Input Similarity from the Neural Network Perspective

Anonymous Author(s)
Affiliation
Address
email

Abstract

We first exhibit a multimodal image registration task, for which a neural network trained on a dataset with noisy labels reaches almost perfect accuracy, far beyond noise variance. This surprising auto-denoising phenomenon can be explained as a noise averaging effect over the labels of similar input examples. This effect theoretically grows with the number of similar examples; the question is then to define and estimate the *similarity* of examples.f

We express a proper definition of similarity, from the neural network perspective, *i.e.* we quantify how undissociable two inputs A and B are, taking a machine learning viewpoint: how much a parameter variation designed to change the output for A would impact the output for B?

We study the mathematical properties of this similarity measure, and show how to use it on a trained network to estimate sample density, in low complexity, enabling new types of statistical analysis for neural networks. We also propose to use it during training, to enforce that examples known to be similar should also be seen as similar by the network, and notice speed-up training effects for certain datasets.

1 Motivation: Dataset self-denoising

In remote sensing imagery, data is abundant but noisy [14]. For instance RGB satellite images and binary cadaster maps (delineating buildings) are numerous but badly aligned for various reasons (annotation mistakes, atmosphere disturbance, elevation variations...). In a recent preliminary work [1], we tackled the task of automatically registering these two types of images together with neural networks, considering as ground truth a dataset of hand-picked relatively-well-aligned areas [13], and hoping the network would be able to learn from such a dataset of imperfect alignments. Learning with noisy labels is indeed an active topic of research [20, 15, 12].

For this, we designed an iterative approach: train, then test on the training set and re-align it accordingly; repeat (for 3 iterations). The results were surprisingly good, yielding far better alignments than the ground truth it learned from, both qualitatively (Figure 1) and quantitatively (Figure 2, obtained on manually-aligned data): the median registration error dropped from 18 pixels to 3.5 pixels, which is the best score one could hope for, given intrinsic ambiguities in such registration task. To check that this performance was not due to a subset of the training data that would be perfectly aligned, we added noise to the ground truth and re-trained from it: the new results were about as good again (dashed lines). Thus the network did learn almost perfectly just from noisy labels.

An explanation for this self-denoising phenomenon is proposed in [11] as follows. Let us consider a regression task, with a L^2 loss, and where true labels y were altered with i.i.d. noise ε of variance v. Suppose a same input $\mathbf{x}$ appears n times in the training set, thus with n different labels $y_i = y + \varepsilon_i$. The network can only output the same prediction for all these n cases (since the input is the same), and the best option, considering the L^2 loss, is to predict the average $\frac{1}{n}\sum_i y_i$, whose distance to the true label y is $O(\frac{v}{\sqrt{n}})$. Thus a denoising effect by a factor $\sqrt{n}$ can be observed. However, the exact same point $\mathbf{x}$ is not likely to appear several times in a dataset (with different labels). Rather, relatively *similar* points may appear, and the amplitude of the self-denoising effect will be a function of their

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

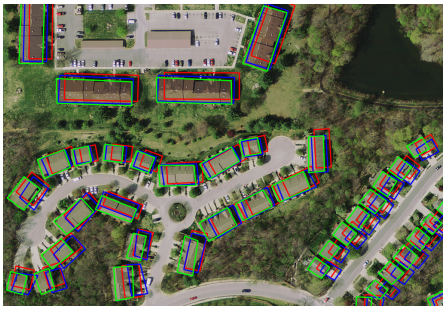

Figure 1: Qualitative alignment results [1] on a crop of bloomington22 from the Inria dataset [13]. Red: initial dataset annotations; blue: aligned annotations round 1; green: aligned annotations round 2.

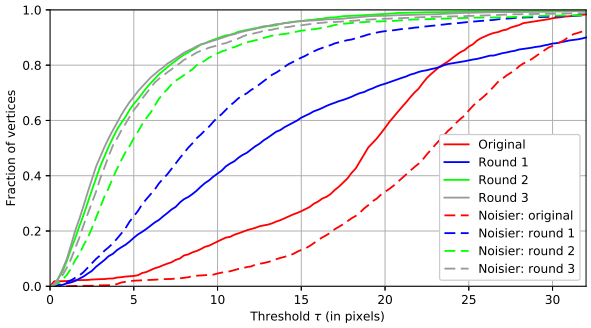

Figure 2: Accuracy cumulative distributions [1] measured with the manually-aligned annotations of bloomington22 [13]. Read as: fraction of image pixels whose registration error is less than threshold τ.

number. Here, the similarity should reflect the neural network perception (similar inputs yield the same output) and not an *a priori* norm chosen on the input space.

The purpose of this article is to express this notion of similarity, from the network's point of view. We first define it, and study it mathematically; we then compute the number of neighbors based on it, and propose for this a very fast estimator. This brings new tools to analyze already-trained networks. As they are differentiable and fast to compute, we show they can be used during training as well, for instance to enforce that given examples should be perceived as similar by the network. Finally, we apply the proposed tools to the initial remote sensing image alignment task.

