Riemannian metrics for neural networks

Yann Ollivier

August 27, 2013

Abstract

We describe four algorithms for neural network training, each adapted to different scalability constraints. These algorithms are mathematically principled and invariant under a number of transformations in data and network representation, from which performance is thus independent. These algorithms are obtained from the setting of differential geometry, and are based on either the natural gradient using the Fisher information matrix, or on Hessian methods, scaled down in a specific way to allow for scalability while keeping some of their key mathematical properties.

The most standard way to train neural networks, backpropagation, has several known shortcomings. Convergence can be quite slow. Backpropagation is sensitive to data representation: for instance, even such a simple operation as exchanging 0's and 1's on the input layer will affect performance (Figure 1), because this amounts to changing the parameters (weights and biases) in a non-trivial way, resulting in different gradient directions in parameter space, and better performance with 1's than with 0's. (In the related context of restriced Boltzmann machines, it has been found that the standard training technique by gradient ascent favors setting hidden units to 1, for very much the same reason [AAHO11, Section 5].) This specific phenomenon disappears if, instead of the logistic function, the hyperbolic tangent is used as the activation function. Scaling also has an effect on performance: for instance, a common recommendation [LBOM96] is to use $1.7159 \tanh(2x/3)$ instead of just $\tanh(x)$ as the activation function.

It would be interesting to have algorithms whose performance is insensitive to particular choices such as scaling factors in network construction, parameter encoding or data representation. Such invariance properties mean more robustness for an algorithm: good performance on a particular problem presumably indicates good performance over a whole class of problems equivalent to the first one by simple (e.g., affine) transformations.

Ways exist to deal with these issues, such as Hessian methods or the natural gradient, which are invariant (and thus preserve performance) over a wide class of changes in the representation of the data and of the network. However, these are generally not scalable (the cost of maintaining the whole

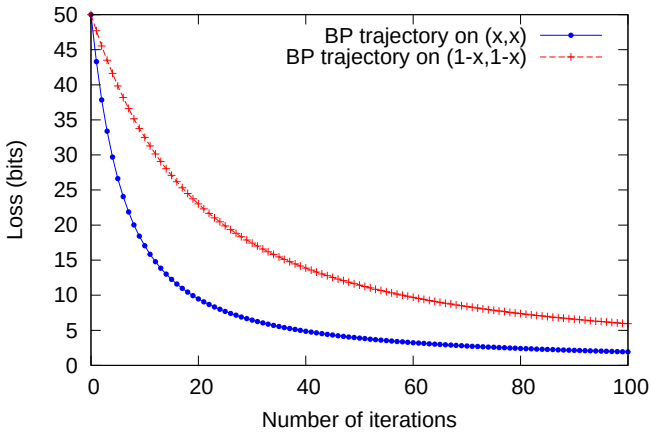

Figure 1: Backpropagation learns better with 1's. A neural network with two layers of size 50 (no hidden layer) is trained to reproduce its input. A random binary sequence x of length 50 with 75% of 1's is generated. The network is trained on the input-output pair (x, x) for 100 backpropagation steps (learning rate 0.01). The experiment is repeated on the input-output pair $(1 - x, 1 - x)$. In both cases all initial weights are set to 0.

Hessian or Fisher information matrix is prohibitive for large networks). The approximations made to ensure their scalability, such as keeping only diagonal terms or making small-rank approximations, break these invariance properties.

We present four algorithms which share a common inspiration from Riemannian metrics and are adapted to four different scalability constraints. The most lightweight has the same complexity as backpropagation, up to a constant factor. The heaviest requires that the network is sparsely connected (the average number of units influencing a given unit must not be too large) and that the output layer is not too large. This latter condition is typically fulfilled in classification tasks.

The *unitwise natural gradient* is a scaled-down version of Amari's natural gradient [Ama98] in which the blocks of incoming parameters to each unit are treated independently, thus dealing, at each unit, with a square matrix indexed by the incoming parameters to this unit. This method has been proposed as far back as [Kur94] to train neural networks; however the presentation in [Kur94] is limited to networks with only one hidden layer, because it relies on an explicit symbolic computation of entries of the Fisher matrix. Here Proposition 26 allows for an efficient computation of the *exact* Fisher information matrix in arbitrary neural networks, by doing n_{out} distinct backpropagations for each sample in the dataset. As a result, the unitwise natural gradient is adapted to situations where both the connectiv-

ity of the network and the output layer are reasonably small.

The *backpropagated metric gradient* is not altogether new either: it can be described as a blockwise quasi-Hessian method in which several approximations (Gauss–Newton and neglecting cross-unit terms) are used. However, we describe it in an intrinsic way: it stems from a well-defined *backpropagated metric* on parameter space, in which no approximations are involved. Invariance properties follow immediately from this viewpoint. It is adapted to networks with reasonably small connectivity but output layers of arbitrary size.

The *quasi-diagonal natural gradient* and *quasi-diagonal backpropagated metric gradient* apply a quasi-diagonal reduction to these two algorithms, which removes the quadratic dependency on connectivity at each unit. This is done in a specific way to keep some (but not all) of the invariance properties, such as insensitivity to using sigmoid or $1.7159 \tanh(2x/3)$. The quasi-diagonal natural gradient still requires that the output layer is not too large, whereas the quasi-diagonal backpropagated metric gradient has the same complexity as ordinary backpropagation up to a constant factor. The quasi-diagonal natural gradient and quasi-diagonal backpropagated metric gradient have not been described before, to the best of our knowledge.

Backpropagation is the simple gradient descent over parameter space. Gradient descents follow the steepest direction of change in parameter space, and implicitly rely on a norm (or quadratic form, or metric) to define the steepest direction: the gradient step $x \leftarrow x - \eta \nabla f$ can actually be rewritten (for small enough η, up to $O(\eta^2)$ and for regular enough functions f) as

$$x \leftarrow \arg\min_y \left\{ f(y) + \frac{1}{2\eta} \|y - x\|^2 \right\} \tag{1}$$

namely, the gradient descent moves into the direction yielding the smallest values of f, penalized by the distance from the current point, measured by the square norm $\|y - x\|^2$. For backpropagation this norm $\|\cdot\|^2$ is the numerical change in the values of the parameters: backpropagation provides the direction of largest improvement for a minimal change in these numerical values. Hence simple changes in parametrization influence the behavior of the algorithm. On the other hand, norms $\|\cdot\|^2$ based on what the network does, rather than how it is represented as numbers, will lead to intrinsic algorithms. This is one of the ideas behind Amari's natural gradient.

In Section 2 we build several invariant norms, by placing neural networks in the context of differential manifolds and Riemannian geometry. The gradient descent coming from an invariant norm (Riemannian metric) will itself be invariant. Moreover, any gradient descent using any norm has the property that small enough learning rates ensure performance improvement at each step.

The resulting algorithms are all invariant under a number of transformations, including affine reparametrization of the unit activities. Among the invariance properties enjoyed by the unitwise natural gradient and the backpropagated metric (but not their quasi-diagonal reductions) are linear recombinations of the input received by a given unit in the network, so that a unit receiving signals f and $f+\varepsilon g$ (as functions over the dataset) will learn an output correlated to g just as fast as a unit receiving signals f and g (on the input layer this can be accounted for by normalizing and de-correlating the inputs, but this could occur at internal units as well). Thus these gradients have a "best-fit" interpretation (Section 3.3): at each unit they solve a least-square problem of matching input signals and desired backpropagated output, an interpretation proposed in [Kur94].

The quasi-diagonal reductions of these algorithms are based on the observation that there is a distinction between weeights w_{ik} and w_{jk} coming to k from different units, but no intrinsic mathematical separation between weights and biases. Intuitively, given that unit k receives a signal $w_{ik}a_i$ from unit i, if we change w_{ik} to $w_{ik}+\delta w_{ik}$, the average signal to unit k will change by $\delta w_{ik}\bar{a}_i$ where $\bar{a}_i$ is the average activation of i. Hence it might be a good idea to automatically add $-\delta w_{ik}\bar{a}_i$ to the bias of k, to compensate. The quasi-diagonal algorithms we present are more sophisticated versions of this, tuned for invariance and using weighted averages. The few added terms in the update sometimes greatly improve performance (Fig 5 on page 46).

None of these algorithms is second-order: they are based on (Riemannian) metrics evaluating the magnitude of the effect on the output of changes in a given direction, thus providing a suitable learning rate for each direction. They emulate second-order effects in the same way the Gauss–Newton algorithm emulates the Newton method[1].

Even though the focus of this article is mainly the mathematics of neural network training, we tested experimentally the impact of using the various methods. We selected a very simple auto-encoding problem on which we expected any training method would perform well. A sparsely connected network with 5 layers of size 100, 30, 10, 30, and 100 was built, and 16 random length-100 binary strings were fed to the input layer, with the target equal to the input. Ideally the network learns to encode each of the 16 samples using 4 bits on the middle layer (thus with room to spare) and rewrites the output from this. The details are given in Section 4.

Even then, backpropagation performs poorly: after $10,000$ batch passes the average log-loss is about 36 bits per sample (out of 100) for sigmoid backpropagation, and about 25 bits per sample for tanh backpropagation. Note that 30 bits per sample would correspond to a method which learns

[1] Actually, in the framework of differential geometry, the Hessian is intrinsically defined only at local optima of the function, so one could say that in such a setting the Newton method approximates the Gauss–Newton algorithm rather than the other way around.

