges connected to the node corresponding to the largest output, i.e., the predicted class. In all cases of interest, $\mathcal{G}(A_{\text{act}}(\mathbf{x}))$ contains connected components reaching the largest output in $\mathbf{y}$ from $\mathbf{x}$.

E Data availability and acknowledgments

The pretrained CNNs used in this study are all publicly available (with a licence, for the MATLAB CNNs), as specified in Section C of these SI. The Imagenette dataset was downloaded from <https://github.com/fastai/imagenette>. The code we developed can be made available upon request.

The computations done in the paper were in part enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

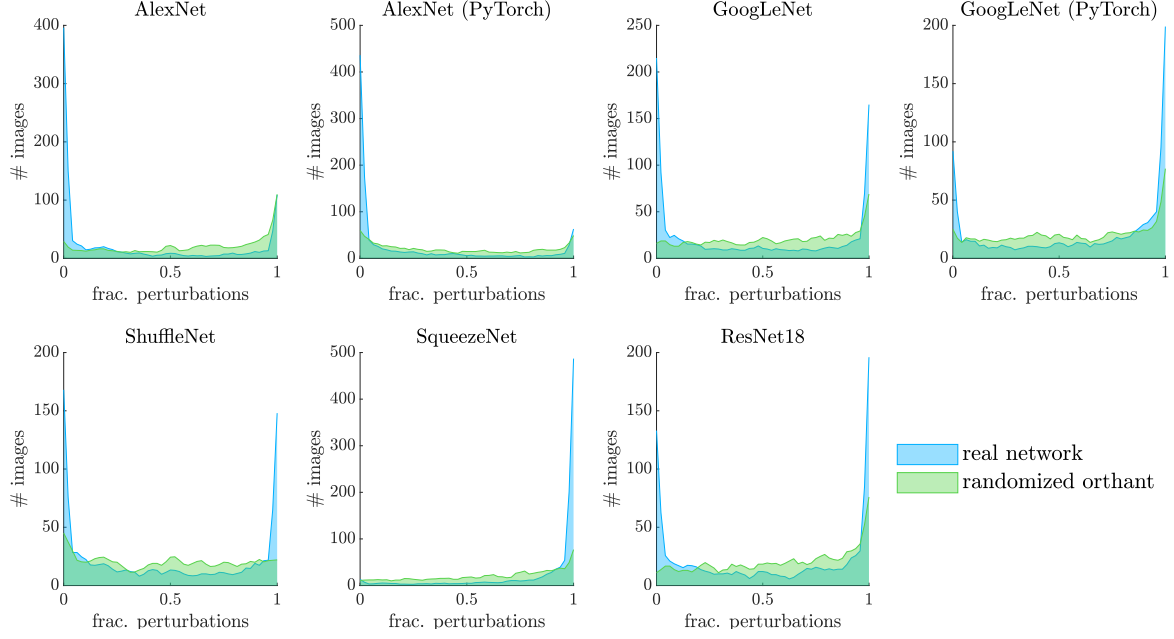


Figure 8: Fraction of perturbations fulfilling $\mathbf{y}_2 \geq_{\mathbb{S}_y} \mathbf{y}_1$ in more than 500 (of the 1000) output nodes. In total, 1000 images were used, and 100 perturbations were computed for each image. Values at 0 and 1 mean that all computed perturbations for that image resulted in outputs $\mathbf{y}_2 \leq_{\mathbb{S}_y} \mathbf{y}_1$, resp. $\mathbf{y}_2 \geq_{\mathbb{S}_y} \mathbf{y}_1$ in more than 500 elements. This corresponds to $\Omega < 0.5$ resp. $\Omega > 0.5$ for all perturbations in Fig. 4. For the real networks, the values are concentrated around 0 or 1, indicating that the number of elements increasing (decreasing) in $\mathbb{S}_y$ are always in majority, and do not switch to a majority decreasing (increasing). No such pattern is clearly visible when using random perturbation directions.