2 Similarity

2.1 Notions of similarities

The notion of similarity between data points is an important topic in the machine learning literature, obviously in domains such as image retrieval, where images similar to a query have to be found; but not only. For instance when training auto-encoders, the quality of the reconstruction is usually quantified as the L^2 norm between the input and output images. Such a similarity measure is however questionable, as color comparison, performed pixel per pixel, is a poor estimate of human perception: the L^2 norm can vary a lot with transformations barely noticeable to the human eye such as small translations or rotations (for instance on textures), and does not carry semantic information, *i.e.* whether the same kind of objects are present in the image.

Therefore, so-called *perceptual losses* [10] were introduced to quantify image similarity: each image is fed to a standard pre-trained network such as VGG, and the activations in a particular intermediate layer are used as descriptors of the image [5, 6]. The distance between two images is then set as the L^2 norm between these activations. Such a distance carries implicitly semantic information, as the VGG network was trained for image classification. However, the choice of the layer to consider is arbitrary. In the ideal case, one would wish to combine the information from all layers, as some are more abstract and some more detail-specific. But then the particular weights chosen to combine the different layers would be also arbitrary. Would it be possible to get a canonical similarity measure, well posed theoretically? In this section we define a proper, intrinsic notion of similarity as seen by the network, relying on how easily it could furthermore distinguish different inputs.

2.2 Similarity from the point of view of the parameterized family of functions

Let f_θ be a parameterized function, typically a neural network already trained for some task, and $\mathbf{x}, \mathbf{x}'$ possible inputs, for instance from the training or test set. For the sake of simplicity, let us suppose in a first step that f_θ is real valued. To express the similarity between $\mathbf{x}$ and $\mathbf{x}'$, as seen by the network, one could compare the output values $f_\theta(\mathbf{x})$ and $f_\theta(\mathbf{x}')$. This is however not very informative, and a same output might be obtained for different reasons.

Instead, we define similarity as the influence of $\mathbf{x}$ over $\mathbf{x}'$, by quantifying how much a supplementary training step for $\mathbf{x}$ would change the output for $\mathbf{x}'$ as well. If $\mathbf{x}$ and $\mathbf{x}'$ are very different from the

point of view of the neural network, changing $f_\theta(\mathbf{x})$ will have little consequence on $f_\theta(\mathbf{x}')$. On the opposite, if they are very similar, changing $f_\theta(\mathbf{x})$ will greatly affect $f_\theta(\mathbf{x}')$ as well.

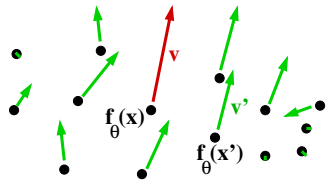

Figure 3: Moves in the space of outputs. We quantify the influence of a data point $\mathbf{x}$ over another one $\mathbf{x}'$ by how much the tuning of parameters θ to obtain a desired output change $\mathbf{v}$ for $f_\theta(\mathbf{x})$ will affect $f_\theta(\mathbf{x}')$ as well.

Formally, if one wants to change the value of $f_\theta(\mathbf{x})$ by a small quantity ε, one needs to update θ by $\delta\theta = \varepsilon \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2}$. Indeed, after the parameter update, the new value at $\mathbf{x}$ will be:

$$f_{\theta+\delta\theta}(\mathbf{x}) \;=\; f_\theta(\mathbf{x}) + \nabla_\theta f_\theta(\mathbf{x}) \cdot \delta\theta + O(\|\delta\theta\|^2) \;=\; f_\theta(\mathbf{x}) + \varepsilon + O(\varepsilon^2).$$

This parameter change induces a value change at any other point $\mathbf{x}'$:

$$f_{\theta+\delta\theta}(\mathbf{x}') \;=\; f_\theta(\mathbf{x}') + \nabla_\theta f_\theta(\mathbf{x}') \cdot \delta\theta + O(\|\delta\theta\|^2) \;=\; f_\theta(\mathbf{x}') + \varepsilon \frac{\nabla_\theta f_\theta(\mathbf{x}') \cdot \nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2} + O(\varepsilon^2).$$

Therefore the kernel $k_\theta^N(\mathbf{x}, \mathbf{x}') = \dfrac{\nabla_\theta f_\theta(\mathbf{x}) \cdot \nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2}$ represents the influence of $\mathbf{x}$ over $\mathbf{x}'$: if one wishes to change the output value $f_\theta(\mathbf{x})$ by ε, then $f_\theta(\mathbf{x}')$ will change by $\varepsilon\, k_\theta^N(\mathbf{x}, \mathbf{x}')$. In particular, if $k_\theta^N(\mathbf{x}, \mathbf{x}')$ is high, then $\mathbf{x}$ and $\mathbf{x}'$ are not distinguishable from the point of view of the network, as any attempt to move $f_\theta(\mathbf{x})$ will move $f_\theta(\mathbf{x}')$ as well (see Fig. 3). We thus see $k_\theta^N(\mathbf{x}, \mathbf{x}')$ as a measure of similarity. Note however that $k_\theta^N(\mathbf{x}, \mathbf{x}')$ is not symmetric.