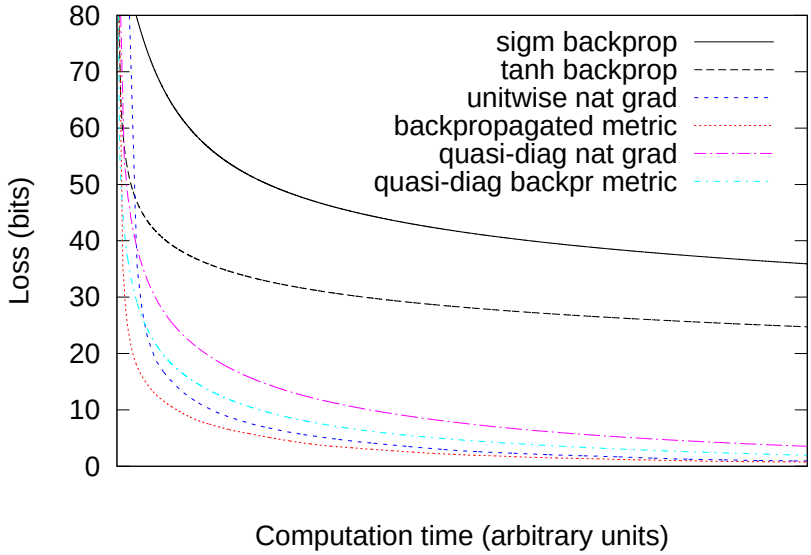

Figure 2: Auto-encoding using a 100–30–10–30–100 deep sparsely connected network. Comparison of backpropagation using sigmoid and tanh activation, and the four algorithms described in Section 1, for a given computation time budget.

only the parameters of the output layer and can reproduce the output if someone fills the last hidden layer with the correct 30 bits.

For the *same* total computation time equivalent to 10,000 batch backpropagations[2], the quasi-diagonal algorithms have a log loss of about 1.5 to 3.5 bits per sample, and both the unitwise natural gradient and the backpropagated metric gradient have a log loss of 0.3 to 1.5 bit per sample, thus essentially solving the problem. See Figure 2.

The impact of the few non-diagonal terms in the quasi-diagonal algorithms was tested by removing them. In this case the quasi-diagonal backpropagated metric gradient reduces to the diagonal Gauss–Newton method [LBOM96, Section 7.4]. Of course, this breaks the invariance properties, thus the impact is different for sigmoid or tanh implementations. The diagonal Gauss–Newton method in sigmoid implementation was found to perform more poorly, with a final log-loss of about 12 bits per sample (Figure 5 on page 46), while in tanh implementation it comes somewhat close to the quasi-diagonal algorithms at about 3.5 bits per sample (presumably because in our problem, the activity of all units, not only input units, stay perfectly centered during training). Thus the quasi-diagonal backpropagated metric

[2] as measured in CPU time on a personal computer, but this can depend a lot on implementation details

gradient can be seen as "the invariant way" to write the diagonal Gauss–Newton method, while performance of the latter is not at all invariant.

We also compared the exact unitwise natural gradient obtained thanks to Proposition 26, to a variant of the natural gradient in which only the gradient terms $bb^\top$ corresponding to the target for each sample are added to the Fisher matrix ([APF00, RMB07] and Section 2.5 below). The latter, when implemented unitwise, performs rather poorly on this auto-encoding task, with a log loss of about 25 to 28 bits per sample. The reason, discussed in Section 4, may be that quality of this approximation to the Fisher matrix strongly depends on output dimensionality.

One lesson from the numerical experiments is that the regularization term $\varepsilon \operatorname{Id}$ added to the matrices, needed to prevent bad behavior upon inversion, formally breaks the invariance properties: individual trajectories in sigmoid or tanh implementations, initialized in the same way, start to differ after a dozen iterations. Still, overall performance is not affected and is the same in both implementations (Figure 4, p. 44).

Though the quasi-diagonal methods perform well, the only methods to sometimes reach very small values of the loss function on this example (less than 0.1 bit per sample) are the unitwise natural gradient and the backpropagated metric, which at each unit maintain a full matrix over the incoming parameters and thus achieve invariance under affine recombination of incoming signals. These two methods are relevant only when network connectivity is not too high. This highlights the interest of sparsely connected networks from a theoretical viewpoint.

A companion article [Oll13] develops related ideas for *recurrent* neural networks and provides more in-depth experiments on complex symbolic data sequences.

Acknowledgments. I would like to thank Youhei Akimoto, Jérémy Bensadon, Cécile Germain, Balázs Kégl and Michèle Sebag for helpful conversations and their feedback on this article.

Notation for neural networks

Consider a directed neural network model: a set $\mathcal{L}$ of units together with a set of directed edges $i \to j$ for $i, j \in \mathcal{L}$, without cycle. Let $\mathcal{L}_{\text{out}}$ be the output units, that is, the units with no outgoing edges, and similarly let $\mathcal{L}_{\text{in}}$ be the set of units without incoming edges. We use the logisic activation function: given an activation level for the input units, each unit j gets an activation level

$$a_j = \operatorname{sigm}\Big(\sum_{i \to j} w_{ij} a_i\Big) = \frac{\mathrm{e}^{\sum_{i \to j} w_{ij} a_i}}{1 + \mathrm{e}^{\sum_{i \to j} w_{ij} a_i}} = \frac{1}{1 + \mathrm{e}^{-\sum_{i \to j} w_{ij} a_i}} \qquad (2)$$

depending on the activation levels of the units i pointing to j and on the firing coefficients w_{ij} from[3] i to j. Biases are treated as the weights w_{0j} from a special always-activated unit 0 ($a_0 \equiv 1$). For a given non-input unit j, we call the parameters w_{0j} and w_{ij} for $i \to j$ the set of *incoming parameters* to unit j. We refer to [RN03], which we mostly follow with minor changes in notation.

The dataset for this network is a set $\mathcal{D}$ of inputs, where each input $x \in \mathbb{R}^{\mathcal{L}_{\text{in}}}$ is a real-valued vector over the input layer. For each input is given a target y in an arbitrary space. We view the network as a probabilistic generative model: given an input $a_i = x_i$ on the input layer $\mathcal{L}_{\text{in}}$, we assume that the activations of the output layer are interpreted in a fixed way as a probability distribution $\omega(y)$ over the target space. The goal is to maximize the probability to output y on input x: we define the loss function

$$\ell(\omega, y) := -\ln \omega(y) \tag{3}$$

the sum of which over the dataset is to be minimized. For instance, interpreting the output layer activities $(a_k)_{k \in \mathcal{L}_{\text{out}}}$ as Gaussian variables with mean a_k and variance 1 leads to a quadratic loss function ℓ.

We will use three possible interpretations of the activities of the output layer as probability distributions.

Definition 1 (Bernoulli, square-loss, and classification interpretations).
The Bernoulli interpretation *of the output layer is a Bernoulli distribution as follows: given the activations $(a_k)_{k \in \mathcal{L}_{\text{out}}}$ of the output layer, the final output is a $\{0,1\}^{\mathcal{L}_{\text{out}}}$-valued random variable $Y = (Y_k)_{k \in \mathcal{L}_{\text{out}}}$ of independent Bernoulli variables, where the activations a_k give the probability to have $Y_k = 1$, namely $\Pr(Y_k = 1) = a_k$.*

The square-loss interpretation *of the output layer sends the activations $(a_k)_{k \in \mathcal{L}_{\text{out}}}$ of the output layer to a random variable $Y = (Y_k)_{k \in \mathcal{L}_{\text{out}}}$ of independent Gaussian variables, where each $Y_k \sim \mathcal{N}(a_k, 1)$ is a Gaussian of mean a_k.*

The classification interpretation *of the output layer sends the output activations $(a_k)_{k \in \mathcal{L}_{\text{out}}}$ to a probability distribution over the set indexing the output layer, where the probability of class $k \in \mathcal{L}_{\text{out}}$ is $a_k^2 / (\sum_{k' \in \mathcal{L}_{\text{out}}} a_{k'}^2)$.*

(Our choice for the classification interpretation is motivated by two arguments, theoretical and practical. First, the set of probability distributions over a finite set, equipped with its Fisher metric, is isometric to the positive quadrant in a sphere and so is naturally parametrized by numbers a_k with $\sum a_k^2 = 1$, and these variables yield a simple expression for the Fisher matrix. Second, taking squares gives a boost to the most activated output unit, in a

[3] What is w_{ij} for some authors is w_{ji} for others. Our convention is the same as [RN03] but for instance [LBOM96] follows the opposite convention.

smooth way—in a deterministic context one would define the most activated output unit as the class of the sample).

A common way to train the network on a given target value y is by backpropagation. Define the backpropagated values b_i (for a given loss function) at each unit i by induction from the output layer:

$$\begin{cases} b_i := -\frac{\partial \ell}{\partial a_i} & \text{for } i \text{ in the output layer } \mathcal{L}_{\text{out}} \\ \quad = \begin{cases} \frac{y_i - a_i}{a_i(1-a_i)} & \text{(Bernoulli)} \\ y_i - a_i & \text{(square-loss)} \\ \frac{2y_i}{a_i} - \frac{2a_i}{\sum_{k \in \mathcal{L}_{\text{out}}} a_k^2} & \text{(classification)} \end{cases} \\ b_i := \sum_{j,\, i \to j} w_{ij} a_j (1 - a_j) b_j & \text{for } i \notin \mathcal{L}_{\text{out}} \end{cases} \tag{4}$$

after having, of course, computed the activation levels a_i by forward propagation. Then b_i indicates how the loss will react to a given change in the activation of unit i:

$$b_i = -\frac{\partial \ell}{\partial a_i} \tag{5}$$

for all units i (not only the output layer), so that b_i indicates how we should modify the activities to decrease ℓ. Moreover we have

$$\frac{\partial \ell}{\partial w_{ij}} = -a_i a_j (1 - a_j) b_j \tag{6}$$

for each edge (ij) in the network. (This includes the bias w_{0j} using $a_0 \equiv 1$.) So backpropagation computes the gradient of the loss with respect to the parameters.

It is sometimes more convenient to work with the reduced variables $\tilde{b}_i := a_i(1 - a_i) b_i$ which satisfy

$$\begin{cases} \tilde{b}_i = y_i - a_i & \text{for } i \text{ in the output layer } \mathcal{L}_{\text{out}} \\ \tilde{b}_i = a_i(1 - a_i) \sum_{j,\, i \to j} w_{ij} \tilde{b}_j & \text{for } i \notin \mathcal{L}_{\text{out}} \end{cases} \tag{7}$$

and then one has

$$\frac{\partial \ell}{\partial w_{ij}} = -a_i \tilde{b}_j \tag{8}$$

The gradient descent with learning rate $\eta > 0$ is then defined as the following update on the firing coefficients:

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial \ell}{\partial w_{ij}} = w_{ij} + \eta\, a_i a_j (1 - a_j) b_j = w_{ij} + \eta\, a_i \tilde{b}_j \tag{9}$$

1 Four invariant gradient algorithms

We now describe four gradient algorithms for network training: the *unitwise natural gradient*, the *quasi-diagonal natural gradient*, the *backpropagated metric gradient*, and the *quasi-diagonal backpropagated metric gradient*. Each of these algorithms is adapted to a different scalability constraint. The unitwise natural gradient requires low connectivity and a small output layer; the quasi-diagonal natural gradient requires a small output layer; the backpropagated metric gradient requires low connectivity; the quasi-diagonal backpropagated metric gradient has the same asymptotic complexity as backpropagation.

These algorithms are the implementation, using sigmoid activation, of the more general versions described in Section 2. As they are designed for invariance properties, implementing them using tanh activation would result in the same output, learning trajectories, and performance, so we simply present one implementation.

We first present the algorithms in a batch version. It is straightforward to adapt them to use random mini-batches from the dataset. In Section 1.3 they are also adapted to an online setting: this can be done using standard techniques because the main quantities involved take the form of averages over the dataset, which can be updated online.

1.1 Unitwise natural gradient and quasi-diagonal natural gradient

The unitwise natural gradient has been proposed as far back as [Kur94] to train neural networks; however the presentation in [Kur94] is limited to networks with only one hidden layer, because it relies on an explicit symbolic computation of entries of the Fisher matrix. Proposition 26 below allows for an efficient computation of the *exact* Fisher information matrix by doing n_{out} distinct backpropagations for each sample in the dataset. This relies on linearity of backpropagation, as follows.

Definition 2 (Backpropagation transfer rates).
Fix an input x for the network and compute the activities by forward propagation. Let k be a unit in the network and k_{out} be a unit in the output layer. Define the backpropagation transfer rates $J_k^{k_{\text{out}}}$ *from k_{out} to k by backpropagating the value 1 at k_{out}. Formally:*

$$\begin{cases} J_{k_{\text{out}}}^{k_{\text{out}}} := 1 \quad , \qquad J_k^{k_{\text{out}}} := 0, & \text{for } k \neq k_{\text{out}} \text{ in the output layer } \mathcal{L}_{\text{out}} \\ J_k^{k_{\text{out}}} := \sum_{j,\, k \to j} w_{kj} a_j (1 - a_j) J_j^{k_{\text{out}}} & \text{for non-output units } k \end{cases} \tag{10}$$

These transfer rates have the property that if backpropagation values b are set on the output layer, then $b_k = \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} J_k^{k_{\text{out}}} b_{k_{\text{out}}}$ for any unit k (see also [LBOM96, Section 7.2]).

Computation of the transfer rates can be done by n_{out} backpropagations. There are further simplifications, since the transfer rates for k in the input layer are never used (as there are no incoming parameters), and the transfer rates on the last hidden layer are readily computed as $J_k^{k_{\text{out}}} = w_{kk_{\text{out}}} a_{k_{\text{out}}}(1 - a_{k_{\text{out}}})$. Thus it is enough to backpropagate the transfer rates from the last hidden layer to the first hidden layer (and so with only one hidden layer, the case considered in [Kur94] for the Fisher matrix, no backpropagation is needed).

Definition 3 (Fisher modulus).
Fix an input x for the network and compute the activities by forward propagation. For each unit k in the network, define the Fisher modulus $\Phi_k(x)$ *of unit k on input x as follows, depending on output layer interpretation.*

- *For the Bernoulli interpretation, set*

$$\Phi_k(x) := \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} \frac{(J_k^{k_{\text{out}}})^2}{a_{k_{\text{out}}}(1 - a_{k_{\text{out}}})} \tag{11}$$

- *For the square-loss interpretation, set*

$$\Phi_k(x) := \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} (J_k^{k_{\text{out}}})^2 \tag{12}$$

- *For the classification interpretation, set*

$$\Phi_k(x) := \frac{4}{S} \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} (J_k^{k_{\text{out}}})^2 - \frac{4}{S^2} \left(\sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} a_{k_{\text{out}}} J_k^{k_{\text{out}}} \right)^2 \tag{13}$$

  where $S := \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} a_{k_{\text{out}}}^2$.

Definition 4 (Unitwise Fisher matrix).
Let k be a unit in the network. Let E_k be the set of incoming units to k (including the always-activated unit 0). The unitwise Fisher matrix *at unit k is the $(\#E_k) \times (\#E_k)$ matrix $F^{(k)}$ defined by*

$$F_{ij}^{(k)} := \mathbb{E}_{x \in \mathcal{D}}\, a_i a_j a_k^2 (1 - a_k)^2 \Phi_k \tag{14}$$

for i and j in E_k (including the unit 0 with $a_0 \equiv 1$). Here $\mathbb{E}_{x \in \mathcal{D}}$ represents the average over samples x in the dataset (all the terms a_i, a_j, a_k, Φ_k depend on the input to the network).

By Proposition 26 below, $F_{ij}^{(k)}$ is the block of the Fisher information matrix associated with the incoming parameters to k, hence the name.

Definition 5 (Unitwise natural gradient).
The unitwise natural gradient *with learning rate $\eta > 0$ updates the parameters of the network as follows.*

For each unit k, define the vector $G^{(k)}$ by

$$G_i^{(k)} := -\mathbb{E}_{x\in\mathcal{D}} \frac{\partial \ell(y)}{\partial w_{ik}} = \mathbb{E}_{x\in\mathcal{D}}\, a_i a_k (1-a_k) b_k \tag{15}$$

where b_k is the backpropagated value at k obtained from the target y associated with x, and where i runs over the incoming units to k (including the always-activated unit $i = 0$). Compute the vector

$$\delta w^{(k)} := (F^{(k)})^{-1} G^{(k)} \tag{16}$$

Then the parameters of the network are updated by

$$w_{ik} \leftarrow w_{ik} + \eta \delta w_i^{(k)} \tag{17}$$

The unitwise natural gradient requires updating a matrix with $(\#E_k)^2$ coefficients for each data sample (and inverting it, but this is done only once per pass over the dataset). This is fine if connectivity is low. We now define a more light-weight version in case connectivity is large. Its computational cost is equivalent to that of ordinary backpropagation provided the output layer is small.

Definition 6 (Quasi-diagonal natural gradient).
The quasi-diagonal natural gradient *with learning rate $\eta > 0$ updates the parameters of the network as follows.*

For each unit k, compute only the entries $F_{00}^{(k)}$, $F_{0i}^{(k)}$, and $F_{ii}^{(k)}$ of the unitwise Fisher matrix at k. Define the vector $G^{(k)}$ as in (15) *above. Define the vector $\delta w^{(k)}$ by*

$$\delta w_i^{(k)} = \frac{G_i^{(k)} F_{00}^{(k)} - G_0^{(k)} F_{0i}^{(k)}}{F_{ii}^{(k)} F_{00}^{(k)} - (F_{0i}^{(k)})^2} \qquad \text{for } i \neq 0 \tag{18}$$

$$\delta w_0^{(k)} = \frac{G_0^{(k)}}{F_{00}^{(k)}} - \sum_{i\neq 0} \frac{F_{0i}^{(k)}}{F_{00}^{(k)}} \delta w_i^{(k)} \tag{19}$$

and update the parameters of the network by

$$w_{ik} \leftarrow w_{ik} + \eta \delta w_i^{(k)} \tag{20}$$

These latter formulas may seem arbitrary. If we remember that (omitting the superscript (k)) $F_{00} = \mathbb{E}_{x\in\mathcal{D}}\, a_k^2 (1-a_k)^2 \Phi_k$, $F_{0i} = \mathbb{E}_{x\in\mathcal{D}}\, a_i a_k^2 (1-a_k)^2 \Phi_k$, and $F_{ii} = \mathbb{E}_{x\in\mathcal{D}}\, a_i^2 a_k^2 (1-a_k)^2 \Phi_k$, we can consider these sums as expectations over the dataset with weights $a_k^2 (1-a_k)^2 \Phi_k$. Then the weighted average of

a_i is $A_i = F_{0i}/F_{00}$ and its weighted variance is $V_i = F_{ii}/F_{00} - A_i^2$ so that we have

$$\delta w_i = \frac{G_i - G_0 A_i}{F_{00} V_i} \tag{21}$$

and in particular the denominator is always positive unless the activity of unit i is constant (in this case, the numerator vanishes too).

A possible interpretation is as follows: If the activity a_i of i is centered over the dataset (with the weights above), then the update is diagonal and the activities are scaled by $1/V_i$. If a_i is not centered, when we change w_{ik} a corresponding term is automatically subtracted from the bias w_{0k} so as not to shift the average activity of unit k, as discussed in the intro.

1.2 Backpropagated metric gradient and quasi-diagonal backpropagated metric gradient

Computing the Fisher matrix as above requires performing n_{out} backpropagations for each sample. If one tries to compute the Fisher modulus Φ directly by backpropagation, the backpropagation equation involves cross-terms between different units. Neglecting these cross-terms results in a simpler version of the Fisher modulus which can be computed in one backward pass; the corresponding backpropagation equation is well-known as an approximation of the Hessian [LBOM96, Section 7]. It turns out this quantity and the associated metric are still intrinsic.

Definition 7 (Backpropagated modulus).
Fix an input x for the network and compute the activities by forward propagation. Define the backpropagated modulus $m_k(x)$ *for each unit k by*

$$m_k(x) := \sum_{j,\, k \to j} w_{kj}^2 a_j^2 (1 - a_j)^2 m_j(x) \tag{22}$$

if k is not an output unit, and, depending on output interpretation,

$$m_k(x) := \begin{cases} \frac{1}{a_k(1-a_k)} & \text{(Bernoulli)} \\ 1 & \text{(square-loss)} \\ \frac{4}{S}(1 - \frac{a_k^2}{S}), \quad S = \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} a_{k_{\text{out}}}^2 & \text{(classification)} \end{cases} \tag{23}$$

for k in the output layer.