Symmetric similarity: correlation Two symmetric kernels natural arise: the inner product:

$$k_\theta^I(\mathbf{x}, \mathbf{x}') \;=\; \nabla_\theta f_\theta(\mathbf{x}) \cdot \nabla_\theta f_\theta(\mathbf{x}') \tag{1}$$

and its normalized version, the correlation:

$$k_\theta^C(\mathbf{x}, \mathbf{x}') = \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} \cdot \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|} \tag{2}$$

which has the advantage of being bounded (in $[-1, 1]$), thus expressing similarity in a usual meaning.

2.3 Properties for vanilla neural networks

Intuitively, inputs that are similar from the network perspective should produce similar outputs; we can check that k_θ^C is a good similarity measure in this respect (all proofs are deferred to the supplementary materials):

Theorem 1. *For any real-valued neural network f_θ whose last layer is a linear layer (without any parameter sharing) or a standard activation function thereof (sigmoid, tanh, ReLU...), and for any inputs $\mathbf{x}$ and $\mathbf{x}'$,*

$$\nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}') \quad\Longrightarrow\quad f_\theta(\mathbf{x}) = f_\theta(\mathbf{x}')\,.$$

Corollary 1. *Under the same assumptions, for any inputs $\mathbf{x}$ and $\mathbf{x}'$,*

$$\begin{aligned} & k_\theta^C(\mathbf{x}, \mathbf{x}') = 1 \quad\Longrightarrow\quad \nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}')\,, \\ \text{hence}\quad & k_\theta^C(\mathbf{x}, \mathbf{x}') = 1 \quad\Longrightarrow\quad f_\theta(\mathbf{x}) = f_\theta(\mathbf{x}')\,. \end{aligned}$$

Furthermore,

Theorem 2. *For any real-valued neural network f_θ without parameter sharing, if $\nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}')$ for two inputs $\mathbf{x}, \mathbf{x}'$, then all useful activities computed when processing $\mathbf{x}$ are equal to the ones obtained when processing $\mathbf{x}'$.*

We name *useful* activities all activities $a_i(\mathbf{x})$ whose variation would have an impact on the output, *i.e.* all the ones satisfying $\frac{df_\theta(\mathbf{x})}{da_i} \neq 0$. This condition is typically not satisfied when the activity is negative and followed by a ReLU, or when it is multiplied by a 0 weight, or when all its contributions to the output annihilate together (*e.g.*, a sum of two neurons with opposite weights: $f_\theta(\mathbf{x}) = \sigma(a_i(\mathbf{x})) - \sigma(a_i(\mathbf{x}))$).

Link with the *perceptual loss* For a vanilla network without parameter sharing, the gradient $\nabla_\theta f_\theta(\mathbf{x})$ is a list of coefficients $\nabla_{w_i^j} f_\theta(\mathbf{x}) = \frac{df_\theta(\mathbf{x})}{db_j}\, a_i(\mathbf{x})$, where w_i^j is the parameter-factor that multiplies the input activation $a_i(\mathbf{x})$ in neuron j, and of coefficients $\nabla_{b_j} f_\theta(\mathbf{x}) = \frac{df_\theta(\mathbf{x})}{db_j}$ for neuron biases, which we will consider as standard parameters $b_j = w_0^j$ that act on a constant activation $a_0(\mathbf{x}) = 1$, yielding $\nabla_{w_0^j} f_\theta(\mathbf{x}) = \frac{df_\theta(\mathbf{x})}{db_j}\, a_0(\mathbf{x})$. Thus the gradient $\nabla_\theta f_\theta(\mathbf{x})$ can be seen as a list of all activation values $a_i(\mathbf{x})$ multiplied by the potential impact on the output $f_\theta(\mathbf{x})$ of the neurons j using them, *i.e.* $\frac{df_\theta(\mathbf{x})}{db_j}$. Each activation appears in this list as many times as it is fed to different neurons. The similarity between two inputs then rewrites:

$$k_\theta^I(\mathbf{x}, \mathbf{x}') = \sum_{\text{activities } i} \lambda_i(\mathbf{x}, \mathbf{x}')\; a_i(\mathbf{x})\, a_i(\mathbf{x}') \qquad \text{where} \qquad \lambda_i(\mathbf{x}, \mathbf{x}') = \sum_{\text{neuron } j \text{ using } a_i} \frac{df_\theta(\mathbf{x})}{db_j} \frac{df_\theta(\mathbf{x}')}{db_j}$$

are data-dependent importance weights. Such weighting schemes on activation units naturally arise when expressing intrinsic quantities; the use of natural gradients would bring invariance to re-parameterization [16, 17]. On the other hand, the inner product related to the perceptual loss would be

$$\sum_{\text{activities } i \neq 0} \lambda_{\text{layer}(i)}\; a_i(\mathbf{x})\, a_i(\mathbf{x}')$$

for some arbitrary fixed layer-dependent weights $\lambda_{\text{layer}(i)}$.