Definition 8 (Backpropagated metric).
Let k be a unit in the network. Let E_k be the set of incoming units to k (including the always-activated unit 0). The backpropagated metric *at unit k is the $(\#E_k) \times (\#E_k)$ matrix $M^{(k)}$ defined by*

$$M_{ij}^{(k)} := \mathbb{E}_{x \in \mathcal{D}}\, a_i a_j a_k^2 (1 - a_k)^2 m_k \tag{24}$$

for i and j in E_k (including the unit 0 with $a_0 \equiv 1$). Here $\mathbb{E}_{x\in\mathcal{D}}$ represents the average over samples x in the dataset (all the terms a_i, a_j, a_k, m_k depend on the input to the network).

The backpropagated metric gradient can thus be described as an approximate, blockwise Hessian method in which the Hessian is approximated by the Gauss–Newton technique with, in addition, cross-unit terms neglected. Such a method turns out to be intrinsic.

Definition 9 (Backpropagated metric gradient).
The backpropagated metric gradient *with learning rate $\eta > 0$ updates the parameters of the network as follows.*

For each unit k, define the vector $G^{(k)}$ by

$$G_i^{(k)} := -\mathbb{E}_{x\in\mathcal{D}} \frac{\partial \ell(y)}{\partial w_{ik}} = \mathbb{E}_{x\in\mathcal{D}}\, a_i a_k (1 - a_k) b_k \tag{25}$$

where b_k is the backpropagated value at k obtained from the target y associated with x, and where i runs over the incoming units to k (including the always-activated unit $i = 0$). Compute the vector

$$\delta w^{(k)} := (M^{(k)})^{-1} G^{(k)} \tag{26}$$

Then the parameters of the network are updated by

$$w_{ik} \leftarrow w_{ik} + \eta \delta w_i^{(k)} \tag{27}$$

Definition 10 (Quasi-diagonal backpropagated metric gradient).
The quasi-diagonal backpropagated metric gradient *with learning rate $\eta > 0$ updates the parameters of the network as follows.*

For each unit k, compute only the entries $M_{00}^{(k)}$, $M_{0i}^{(k)}$, and $M_{ii}^{(k)}$ of backpropagated metric at k. Define the vector $G^{(k)}$ as in (15) above. Define the vector $\delta w^{(k)}$ by

$$\delta w_i^{(k)} = \frac{G_i^{(k)} M_{00}^{(k)} - G_0^{(k)} M_{0i}^{(k)}}{M_{ii}^{(k)} M_{00}^{(k)} - (M_{0i}^{(k)})^2} \qquad \text{for } i \neq 0 \tag{28}$$

$$\delta w_0^{(k)} = \frac{G_0^{(k)}}{M_{00}^{(k)}} - \sum_{i\neq 0} \frac{M_{0i}^{(k)}}{M_{00}^{(k)}} \delta w_i^{(k)} \tag{29}$$

and update the parameters of the network by

$$w_{ik} \leftarrow w_{ik} + \eta \delta w_i^{(k)} \tag{30}$$

The same remarks as for the quasi-diagonal natural gradient apply for interpreting the various terms. The denominator $M_{ii}^{(k)} M_{00}^{(k)} - (M_{0i}^{(k)})^2$ can be seen as a weighted variance of the activity of unit i, and is positive unless a_i is constant over the dataset. The contribution of $\delta w_i^{(k)}$ to $\delta w_0^{(k)}$ compensates the change of average activity induced by a change of w_{ik}.

The asymptotic cost of this update is the same as for backpropagation.

If, in the quasi-diagonal backpropagated metric gradient, the non-diagonal terms are neglected ($M_{0i}^{(k)}$ is set to 0), then this reduces to the diagonal Gauss–Newton method equations from [LBOM96, Section 7.4].

Remark 11.
On the incoming parameters to the output layer, the unitwise natural gradient and the backpropagated metric gradient coincide if the Bernoulli or square-loss interpretation is used. (Actually, with learning rate $\eta = 1$ they also both coincide with the Newton method restricted to the output layer parameters.)

Remark 12.
Since these algorithms rely on inverting matrices, regularization is an issue. In practice, terms $\varepsilon \operatorname{Id}$ have to be added to F and M before inversion; terms ε have to be added to the diagonal terms $F_{00}^{(k)}$, $F_{ii}^{(k)}$, $M_{00}^{(k)}$ and $M_{ii}^{(k)}$ in the quasi-diagonal reduction. This formally breaks the invariance properties; Section 3.3 elaborates on this. Still, this operation preserves the guarantee of improvement for small enough learning rates.

1.3 Adaptation to an online setting

The unitwise natural gradient and unitwise backpropagated metric gradient both update the weights by

$$\delta w = A^{-1} G \tag{31}$$

with G the gradient of the loss function over the dataset, and A a positive-definite, symmetric matrix. A key feature here is that the matrix A takes the form of an expectation over the dataset: $F_{ij}^{(k)} = \mathbb{E}_{x \in \mathcal{D}}\, a_i a_j a_k^2 (1 - a_k)^2 \Phi_k$ for the Fisher matrix, and $M_{ij}^{(k)} = \mathbb{E}_{x \in \mathcal{D}}\, a_i a_j a_k^2 (1 - a_k)^2 m_k$ for the backpropagated metric.

Any such algorithm can be turned online using a standard construction as follows (compare e.g. [RMB07]). Another possibility is, of course, to use mini-batches.

In the following, A stands for either the unitwise Fisher matrix or the backpropagated metric. Let $A(x)$ be the corresponding contribution of each input x in the expectation, namely, $A(x)_{ij}^{(k)} = a_i a_j a_k^2 (1 - a_k)^2 \Phi_k$ for the Fisher metric and $A(x)_{ij}^{(k)} = a_i a_j a_k^2 (1 - a_k)^2 m_k$ for the backpropagated metric, so that $A = \mathbb{E}_{x \in \mathcal{D}} A(x)$.

At each step t, we use one new sample in the dataset, update an estimate $A^{(t)}$ of A, and follow a gradient step for this sample, as follows.

- Initialize the matrix $A^{(0)}$ by using a small subsample $\mathcal{D}_{\text{init}} \subset \mathcal{D}$, for instance the first n_{init} samples in the dataset. Namely:

$$A^{(0)} := \mathbb{E}_{x \in \mathcal{D}_{\text{init}}} A(x) \tag{32}$$

- Fix a discount factor $0 < \gamma < 1$. For each new sample x_t, compute its contribution $A(x_t)$ and update A by

$$A^{(t)} := (1-\gamma)A^{(t-1)} + \gamma A(x_t) \tag{33}$$

- Compute the inverse of $A^{(t)}$ from the inverse of $A^{(t-1)}$ by the Sherman–Morrison formula at each unit, using the fact that $A(x_t)$ is a rank-one matrix at each unit. (This way matrix inversion is no more costly than the rest of the step.)

- Compute the negative gradient $G(x_t)$ of the loss function on input x_t by backpropagation.

- Update the parameters by

$$w \leftarrow w + \eta_t (A^{(t)})^{-1} G(x_t) \tag{34}$$

  where η_t is the learning rate.

- For the quasi-diagonal reductions of the algorithms, only the entries A_{00}, A_{ii} and A_{0i} of the matrix A are updated at each step. No matrix inversion is required for the update equations (18)–(19) and (28)–(29).

We could also initialize $A^{(0)}$ to a simple matrix like Id, but this breaks the invariance properties of the algorithms.

The update rule for $A^{(t)}$ depends on the discount factor γ. It should be large enough so that a large number of data points contribute to the computation of A, but small enough to be reactive so that A evolves as training gets along. In our setting, from the particular form of $A(x)$ at a unit k we see that each $A(x_t)$ contributes a rank-one matrix. This means that γ should be much smaller than $1/n_k$ with n_k the number of parameters at unit k, because otherwise the estimated matrix $A^{(t)}$ will be close to a low-rank matrix, and presumably a poor approximation of the true matrix A, unreliable for numerical inversion.

The same remark applies to the number n_{init} of samples used for initialization: it should be somewhat larger than the number of parameters at each unit, otherwise $A^{(0)}$ will be of low rank.

2 Constructing invariant algorithms: Riemannian metrics for neural networks

2.1 Gradient descents and metrics, natural metrics

The gradient of a function f on $\mathbb{R}^d$ gives the direction of steepest ascent: among all (very small) vectors with a given norm, it provides the greatest variation of f. Formally, the gradient ∇f of a smooth function f is defined by the property that

$$f(x+\varepsilon v) = f(x) + \varepsilon\langle \nabla f, v\rangle + O(\varepsilon^2) \tag{35}$$

for any vector v, for small enough ε. This depends on the choice of a scalar product $\langle\cdot,\cdot\rangle$. In an orthonormal basis, the coordinates of the gradient are simply the partial derivatives $\partial f/\partial x_i$ so that gradient descent is

$$x_i \leftarrow x_i - \eta\, \partial f/\partial x_i \tag{36}$$

in an orthonormal basis.

For a given norm of the vector v, the quantity $\langle \nabla f, v\rangle$ is maximal when v is collinear with ∇f: so the gradient ∇f indeed gives the direction of steepest ascent among all vectors with a given norm. The gradient step $x \leftarrow x - \eta\, \nabla f$ can actually be rewritten (for small enough η, up to $O(\eta^2)$ and for regular enough functions f) as

$$x \leftarrow \arg\min_y \left\{ f(y) + \frac{1}{2\eta} \|y-x\|^2 \right\} \tag{37}$$

namely, the gradient descent moves into the direction yielding the smallest values of f, penalized by the distance from the current point[4]. This allows to see how the choice of the scalar product will influence the direction of the gradient ∇f: indeed, another scalar product will define another norm $\|v\|^2 = \langle v, v\rangle$ for the vector v, so that the steepest direction among all vectors with a given norm will not be the same. The norm thus defines directions v in which it is "cheap" or "expensive" to move; the gradient is the direction of steepest ascent taking this into account.

If we happen to work in a non-orthonormal basis of vectors $v_1, \ldots, v_d$, the gradient is given by $A^{-1}\partial f/\partial x$ where $\partial f/\partial x$ is the vector of partial derivatives with respect to the coordinates x_i in the basis, and A is the symmetric matrix made of the scalar products of the basis vectors with themselves: $A_{ij} := \langle\, v_i \mid v_j \,\rangle$. Thus gradient descent of f takes the form

$$x \leftarrow x - \eta\, A^{-1}\partial f/\partial x \tag{38}$$

[4]This can be used to define or study the direction of the gradient in more general metric spaces (e.g., [AGS05, Chapter 2]).

Conversely, we can start with a norm on $\mathbb{R}^d$ defined through a positive-definite, symmetric matrix A (which thus defines "cheap" and "expensive" directions). The gradient ascent in the *metric* A will then be given by (38).

So any update of the form above with A a symmetric, positive-definite matrix can be seen as the gradient descent of f using some metric. The metric A may even depend on the current point x, defining a *Riemannian metric*.

An important feature of gradient descent, in any metric, is that for η small enough, the step is guaranteed to decrease the value of f.

The choice of a metric A represents the choice of a norm for vectors in parameter space. Conversely, choosing a set of parameters and using the "naive" gradient ascent for these parameters amounts to implicitly deciding that these parameters form an orthonormal basis.

For the neural network above, the gradient ascent $w_{ij} \leftarrow w_{ij} - \eta \frac{\partial \ell}{\partial w_{ij}}$ corresponds to the choice of $A = \mathrm{Id}$ on parameter space, namely, the norm of a change of parameters $\delta w = (\delta w_{ij})$ is $\|\delta w\|^2 := \sum |\delta w_{ij}|^2$. Thus, gradient descent for w_{ij} gives the best δw for a given norm $\|\delta w\|$, that is, the best change of f for a given change in the numerical value of the parameters.

Example: from sigmoid to tanh activation function. Neural networks using the sigmoid and tanh activation function are defined, respectively, by

$$a_k = \mathrm{sigm}(\sum_{i,\, i\to k} a_i w_{ik}) \tag{39}$$

and

$$a'_k = \tanh(\sum_{i,\, i\to k} a'_i w'_{ik}) \tag{40}$$

(including the biases w_{0k} and w'_{0k}). Since $\tanh(x) = 2\,\mathrm{sigm}(2x) - 1$, the activitives of the network correspond to each other via $a'_k = 2a_k - 1$ for all k if we set

$$w'_{ik} = \frac{w_{ik}}{4} \tag{41}$$

for $i \neq 0$, and

$$w'_{0k} = \frac{w_{0k}}{2} + \frac{1}{4}\sum_{i\neq 0} w_{ik} \tag{42}$$

for the biases.

Consequently, while the gradient for the sigmoid function will try to improve performance while minimizing the change to the numerical values of w_{ik} and w_{0k}, the gradient for the tanh function will do the same for the numerical values of w'_{ik} and w'_{0k}, obviously resulting in different updates. If we follow the tanh gradient and rewrite it back in terms of the variables w_{ik},

we see that the tanh update expressed in the variables w_{ik} is

$$w_{ik} \leftarrow w_{ik} + 16(\delta w_{ik} - \frac{1}{2}\delta w_{0k}) \qquad (i \neq 0) \tag{43}$$

and

$$w_{0k} \leftarrow w_{0k} + 4\delta w_{0k} - 8\sum_{i\neq 0}(\delta w_{ik} - \frac{1}{2}\delta w_{0k}) \tag{44}$$

where δw_{ik} is the update that would have been applied to w_{ik} if we were following the standard sigmoid backpropagation. Indeed this takes the form of a symmetric matrix applied to δw_{ik} (the cross-contributions of δw_{0k} to w_{ik} and of δw_{ik} to w_{0k} are the same).

Apart from an obvious speedup factor, an important difference between this update and ordinary (sigmoid) backpropagation on the w_{ik} is that each time a weight w_{ik} is updated, there is an opposite, twice as small contribution to w_{0k}: in this sense, it is as if this update assumes that the activities a_i are centered around $1/2$ so that when w_{ik} gets changed to $w_{ik} + c$, one "needs" to add $-c/2$ to the bias so that things stay the same on average.

Newton's method and gradient descent. To find the minimum of a function f on $\mathbb{R}$, one can use the Newton method to solve $f' = 0$, namely, $x \leftarrow x - f'(x)/f''(x)$. In higher dimension this becomes

$$x \leftarrow x - (\mathrm{Hess}\, f)^{-1}\partial f/\partial x \tag{45}$$

where $\partial f/\partial x$ is the vector of partial derivatives, and $(\mathrm{Hess}\, f)_{ij} := \partial^2 f/\partial x_i \partial x_j$ is the Hessian matrix of f.

Around a non-degenerate minimum of f, the Hessian $\mathrm{Hess}\, f$ will be a positive-definite matrix. So the Newton method can be seen as a gradient descent with learning rate $\eta = 1$, in the metric $A = \mathrm{Hess}\, f$, when one is close enough to a minimum.

Intrinsic metrics. There could be a lot of arguing and counter-arguing about the "right" way to write the parameters with respect to which the gradient should be taken. The solution to avoid these choices is known: use metrics that depend on what the system does, rather than on how the parameters are decomposed as numbers.

The *Fisher metric*, which defines a *natural gradient* [AN00], is one such metric. Namely: the size (norm) of a change of parameters is measured by the change it induces on the probability distribution of the output of the model. The symmetric matrix A used in the gradient update is then the *Fisher information matrix*. We will use scaled-down versions of the Fisher metric for better scalability.

We present another metric for neural networks, the *backpropagated metric*. The size of a change of parameters at a given unit is measured by the effect it has on the units it directly influences, which is itself measured recursively in the same way up to the output layer. The matrix defining this metric is obtained by well-known equations related to the Gauss–Newton approximation of the Hessian.

2.2 Intrinsic metrics and their computation by backpropagation

Here we rewrite the definition of neural networks in the language of differential manifolds and Riemannian geometry; this allows to define metrics directly in an intrinsic way.

Consider a neural-like network made of units influencing each other. The activity of each unit k takes values in a space $\mathcal{A}_k$ which we assume to be a differentiable manifold (typically $\mathbb{R}$ without a preferred origin and scale, but we allow room for multidimensional activations). Suppose that the activation of the network follows

$$a_k = f^k_{\theta_k}(a_{i_1}, \dots, a_{i_{n_k}}) \tag{46}$$

where $a_{i_1}, \dots, a_{i_{n_k}}$ are the units pointing to k, and where $f^k_{\theta_k}$ is a function from $\mathcal{A}_{i_1} \times \cdots \times \mathcal{A}_{i_{n_k}}$ to $\mathcal{A}_k$, depending on a parameter θ_k which itself belongs to a manifold Θ_k. Here we have no special, always-activated unit coding for biases: the biases are a part of the parameters θ_k.

We shall also assume that the output units in the network are interpreted through a final decoding function to produce an object $\omega = \omega((a_k)_{k \in \mathcal{L}_{\text{out}}})$ relevant to the initial problem, also assumed to belong to a differentiable manifold.

To implement any gradient ascent over the parameters θ, we first need a (Riemannian) metric on the parameter space. Such a metric can be defined by choosing a parametrization by $\mathbb{R}^d$ and deciding that the elementary vectors of $\mathbb{R}^d$ are orthogonal, but this is not intrinsic: different parametrizations will lead to different learning trajectories.

In this setting, an object is said to be *intrinsic* if it does not depend on a choice of parametrization of any of the manifolds involved (activities, parameters, final output). Hopefully, casting the activities as elements in an abstract manifold, and writing intrinsic algorithms that will not depend on how this manifold is represented as numbers, allows the algorithms to be agnostic as to any physical interpretation of these activities (activation levels, activation frequencies, log-frequencies, synchronized activity of a group of neurons...)

We assume that we are given a meaningful Riemannian metric on the final output ω: that is, we know how to measure the size of a change in the

output. For instance, if ω describes a probability distribution over a target variable y, we can use the Fisher metric over ω.

Then there are several possibilities to define intrinsic Riemannian metrics on parameter space. The most direct one is the Fisher metric: the output is seen as a function of all parameters, and the norm of a change of parameter $\delta\theta$ (over all parameters at once) is the norm of the change it induces on the output. This is not scalable.

A more scalable version is to break down the change of parameter into a sum of changes of incoming parameters to each unit and take the Fisher metric at each unit independently. This is the unitwise Fisher metric. As we will see, it scales well to sparsely connected networks if the output layer is not too large.

An even simpler version is the backpropagated metric, defined by backwards induction from the output layer: the norm of a change of parameter on the output layer is the norm of the change it induces on the final result, and the norm of a change of parameter at an internal unit is the sum of the norm of the resulting changes at the units it influences directly.

In what follows, we use the standard objects of differential geometry but try to present them in an intuitive way. The notation δa, $\delta\theta$, $\delta\omega$ denotes tangent vectors on the corresponding manifolds (intuitively, differences between two very close values of a or θ or ω). The notation $\frac{\partial a_i}{\partial a_k}$ denotes the differential of the activity a_i seen as a function of a_k (intuitively, this is just the Jacobian matrix of partial derivatives). The various metrics involved are $(0,2)$-tensors, but we use standard matrix notation for them.

Definition 13 (Natural metric, unitwise natural metric, backpropagated metric).
Let $\|\delta\omega\|^2 = \sum I_{ij}\delta\omega_i\delta\omega_j = \delta\omega^\top I\delta\omega$ be a metric on the final output of the network, given by the symmetric, positive-definite matrix I. We define three metrics on the parameter set.