2.4 Properties for parameter-sharing networks

When sharing weights, as in convolutional networks, the gradient $\nabla_\theta f_\theta(\mathbf{x})$ is made of the same coefficients (impact-weighted activations) but summed over shared parameters. Denoting by $\mathcal{S}(i)$ the set of (neuron, input activity) pairs where the parameter w_i is involved,

$$k_\theta^I(\mathbf{x}, \mathbf{x}') \;=\; \sum_{\text{params } i} \left(\sum_{(j,k) \in \mathcal{S}_i} a_k(\mathbf{x}) \frac{df_\theta(\mathbf{x})}{db_j} \right) \left(\sum_{(j,k) \in \mathcal{S}_i} a_k(\mathbf{x}') \frac{df_\theta(\mathbf{x}')}{db_j} \right)$$

Thus, in convolutional networks, k_θ^I similarity does not imply similarity of first layer activations anymore, but only of their (impact-weighted) spatial average. More generally, any invariance introduced by a weight sharing scheme in an architecture will be reflected in the similarity measure k_θ^I, which is expected as k_θ^I was defined as the input similarity *from the neural network perspective*.

Note that this type of objects was recently studied from an optimization viewpoint under the name of Neural Tangent Kernel [9, 2] in the infinite layer width limit.

3 Higher output dimension

Let us now study the more complex case where $f_\theta(\mathbf{x})$ is a vector $\left(f_\theta^i(\mathbf{x})\right)_{i \in [1,d]}$ in $\mathbb{R}^d$ with $d > 1$. Under a mild hypothesis on the network (output expressivity), always satisfied unless specially designed not to:

Theorem 3. *The optimal parameter change $\delta\theta$ to push $f_\theta(\mathbf{x})$ in a direction $\mathbf{v} \in \mathbb{R}^d$ (with a force $\varepsilon \in \mathbb{R}$),* i.e. *such that $f_{\theta+\delta\theta}(\mathbf{x}) - f_\theta(\mathbf{x}) = \varepsilon \mathbf{v}$, induces at any other point $\mathbf{x}'$ the following output variation:*

$$f_{\theta+\delta\theta}(\mathbf{x}') - f_\theta(\mathbf{x}') = \varepsilon\, K_\theta(\mathbf{x}', \mathbf{x})\, K_\theta(\mathbf{x}, \mathbf{x})^{-1}\, \mathbf{v} \;+\; O(\varepsilon^2) \tag{3}$$

where the $d \times d$ kernel matrix $K_\theta(\mathbf{x}', \mathbf{x})$ is defined by $K_\theta^{ij}(\mathbf{x}', \mathbf{x}) = \nabla_\theta f_\theta^i(\mathbf{x}') \cdot \nabla_\theta f_\theta^j(\mathbf{x})$.

The similarity kernel is now a matrix and not just a single value, as it describes the relation between moves $\mathbf{v} \in \mathbb{R}^d$. Note that these matrices K_θ are only $d \times d$ where d is the output dimension. They are thus generally small and easy to manipulate or inverse.

Normalized similarity matrix The unitless symmetrized, normalized version of the kernel (3) is:

$$K_\theta^C(\mathbf{x}, \mathbf{x}') \;=\; K_\theta(\mathbf{x}, \mathbf{x})^{-1/2}\; K_\theta(\mathbf{x}, \mathbf{x}')\; K_\theta(\mathbf{x}', \mathbf{x}')^{-1/2}\,. \tag{4}$$

It has the following properties: its coefficients are bounded, in $[-1, 1]$; its trace is at most d; its (Frobenius) norm is at most $\sqrt{d}$; self-similarity is identity: $\forall \mathbf{x},\;\; K_\theta^C(\mathbf{x}, \mathbf{x}) = \mathrm{Id}$; the kernel is symmetric, in the sense that $K_\theta^C(\mathbf{x}', \mathbf{x}) = K_\theta^C(\mathbf{x}, \mathbf{x}')^T$.

Similarity in a single value To summarize the similarity matrix $K_\theta^C(\mathbf{x}, \mathbf{x}')$ into a single real value in $[-1, 1]$, we consider:

$$k_\theta^C(\mathbf{x}, \mathbf{x}') \;=\; \frac{1}{d} \operatorname{Tr} K_\theta^C(\mathbf{x}, \mathbf{x}') \;. \tag{5}$$

It can be shown indeed that if $k_\theta^C(\mathbf{x}, \mathbf{x}')$ is close to 1, then $K_\theta^C(\mathbf{x}, \mathbf{x}')$ is close to Id, and reciprocally. See the supplementary materials for more details and a discussion about the links between $\frac{1}{d} \operatorname{Tr} K_\theta^C(\mathbf{x}, \mathbf{x}')$ and $\left\| K_\theta^C(\mathbf{x}, \mathbf{x}') - \mathrm{Id} \right\|_F$.