- *The* natural metric *on the parameter set $\theta = (\theta_k)$ is defined as follows. Let x be an input in the dataset $\mathcal{D}$ and let $\omega(x)$ be the final output of the network run with input x and parameter θ. Let $\delta\theta$ be a variation of θ and let $\delta\omega(x)$ be the resulting variation of $\omega(x)$. Let*

$$\|\delta\theta\|^2_{\text{nat},x} := \|\delta\omega(x)\|^2 \tag{47}$$

and then define the natural metric by

$$\|\delta\theta\|^2_{\text{nat}} := \mathbb{E}_{x\in\mathcal{D}}\,\|\delta\theta\|^2_{\text{nat},x} \tag{48}$$

In matrix form, we have $\delta\omega(x) = \frac{\partial\omega(x)}{\partial\theta}\delta\theta$ where $\frac{\partial\omega}{\partial\theta}$ is the Jacobian matrix of $\omega(x)$ as a function of θ, so that the natural metric is given

by the matrix

$$\|\delta\theta\|^2_{\text{nat}} = \mathbb{E}_{x\sim\mathcal{D}}\delta\theta^\top \frac{\partial\omega(x)}{\partial\theta}^\top I \frac{\partial\omega(x)}{\partial\theta}\delta\theta \tag{49}$$

The natural metric is given by a matrix of size $\dim\theta = \sum_k \dim\theta_k$.

- *The* unitwise natural metric *on the parameter set* θ *is*

$$\|\delta\theta\|^2_{\text{u-nat}} := \sum_k \|\delta\theta_k\|^2_{\text{nat}} \tag{50}$$

where k *runs over the units of the network and* $\delta\theta_k$ *is the variation of the incoming parameters to unit* k*. This metric is given by keeping only the block-diagonal terms incoming to each unit in the matrix defining the natural metric.*

In case ω *is a probability distribution and the metric* I *on* ω *is the Fisher metric, we also call* $\|\delta\theta\|_{\text{nat}}$ *and* $\|\delta\theta\|_{\text{u-nat}}$ *the* Fisher metric *and* unitwise Fisher metric*.*

- *The* backpropagated metric *on* θ *is defined as follows. Let* x *be an input in the data. We first define a metric on each of the activities* a_k*, depending on the input* x*, working from the output layer backwards.*

  Given a change $\delta a_{k_{\text{out}}}$ *in the activity at an output unit* k_{out}*, let* $\delta\omega(x)$ *be the corresponding change in the final output and set*

$$\|\delta a_{k_{\text{out}}}\|^2_{\text{bp},x} := \|\delta\omega(x)\|^2 \tag{51}$$

The metric on internal units k *is defined as follows: Given a change* δa_k *in the activity of unit* k*, let* δa_i *be the resulting changes in the activities of units* $k \to i$ *directly influenced by* k*. Define by induction from the output layer*

$$\|\delta a_k\|^2_{\text{bp},x} := \sum_{i,\, k\to i} \|\delta a_i\|^2_{\text{bp},x} \tag{52}$$

Given a change $\delta\theta_k$ *of the incoming parameters to unit* k*, let* δa_k *be the resulting change of activity of unit* k *on input* x *and let*

$$\|\delta\theta_k\|^2_{\text{bp},x} := \|\delta a_k\|^2_{\text{bp},x} \tag{53}$$

and, finally, define the backpropagated metric by

$$\|\delta\theta_k\|^2_{\text{bp}} := \mathbb{E}_{x\in\mathcal{D}}\,\|\delta\theta_k\|^2_{\text{bp},x} \tag{54}$$

and

$$\|\delta\theta\|^2_{\text{bp}} := \sum_{k\in\mathcal{L}} \|\delta\theta_k\|^2_{\text{bp}} \tag{55}$$

Another metric, the *tensor square differential* (TSD) metric, can be defined from slightly different ingredients. It corresponds to an often-used variant of the natural gradient (e.g., [APF00, RMB07]), in which the expectation under the current probability distribution is replaced with a similar term involving only the desired target y for each input x (more details in Section 2.5). As the formula (58) below shows, it can be readily computed by backpropagation.

Whereas the metrics above depend on the actual output $\omega(x)$ for each input x, together with a metric on ω, but not on any target value for x, the TSD metric depends on a loss function $\ell(\omega(x), y(x))$ encoding the deviation of $\omega(x)$ from a desired target $y(x)$ for x; but not on a choice of metric for ω.

Definition 14 (Tensor square differential metric). *For each input x in the dataset $\mathcal{D}$, let $\omega(x)$ be the final output of the network run with input x and parameter θ. Let $\ell(\omega(x), y(x))$ be the loss function measuring how $\omega(x)$ departs from the desired output $y(x)$ for x.*

The tensor square differential metric *is defined as follows. Let $\delta\theta$ be a variation of θ and let $\delta\ell_x$ be the resulting variation of $\ell(\omega(x), y(x))$. Define*

$$\|\delta\theta\|^2_{\text{tsd}} := \mathbb{E}_{x\in\mathcal{D}}(\delta\ell_x)^2 \tag{56}$$

In matrix form, this metric is

$$\|\delta\theta\|^2_{\text{tsd}} = \mathbb{E}_{x\sim\mathcal{D}}\delta\theta^\top \frac{\partial\ell}{\partial\theta}^\top \frac{\partial\ell}{\partial\theta}\delta\theta \tag{57}$$

where $\frac{\partial\ell}{\partial\theta}$ is the row vector of partial derivatives (the differential) of the loss function. Thus this metric is given by the matrix

$$\mathbb{E}_{x\sim\mathcal{D}}\frac{\partial\ell}{\partial\theta}^\top \frac{\partial\ell}{\partial\theta} \tag{58}$$

hence its name.

The unitwise tensor square differential metric *is defined by*

$$\|\delta\theta\|^2_{\text{u-tsd}} := \sum_k \|\delta\theta_k\|^2_{\text{nat}} \tag{59}$$

where k runs over the units of the network and $\delta\theta_k$ is the variation of the incoming parameters to unit k. This metric is given by keeping only the block-diagonal terms incoming to each unit in the matrix defining the tensor square differential metric.

The TSD metric has been used simply under the name "natural gradient" in [APF00, RMB07], which can lead to some confusion because it is distinct from the natural metric using the true Fisher information matrix (see the discussion in [PB13]). Since moreover the TSD metric makes sense for optimization situations more general than the natural gradient, in which the loss function is not necessarily of the form $\ln p_\theta$ for a probabilistic model p, we chose a purely descriptive name.

Remark 15 (TSD gradient equalizes the gain over the samples).
The TSD metric is (up to a scaling factor) the only way to define a gradient step for which progress is most evenly distributed among all data samples. Namely, if the average loss if defined as $L = \mathbb{E}_{x\in\mathcal{D}}\ell$, then the direction $\delta\theta$ given by the gradient of L computed in the TSD metric (Section 2.4) has the following property: Among all directions $\delta\theta$ yielding the same infinitesimal increment δL at first order, it is the one for which the increment is most evenly spread over the data samples $x \in \mathcal{D}$, in the sense that $\mathrm{Var}_{x\in\mathcal{D}}\, \delta\ell$ is minimal. A proof is given in the Appendix.

The same does not hold, in general, for the unitwise TSD metric.

For the Fisher metric, the following is well-known.

Proposition 16 (Invariance).
The natural metric, unitwise natural metric, backpropagated metric, and plain and unitwise tensor square differential metrics are intrinsic: $\|\delta\theta\|_{\text{nat}}$, $\|\delta\theta\|_{\text{u-nat}}$, $\|\delta\theta\|_{\text{bp}}$, $\|\delta\theta\|_{\text{tsd}}$, *and* $\|\delta\theta\|_{\text{u-tsd}}$ *do not depend on a choice of parametrization for the activations* a_k *and for the parameter* θ_k *at each unit* k.

Proof. These metrics have been defined without any reference to parametrizations. □

The natural metric actually has stronger invariance properties than the unitwise natural metric: it does not depend on a change of parametrization of the whole parameter $\theta = (\theta_k)$ that would mix the various components. As such, the unitwise natural metric depends on a choice of decomposition of the network into units, while the natural metric is only a function of the input-output relationship of the whole network. The same holds for the TSD and unitwise TSD metrics.

Remark 17 (Unitwise metrics as change in activation profile).
We saw above that the metric used to define a gradient represents a "cost" of moving in certain directions. All three unitwise metrics (unitwise natural, backpropagated, and unitwise TSD) share a common property: these metrics decompose as a sum, over the units k, of terms of the form $\|\delta\theta_k\|^2 = \mathbb{E}_{x\in\mathcal{D}}\, c_{x,k}\, \|\delta a_k(x)\|^2$ where $\delta a_k(x)$ is the resulting change of activity at k on input x, and $c_{x,k}$ is a weight (different for these three metrics) estimating the influence of k on the output. Thus, the "cost" of a change at unit k according to these metrics, is an average square norm of the resulting *change in activation profile* $a_k(x)$ over x in the dataset. This is related to the *best-fit interpretation* of these metrics (Section 3.3).

To compute the natural and unitwise natural metrics, it is enough to compute the Jacobian matrix $\frac{\partial \omega}{\partial \theta}$. This can be done by performing one backpropagation for each component of the output layer, for each input $x \in \mathcal{D}$, as follows.

Definition 18 (Backpropagation transfer rates).
Let k_{out} be an output unit and let k be any unit in the network. The backpropagation transfer rate $J_k^{k_{\text{out}}}$ *from k_{out} to k is the* $\dim(a_{k_{\text{out}}}) \times \dim(a_k)$ *matrix defined by*

$$\begin{cases} J_{k_{\text{out}}}^{k_{\text{out}}} := \mathrm{Id}_{\dim(a_{k_{\text{out}}})} & \\ J_k^{k_{\text{out}}} := 0 & \text{for } k \neq k_{\text{out}} \text{ in the output layer } \mathcal{L}_{\text{out}} \\ J_k^{k_{\text{out}}} := \sum_{j,\, k \to j} J_j^{k_{\text{out}}} \frac{\partial a_j}{\partial a_k} & \text{for non-output units } k \end{cases} \tag{60}$$

where $\frac{\partial a_j}{\partial a_k}$ is the Jacobian matrix of the activation function from unit k to unit j. Then we have $J_k^{k_{\text{out}}} = \frac{\partial a_{k_{\text{out}}}}{\partial a_k}$.

This depends on an input x: the activation state of the network has to be computed by forward propagation before these quantities can be computed.

Typically the activities are one-dimensional, not multidimensional, so that each $J_k^{k_{\text{out}}}$ is just a number, not a matrix. In this case, all the transfer rates $J_k^{k_{\text{out}}}$ can be computed by performing n_{out} distinct backpropagations each initialized with a single 1 on the output layer.

Since the influence of the parameter θ_k on the output channels through the activity of unit k, the unitwise natural metric at k can be computed from a single number (if activities are one-dimensional) measuring the influence of unit k on the output, the *Fisher modulus*.

Definition 19 (Fisher modulus).
Let x be an input. Let k be a unit in the network. Let I be the metric on the final output ω. The Fisher modulus $\Phi_k(x)$ *of k on input x is the* $\dim(a_k) \times \dim(a_k)$ *matrix given by*

$$\Phi_k(x) := \left(\sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} \frac{\partial \omega}{\partial a_{k_{\text{out}}}} J_k^{k_{\text{out}}} \right)^\top I \left(\sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} \frac{\partial \omega}{\partial a_{k_{\text{out}}}} J_k^{k_{\text{out}}} \right) \tag{61}$$

For each input x, the Fisher modulus is an intrinsic metric on a_k: for a given input x, the norm

$$\|\delta a_k\|_{\text{F-mod}}^2 := \delta a_k^\top \Phi_k \delta a_k \tag{62}$$

does not depend on any choice of parametrization.

Note that $\frac{\partial \omega}{\partial a_{k_{\text{out}}}}$ depends on the output layer interpretation but not on any parameter θ. Thus, since the transfer rates J can be computed by backpropagation, the Fisher modulus only involves known quantities.

Proposition 20 (Computation of the unitwise natural metric).
The unitwise natural metric at unit k is given by

$$\|\delta\theta_k\|^2_{\text{u-nat}} = \mathbb{E}_{x\in\mathcal{D}}\ \|\delta a_k(x)\|^2_{\text{F-mod}} \tag{63}$$

$$= \mathbb{E}_{x\in\mathcal{D}}\ \delta\theta_k^\top \frac{\partial a_k}{\partial \theta_k}^\top \Phi_k \frac{\partial a_k}{\partial \theta_k}\delta\theta_k \tag{64}$$

where $\delta a_k(x)$ is the variation of $a_k(x)$ induced by $\delta\theta$, and $\frac{\partial a_k}{\partial \theta_k}$ is the Jacobian matrix of the activation function at k. Thus the matrix defining the unitwise natural metric at unit k is

$$F^{(k)} = \mathbb{E}_{x\in\mathcal{D}} \frac{\partial a_k}{\partial \theta_k}^\top \Phi_k \frac{\partial a_k}{\partial \theta_k} \tag{65}$$

Proof. By definition of the transfer rates J we have $J_k^{k_{\text{out}}} = \frac{\partial a_{k_{\text{out}}}}{\partial a_k}$. Thus $\sum_{k_{\text{out}}\in\mathcal{L}_{\text{out}}} \frac{\partial \omega}{\partial a_{k_{\text{out}}}} J_k^{k_{\text{out}}} = \frac{\partial \omega}{\partial a_k}$ so that

$$\Phi_k = \frac{\partial \omega}{\partial a_k}^\top I \frac{\partial \omega}{\partial a_k} \tag{66}$$

hence

$$\frac{\partial a_k}{\partial \theta_k}^\top \Phi_k \frac{\partial a_k}{\partial \theta_k} = \frac{\partial a_k}{\partial \theta_k}^\top \frac{\partial \omega}{\partial a_k}^\top I \frac{\partial \omega}{\partial a_k}\frac{\partial a_k}{\partial \theta_k} = \frac{\partial \omega}{\partial \theta_k}^\top I \frac{\partial \omega}{\partial \theta_k} \tag{67}$$

which, after averaging over the dataset, is the definition of the unitwise natural metric at k. □

An analogous formula can be defined for the full (rather than unitwise) Fisher matrix, by defining a Fisher modulus $\Phi_{kk'}$ indexed by two units, and using unit k' on the left and k on the right in (61). Then the block of entries of the Fisher matrix corresponding to parameters θ_k and $\theta_{k'}$ is

$$\mathbb{E}_{x\in\mathcal{D}} \frac{\partial a_{k'}}{\partial \theta_{k'}}^\top \Phi_{kk'} \frac{\partial a_k}{\partial \theta_k} \tag{68}$$

(see also Proposition 26).

The unitwise Fisher metric is costly to compute when the output layer is large. We can define another intrinsic metric for the activation of unit k, simply by backpropagating the metric of the output layer.

The changes in the output induced by a change of θ_k all transit through the activation of unit k. So if we have an intrinsic metric $\|\delta a_k\|^2$ for the activation of unit k, we can immediately define an intrinsic metric for θ_k, by looking at the resulting change $\delta a_k = \frac{\partial a_k}{\partial \theta_k}\delta\theta_k$ induced by a change $\delta\theta_k$, and defining the norm of $\delta\theta_k$ to be the norm of the resulting δa_k. If the metric

on a_k is given, in some parametrization of a_k, by $\|\delta a_k\|^2 = \delta a_k^\top g_k \delta a_k$ where g_k is a symmetric, positive-definite matrix of size $(\dim a_k) \times (\dim a_k)$, then defining $\|\delta\theta_k\|$ to be the norm of this δa_k,

$$\|\delta\theta_k\| := \left\| \frac{\partial a_k}{\partial \theta_k} \delta\theta_k \right\| \tag{69}$$

yields

$$\|\delta\theta_k\|^2 = \left(\frac{\partial a_k}{\partial \theta_k} \delta\theta_k \right)^\top g_k \left(\frac{\partial a_k}{\partial \theta_k} \delta\theta_k \right) \tag{70}$$

in other words, the matrix defining this metric is $\frac{\partial a_k}{\partial \theta_k}^\top g_k \frac{\partial a_k}{\partial \theta_k}$.

The unitwise Fisher metric is obtained from the Fisher modulus by this construction. We now define another intrinsic modulus playing the same role for the backpropagated metric.

Proposition 21 (Backpropagated modulus and computation of the backpropagated metric).
Let x be an input. Let k be a unit in the network. Let I be the metric on the final output ω. The backpropagated modulus $m_k(x)$ *at k is the* $\dim(a_k) \times \dim(a_k)$ *matrix given by*

$$m_k(x) := \begin{cases} \frac{\partial \omega}{\partial a_k}^\top I \frac{\partial \omega}{\partial a_k} & \text{for } k \text{ in the output layer} \\ \sum_{j,\, k \to j} \frac{\partial a_j}{\partial a_k}^\top m_j \frac{\partial a_j}{\partial a_k} & \text{for } k \text{ an internal unit} \end{cases} \tag{71}$$

Then, for each input x, the backpropagated metric on a_k is given by the backpropagated modulus:

$$\|\delta a_k\|^2_{\mathrm{bp},x} = \delta a_k^\top m_k \delta a_k \tag{72}$$

and so the backpropagated metric on θ_k is given by the matrix $\mathbb{E}_{x\in\mathcal{D}} \frac{\partial a_k}{\partial \theta_k}^\top m_k \frac{\partial a_k}{\partial \theta_k}$, namely,

$$\|\delta\theta_k\|^2_{\mathrm{bp}} = \mathbb{E}_{x\in\mathcal{D}}\, \delta\theta_k^\top \frac{\partial a_k}{\partial \theta_k}^\top m_k \frac{\partial a_k}{\partial \theta_k} \delta\theta_k \tag{73}$$

Proof. Immediate from the definition of the backpropagated metric and $\delta a_i = \frac{\partial a_i}{\partial a_k} \delta a_k$ and $\delta a_k = \frac{\partial a_k}{\partial \theta_k} \delta\theta_k$. □

Like the Fisher modulus, the backpropagated modulus is a single number when activities are one-dimensional. The cost of its computation is the same as one backpropagation pass.

The equation defining the backpropagated modulus is well-known: it is related to the so-called Gauss–Newton approximation to the Newton method (see for instance [LBOM96], Section 7), which consists in computing the Hessian of the loss function and throwing away all terms involving the second derivative of the activation function (those could result in non–positive-definite terms, in which case the Newton method is ill-behaved), with the

additional approximation that cross-terms between different units are also thrown away. Here we see that no approximation is involved: we do not throw away annoying terms, we simply define an intrinsic metric. There is actually no meaningful notion of the Hessian of a function on manifolds [GHL87, paragraph 3.37] unless additional structure (affine, Riemannian) is given or we are at a critical point of the function; the annoying terms above are precisely the terms that prevent such a notion to exist. So one could even say, in the context of differential geometry, that the Newton method is an approximation of the backpropagated metric rather than the other way round.

The backpropagated modulus and the Fisher modulus are related: If one tries to write a backpropagated equation to compute the Fisher modulus Φ_k in terms of the Fisher modulus at units pointed by k, one finds a quadratic (instead of linear) backpropagation equation with terms involving pairs of units. Keeping only the terms involving a single unit yields the equation defining the backpropagated modulus.

2.3 Quasi-diagonal reduction of a unitwise metric

The unitwise Fisher metric, backpropagated metric, and unitwise TSD metric, still involve a full matrix on the incoming parameter space at each unit, and are thus not adapted if network connectivity is large. We now introduce two metrics enjoying lesser invariance properties than the above, but quicker to compute.

Given an intrinsic metric $\|\delta\theta_k\|$ on θ_k (such as the unitwise Fisher or backpropagated metric), we are going to define a simpler one, $\|\delta\theta_k\|_{\mathrm{qd}}$. The inverse of the matrix defining this metric will be *quasi-diagonal*, with the only non-zero diagonal terms being those between a weight and the bias. This will allow for quick gradient steps costing no more than classical backpropagation.