Metrics on output: rotation invariance Similarity in $\mathbb{R}^d$ might be richer than just estimating distances in L^2 norm. For instance, for our 2D image registration task, the network could be known (or desired) to be equivariant to rotations. The similarity between two output variations $\mathbf{v}$ and $\mathbf{v}'$ can be made rotation-invariant by applying the rotation that best aligns $\mathbf{v}$ and $\mathbf{v}'$ beforehand. This can actually be easily computed in closed form and yields:

$$k_\theta^{C,\mathrm{rot}}(\mathbf{x}, \mathbf{x}') \;=\; \frac{1}{2} \sqrt{\left\| K_\theta^C(\mathbf{x}, \mathbf{x}') \right\|_F^2 + 2 \det K_\theta^C(\mathbf{x}, \mathbf{x}')} \;.$$

Note that other metrics are possible in the output space. For instance, the loss metric quantifies the norm of a move $\mathbf{v}$ by its impact on the loss $\frac{dL(y)}{dy}\big|_{f_\theta(\mathbf{x})}(\mathbf{v})$. It has a particular meaning though, and is not always relevant, *e.g.* in the noisy label case seen in Section 1.

The case of classification tasks When the output of the network is a probability distribution $p_{\theta,\mathbf{x}}(c)$, over a finite number of given classes c for example, it is natural from an information theoretic point of view to rather consider $f_\theta^c(\mathbf{x}) = -\log p_{\theta,\mathbf{x}}(c)$. This is actually the quantities computed in the pre-softmax layer from which common practice directly computes the cross-entropy loss.

It turns out that the L^2 norm of variations δf in this space naturally corresponds to the Fisher information metric, which quantifies the impact of parameter variations $\delta\theta$ on the output probability $p_{\theta,\mathbf{x}}$, as $\mathrm{KL}(p_{\theta,\mathbf{x}} || p_{\theta+\delta\theta,\mathbf{x}})$. The matrices $K_\theta(\mathbf{x}, \mathbf{x}) = \left(\nabla_\theta f_\theta^c(\mathbf{x}) \cdot \nabla_\theta f_\theta^{c'}(\mathbf{x}) \right)_{c,c'}$ and $F_{\theta,\mathbf{x}} = \mathbb{E}_c \left[\nabla_\theta f_\theta^c(\mathbf{x}) \, \nabla_\theta f_\theta^c(\mathbf{x})^T \right]$ are indeed to each other what correlation is to covariance. Thus the quantities defined in Equation (5) already take into account information geometry when applied to the pre-softmax layer, and do not need supplementary metric adjustment.

Faster setup for classification tasks with many classes In a classification task in d classes with large d, the computation of $d \times d$ matrices may be prohibitive. As a workaround, for a given input training sample $\mathbf{x}$, the classification task can be seen as a binary one (the right label c_R *vs.* the other ones), in which case the d outputs of the neural network can be accordingly combined in a single real value. The 1D similarity measure can then be used to compare any training samples of the same class.

When making statistics on similarity values $\mathbb{E}_{\mathbf{x}'} \left[k_\theta^C(\mathbf{x}, \mathbf{x}') \right]$, another possible task binarization approach is to sample an adversary class c_A along with $\mathbf{x}'$, and hence consider $\nabla_\theta f_\theta^{c_R}(\mathbf{x}) - \nabla_\theta f_\theta^{c_A}(\mathbf{x})$. Both approaches will lead to similar results in Section 5.

4 Estimating density

In this section, we use similarity to estimate input neighborhoods and perform statistics on them.

4.1 Estimating the number of neighbors

Given a point $\mathbf{x}$, how many samples $\mathbf{x}'$ are similar to $\mathbf{x}$ according to the network? This can be measured by computing $k_\theta^C(\mathbf{x}, \mathbf{x}')$ for all $\mathbf{x}'$ and picking the closest ones, *i.e. e.g.* the $\mathbf{x}'$ such that $k_\theta^C(\mathbf{x}, \mathbf{x}') \geqslant 0.9$. More generally, for any data point $\mathbf{x}$, the histogram of the similarity $k_\theta^C(\mathbf{x}, \mathbf{x}')$ over all $\mathbf{x}'$ in the dataset (or a representative subset thereof) can be drawn, and turned into an estimate of the number of neighbors of $\mathbf{x}$. To do this, several types of estimates are possible:

- hard-thresholding, for a given threshold $\tau \in [0, 1]$: $N_\tau(\mathbf{x}) = \sum_{\mathbf{x}'} \mathbb{1}_{k_\theta^C(\mathbf{x},\mathbf{x}') \geqslant \tau}$
- soft estimate: $N_S(\mathbf{x}) \;=\; \sum_{\mathbf{x}'} k_\theta^C(\mathbf{x}, \mathbf{x}')$
- less-soft positive-only estimate ($\alpha > 0$): $N_\alpha^+(\mathbf{x}) \;=\; \sum_{\mathbf{x}'} \mathbb{1}_{k_\theta^C(\mathbf{x},\mathbf{x}') > 0} \, k_\theta^C(\mathbf{x}, \mathbf{x}')^\alpha$

In practice we observe that k_θ^C is very rarely negative, and thus the soft estimate N_S can be justified as an average of the hard-thresholding estimate N_τ over all possible thresholds τ:

$$\int_{\tau=0}^{1} N_\tau(\mathbf{x}) d\tau \;=\; \sum_{\mathbf{x}'} \int_{\tau=0}^{1} \mathbb{1}_{k_\theta^C(\mathbf{x},\mathbf{x}') \geqslant \tau} \, d\tau \;=\; \sum_{\mathbf{x}'} k_\theta^C(\mathbf{x},\mathbf{x}') \, \mathbb{1}_{k_\theta^C(\mathbf{x},\mathbf{x}') \geqslant 0} \;=\; N_1^+(\mathbf{x}) \;\simeq\; N_S(\mathbf{x})$$

4.2 Low complexity of the soft estimate $N_S(\mathbf{x})$

The soft estimate $N_S(\mathbf{x})$ is rewritable as:

$$\sum_{\mathbf{x}'} k_\theta^C(\mathbf{x},\mathbf{x}') = \sum_{\mathbf{x}'} \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} \cdot \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|} = \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} \cdot \mathbf{g} \quad \text{with} \quad \mathbf{g} = \sum_{\mathbf{x}'} \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|}$$

and consequently $N_S(\mathbf{x})$ can be computed jointly for all $\mathbf{x}$ in linear time $O(|\mathcal{D}|p)$ in the dataset size $|\mathcal{D}|$ and in the number of parameters p, in just two passes over the dataset, when the output dimension is 1. For higher output dimensions d, a similar trick can be used and the complexity becomes $O(|\mathcal{D}|d^2p)$. For classification tasks with a large number d of classes, the complexity can be reduced to $O(|\mathcal{D}|p)$ through an approximation consisting in binarizing the task (*c.f.* end of Section 3).

4.3 Test of the various estimators

In order to rapidly test the behavior of all possible estimators, we applied them to a toy problem where the network's goal is to predict a sinusoid. To change the difficulty of the problem, we vary its frequency, while keeping the number of samples constant. More details and results of the toy problem are in the suppl. materials. Fig.4 shows for each estimator (with different parameters when relevant), the result of their neighbor count estimation. When the frequency f of the sinusoid to predict increases, the number of neighbors decreases in $\frac{1}{f}$ for every estimator. This aligns with our intuition that as the problem gets harder, the network needs to distinguish input samples more to achieve a good performance, thus the amount of neighbors is lower. In particular we observe that the proposed $N_S(\mathbf{x})$ estimator behaves well, thus we will use that one in bigger studies requiring an efficient estimator.

4.4 Further potential uses for fitness estimation

When the number of neighbors of a training point $\mathbf{x}$ is very low, the network is able to set any label to $\mathbf{x}$, as this won't interfere with other points, by definition of our similarity criterion $k_\theta(\mathbf{x},\mathbf{x}')$. This is thus a typical overfit case, where the network can learn by heart a label associated to a particular, isolated point.

On the opposite, when the set of neighbors of $\mathbf{x}$ is a too big fraction of the dataset, comprising varied elements, by definition of $k_\theta(\mathbf{x},\mathbf{x}')$ the network is not able to distinguish them, and consequently it can only provide a common output for all of them. Therefore it might not be able to express variety enough, which would be a typical underfit case.

The quality of fit can thus be observed by monitoring the number of neighbors together with the variance of the desired labels in the neighborhoods (to distinguish underfit from just high density).

Prediction uncertainty A measure of the uncertainty of a prediction $f_\theta(\mathbf{x})$ could be to check how easy it would have been to obtain another value during training, without disturbing the training of other points. A given change $\mathbf{v}$ of $f_\theta(\mathbf{x})$ induces changes $\frac{k_\theta^I(\mathbf{x},\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2}\mathbf{v}$ over other points $\mathbf{x}'$ of the dataset, creating a total L^1 disturbance $\sum_{\mathbf{x}'} \| \frac{k_\theta^I(\mathbf{x},\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2}\mathbf{v}\|$. The uncertainty factor would then be the norm of $\mathbf{v}$ affordable within a disturbance level, and quickly approximable as $\frac{\|\nabla_\theta f_\theta(\mathbf{x})\|^2}{\sum_{\mathbf{x}'} k_\theta^I(\mathbf{x},\mathbf{x}')}$.

5 Enforcing similarity

The similarity criterion we defined could be used not only to estimate how similar two samples are perceived, after training, but also to incite the network, during training, to evolve in order to consider these samples as similar.

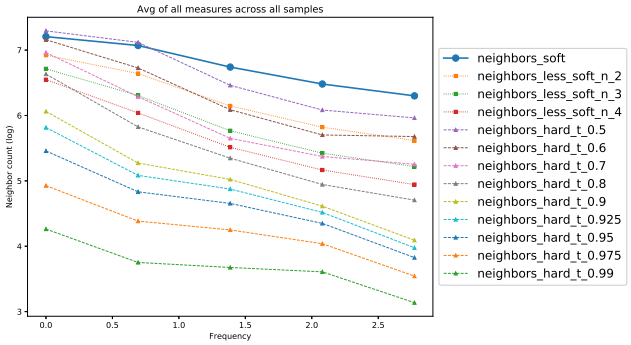

Figure 4: Density estimation using the various approaches (log scale). All approaches behave similarly and show good results.

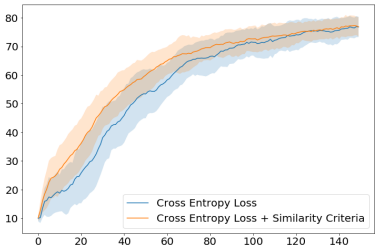

Figure 5: Validation accuracy of a neural network trained on MNIST with and without the similarity criterion (note that the x-axis is the number of minibatches presented to the network, not of epochs).

Asking two samples to be treated as similar If two inputs $\mathbf{x}$ and $\mathbf{x}'$ are known to be similar (from a human point of view), one can enforce their similarity from the network perspective, by adding to the loss the term:

$$-k_\theta^C(\mathbf{x}, \mathbf{x}') \ .$$

Asking a distribution of samples to be treated as similar By extension, to enforce the similarity of a subset $\mathcal{S}$ of training samples, of size $n = |\mathcal{S}|$, one might consider the average pairwise similarity k_θ^C over all pairs, or the standard deviation of the gradients. Both turn out to be equivalent to maximizing the norm of the gradient mean $\mu = \frac{1}{n}\sum_{i\in\mathcal{S}} \frac{\nabla_\theta f_\theta(\mathbf{x}_i)}{\|\nabla_\theta f_\theta(\mathbf{x}_i)\|}$:

$$\frac{1}{n(n-1)} \sum_{i,j\in\mathcal{S}, i\neq j} k_\theta^C(x_i, x_j) \ = \ \frac{n}{n-1}\|\mu\|^2 - \frac{1}{n-1} \qquad \text{and} \qquad \operatorname*{var}_{i\in\S} \frac{\nabla_\theta f_\theta(\mathbf{x}_i)}{\|\nabla_\theta f_\theta(\mathbf{x}_i)\|} = 1 - \|\mu\|^2 \, .$$

In practice, common deep learning platforms are much faster when using mini-batches, but then return only the gradient sum $\sum_{i\in\mathcal{B}} \nabla_\theta f_\theta(\mathbf{x}_i)$ over a mini-batch $\mathcal{B}$, not individual gradients, preventing the normalization of each of them to compute k_θ^C or μ. So instead we compare means of un-normalized gradients, over two mini-batches $\mathcal{B}_1$ and $\mathcal{B}_2$ comprising each n_B samples from $\mathcal{S}$, which yields the criterion:

$$n_B \, \frac{\|\mu_1 - \mu_2\|^2}{\|\mu_1\|\|\mu_2\|} \qquad \text{where} \quad \mu_k = \frac{1}{n} \sum_{i\in\mathcal{B}_k} \nabla_\theta f_\theta(\mathbf{x}_i) \, .$$

The factor n_B counterbalances the $\frac{1}{\sqrt{n_B}}$ variance reduction effect due to averaging over n_B samples.

Group invariance The distributions of samples asked to be seen as similar could be group orbits [3]. A differential formulation of group invariance enforcement is also proposed in the suppl. materials.

Complexity The *double-backpropagation* routine, available on common deep learning platforms, allows the optimization of such criteria [4, 8, 18, 7], roughly doubling the computational time of a gradient step.

Dynamics of learning Our approach enforces similarity not just at the output level, but within the whole internal computational process. Therefore, during training, information is provided directly to each parameter instead of being back-propagated through possibly many layers. Thus the dynamics of learning are expected to be different, especially for deep networks.

To test this hypothesis, we train a small network on MNIST with and without the similarity criteria acting as an auxiliary loss (see Fig. 5). As a result, we observe an acceleration of the convergence very early in the learning process. It is worth noting that this effect can be observed across a wide range of different neural architectures. We performed additional experiments on toy datasets as well as on CIFAR10 with no or only negligible improvements. All together this suggests that using the similarity criteria during training may be beneficial to specific datasets as opposed to specific architectures, and indeed, as the class intra-variability in CIFAR10 is known to be high, considering all examples of a class of CIFAR10 as similar is less relevant.

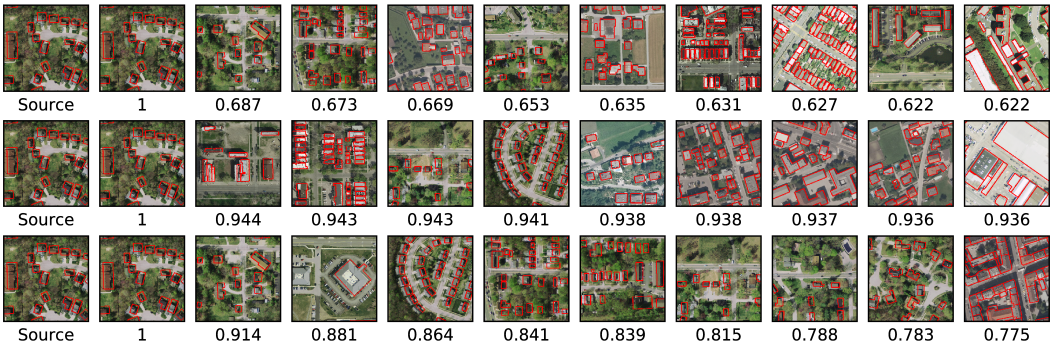

Figure 6: Example of nearest neighbors for a patch. Each line corresponds to a round. Each patch has its similarity written under it.

(a) Round 1 (b) Round 2 (c) Round 3

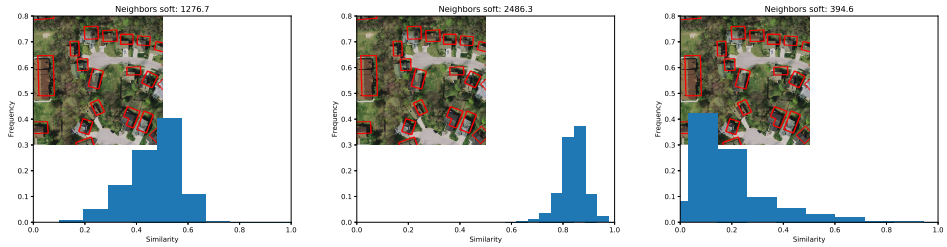

Figure 7: Histograms of similarities for one patch across rounds.

6 Application to the dataset self-denoising experiment

We studied the multi-round training scheme of [1] by applying our similarity measure to a sampling of input patches of the training dataset for one network per round. The principle of the multi-round training scheme is to reduce the noise of the annotations, obtaining aligned annotations in the end. More details in the supplementary materials. For a certain input patch, we computed its similarity with all the other patches for the 3 networks. With those similarities we can compute the nearest neighbors of that patch, see Fig.6. The input patch is of a suburb area with sparse houses and individual trees. The closest neighbors look similar as they usually feature the same types of buildings, building arrangement and vegetation. However sometimes the network sees a patch as similar when it is not clear from our point of view (for example patches with large buildings). For more in-depth results, we computed the histogram of similarities, see Fig7. We observe that round 2 is quite different in that the patch is closer to all other patches than in other rounds. We observe the same behavior in 19 other input patches (see supplementary materials). By round 3, the multiple-rounds method has already converged (see Fig.2, the grey curve is on top of the green one), this is somehow reflected in the neighbor count for round 3 but we do not have an explanation for this.

7 Conclusion

We defined a proper notion of input similarity as perceived by the neural network, based on the ability of the network to distinguish the inputs. This brings a new tool to analyze trained networks, in plus of visualization tools such as grad-CAM [19]. We showed how to turn it into a density estimator, which was validated on a controlled experiment, and usable to perform fast statistics on large datasets. It opens the door to underfit/overfit/uncertainty analyses or even control during training, as it is differentiable and computable at low cost. We also showed that any desired similarity could be enforced during training, at reasonable cost, and noticed a dataset-dependent boosting effect that should be further studied along with robustness to adversarial attacks, as the training differs significantly from usual methods. The code is available at `https://github.com/netsimilarity` .

References

[1] Anonymous. Noisy supervision for correcting misaligned cadaster maps without perfect ground truth data. In *Not Yet Published*, 2019.

[2] Lenaic Chizat and Francis Bach. A note on lazy training in supervised differentiable programming. *arXiv preprint arXiv:1812.07956*, 2018.

[3] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999, 2016.

[4] Harris Drucker and Yann Le Cun. Double backpropagation increasing generalization performance. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 2, pages 145–150. IEEE, 1991.

[5] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in neural information processing systems*, pages 262–270, 2015.

[6] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[7] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Advances in neural information processing systems*, pages 529–536, 1995.

[9] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[10] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[11] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. In *International Conference on Machine Learning*, pages 2971–2980, 2018.

[12] Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo, and Li-Jia Li. Learning from noisy labels with distillation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1910–1918, 2017.

[13] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Can semantic labeling methods generalize to any city? the Inria aerial image labeling benchmark. In *IGARSS*, 2017.

[14] Volodymyr Mnih and Geoffrey E Hinton. Learning to label aerial images from noisy data. In *Proceedings of the 29th International conference on machine learning (ICML-12)*, pages 567–574, 2012.

[15] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Learning with noisy labels. In *Advances in neural information processing systems*, pages 1196–1204, 2013.

[16] Yann Ollivier. Riemannian metrics for neural networks I: feedforward networks. *Information and Inference: A Journal of the IMA*, 4(2):108–153, 03 2015. ISSN 2049-8772. doi: 10.1093/imaiai/iav006. URL `https://doi.org/10.1093/imaiai/iav006`.

[17] Yann Ollivier. Riemannian metrics for neural networks II: recurrent networks and learning symbolic data sequences. *Information and Inference: A Journal of the IMA*, 4(2):154–193, 03 2015. ISSN 2049-8772. doi: 10.1093/imaiai/iav007. URL `https://doi.org/10.1093/imaiai/iav007`.

[18] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 833–840. Omnipress, 2011.

[19] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

[20] Sainbayar Sukhbaatar, Joan Bruna, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. Training convolutional networks with noisy labels. *arXiv preprint arXiv:1406.2080*, 2014.