This relies on the affine structure in neural networks: this simplification makes sense in a somewhat more restricted setting than the general setting above. Suppose that the activation function

$$a_k = f^k_{\theta_k}(a_{i_1}, \dots, a_{i_{n_k}}) \tag{74}$$

can be written as a composition of a fixed, non-linear activation function φ and a quantity y_k that is an affine function of $a_{i_1}, \dots, a_{i_{n_k}}$:

$$a_k = \varphi(y_{k,\theta_k}(a_{i_1}, \dots, a_{i_{n_k}})) \tag{75}$$

such that when θ_k ranges over its values, y_{k,θ_k} ranges over all possible affine functions of $a_{i_1}, \dots, a_{i_{n_k}}$. For this to make sense, we now have to assume

that the activities a_k live in an affine space. So let us go back[5] to activities with values in $\mathbb{R}$, but without any preferred origin and scale for activities: we look for invariance under replacement of a_i with $\alpha_i a_i + \beta_i$.

In any given parametrization (choice of origin and basis) of a_i and y_k we can write

$$y_k = \sum_{i,\, i \to k} w_{ik} a_i + w_{0k} \tag{76}$$

for some values w_{ik}; specifying the parameter θ_k is equivalent to specifying these quantities.

But this decomposition will change if we change the affine parametrization of activities: if $a'_i = \alpha_i a_i + \beta_i$ and $y'_k = \gamma_k y_k + \delta_k$ the relation becomes $y'_k = \delta_k + \sum_i \gamma_k w_{ik} \alpha_i^{-1}(a'_i - \beta_i) + \gamma_k w_{0k} = \sum_i (\gamma_k w_{ik} \alpha_i^{-1}) a'_i + (\gamma_k w_{0k} + \delta_k - \sum_i w_{ik} \alpha_i^{-1} \beta_i)$ so that the new weights are $w'_{ik} = \gamma_k w_{ik} \alpha_i^{-1}$ and the new bias is $w'_{0k} = \gamma_k w_{0k} + \delta_k - \sum_i w_{ik} \alpha_i^{-1} \beta_i$. In particular we see that there is no intrinsic "separation" between the bias and the weights; but that there is a separation between w_{ik} and $w_{i'k}$ for different incoming units i and i'. This is formalized as follows.

Let $\delta\theta_k$ be a change of parameter θ_k and let δw_{ik} be the resulting change of w_{ik}, with $i \neq 0$. If $\delta w_{ik} = 0$ in one parametrization, then we have $\delta w'_{ik} = 0$ as well in any other affine parametrization since $w'_{ik} = \gamma_k w_{ik} \alpha_i^{-1}$. Say that a change of parameter $\delta\theta_k$ *does not involve unit* i if δw_{ik} vanishes: this does not depend on the chosen affine parametrization of activities and is thus intrinsic.

Moreover, having $\delta w_{ik} = 0$ does not depend on the input or current activation state of the network: this is where we use that y_k is an affine function of the (a_i).

For the bias the situation is different: the expression $w'_{0k} = \gamma_k w_{0k} + \delta_k - \sum_i w_{ik} \alpha_i^{-1} \beta_i$ giving the bias in a parametrization from the bias in another parametrization is more complex, and so the fact that $\delta w_{0k} = 0$ does depend on the parametrization. This is where the metric $\|\delta\theta_k\|$ we are trying to simplify comes into play.

Say that a change of parameter $\delta\theta_k$ is *pure bias* if it does not involve any unit i incoming to k, i.e., if $\delta w_{ik} = 0$ for all $i \neq 0$. Say that $\delta\theta_k$ is *bias-free* if it is orthogonal, in the metric $\|\delta\theta_k\|$ we are trying to simplify, to all pure-bias vectors. Thus being bias-free does not simply mean $\delta w_{0k} = 0$. Being bias-free is an intrinsic condition; let us work it out.

Let $A_{ii'}$ be the symmetric matrix defining the metric $\|\delta\theta_k\|$ in a given

[5] Technically, this would be best described with activities taking values in affine manifolds. The "weights" are then the partial derivatives $\partial y_k / \partial a_i$, which are well-defined linear maps from the tangent space of a_i to the tangent space of y_k.

parametrization. The associated scalar product is

$$\langle \delta\theta_k, \delta\theta'_k \rangle = \sum_i \sum_{i'} A_{ii'} \delta w_{ik} \delta w'_{i'k} + \sum_i A_{0i} (\delta w_{0k} \delta w'_{ik} + \delta w'_{0k} \delta w_{ik}) + A_{00} \delta w_{0k} \delta w'_{0k} \tag{77}$$

with $A_{0i} = A_{i0}$.

In particular, if the only non-zero component of $\delta\theta_k$ is δw_{ik}, then its scalar product with a pure bias $\delta w'_{0k}$ will be $A_{0i} \delta w'_{0k} \delta w_{ik}$. On the other hand, if to $\delta\theta_k$ we add a bias component $\delta w_{0k} = -A_{00}^{-1} A_{0i} \delta w_{ik}$, then the scalar product with any pure bias will vanish. Such a $\delta\theta_k$ is thus bias-free.

In the case when the parameter θ_k allows to represent all affine functions of the incoming activations, we can decompose a variation $\delta\theta_k$ of θ_k into components $\delta\theta_{ki}$ each involving only one incoming unit i, and a pure bias component $\delta\theta_{k0}$. This decomposition is unique if we impose that each $\delta\theta_{ki}$ is bias-free. Explicitly, if in some parametrization we have $\delta\theta_k = (\delta w_{0k}, \delta w_{1k}, \ldots, \delta w_{n_k k})$ this decomposition is

$$\delta\theta_{ki} = (-A_{00}^{-1} A_{0i} \delta w_{ik}, 0, \ldots, \delta w_{ik}, 0, \ldots, 0) \tag{78}$$

and

$$\delta\theta_{k0} = (\delta w_{0k} + \sum_i A_{00}^{-1} A_{0i} \delta w_{ik}, 0, \ldots, 0) \tag{79}$$

The decomposition $\delta\theta_k = \delta\theta_{k0} + \sum_i \delta\theta_{ki}$ is intrinsic.

We can then define a new intrinsic metric on $\delta\theta_k$ by setting

$$\|\delta\theta_k\|^2_{\mathrm{qd}} := \|\delta\theta_{k0}\|^2 + \sum_i \|\delta\theta_{ki}\|^2 \tag{80}$$

which is readily computed:

$$\begin{aligned} \|\delta\theta_k\|^2_{\mathrm{qd}} &= A_{00} \left(\delta w_{0k} + \sum_i A_{00}^{-1} A_{0i} \delta w_{ik} \right)^2 \\ &\quad + \sum_i (A_{ii} \delta w_{ik}^2 - 2 g_{0i} (A_{00}^{-1} A_{0i} \delta w_{ik}) \delta w_{ik} + A_{00} (A_{00}^{-1} A_{0i} \delta w_{ik})^2) \\ &= A_{00} \delta w_{0k}^2 + 2 \sum_i A_{0i} \delta w_{0k} \delta w_{ik} + \sum_{i,i'} A_{00}^{-1} A_{0i} A_{0i'} \delta w_{ik} \delta w_{i'k} \\ &\quad + \sum_i (A_{ii} - A_{00}^{-1} A_{0i}^2) \delta w_{ik}^2 \end{aligned} \tag{81}$$

Thus, this metric is defined by a matrix $\tilde{A}$ given by $\tilde{A}_{00} = A_{00}$, $\tilde{A}_{0i} = A_{0i}$ and $\tilde{A}_{ii'} = A_{00}^{-1} A_{0i} A_{0i'} + \mathbb{1}_{i=i'} (A_{ii} - A_{00}^{-1} A_{0i}^2)$.

Definition 22.
Quasi-diagonal reduction *is the process which, to an intrinsic metric defined by a matrix A in affine coordinates, associates the metric defined by the matrix*

$$\tilde{A} := \mathrm{diag}(A) + A_{00}^{-1} (v \otimes v) - \mathrm{diag}(A_{00}^{-1} (v \otimes v)) \tag{82}$$

where

$$v = (A_{00}, \dots, A_{0i}, \dots) \tag{83}$$

The quasi-diagonal backpropagated metric *is the quasi-diagonal metric obtained from the backpropagated metric. The* quasi-diagonal Fisher metric *is the one obtained from the unitwise Fisher metric. The* quasi-diagonal TSD metric *is the one obtained from the unitwise TSD metric.*

The reasoning in this section can be summarized as follows.

Proposition 23.
Assume that the activation function is a fixed non-linear function composed with an affine function. Then the quasi-diagonal reduction $\tilde{A}$ of an intrinsic metric A is intrinsic.

Importantly, the matrix $\tilde{A} = \mathrm{diag}(A) + A_{00}^{-1}(v \otimes v) - \mathrm{diag}(A_{00}^{-1}(v \otimes v))$ is the sum of a diagonal matrix and a rank-1 matrix. This allows for easy inversion, resulting in a quasi-diagonal inverse matrix.

Proposition 24 (Quasi-diagonal gradient step).
Let $\tilde{A}$ be the quasi-diagonal reduction of A. Let $b = (b_0, \dots, b_i, \dots)$ and $w = \tilde{A}^{-1}b$. Then w is given by

$$w_i = \frac{b_i A_{00} - b_0 A_{0i}}{A_{ii} A_{00} - A_{0i}^2} \qquad \text{for } i \neq 0 \tag{84}$$

$$w_0 = \frac{b_0}{A_{00}} - \sum_{i \neq 0} \frac{A_{0i}}{A_{00}} w_i \tag{85}$$

Thus, only the entries A_{00}, A_{ii} and A_{0i} of the original matrix A need to be known in order to implement gradient descent using the quasi-diagonal metric defined by $\tilde{A}$.

Note that if the original matrix A is positive-definite, we have $A_{00} > 0$ and $A_{00}A_{ii} > A_{0i}^2$ (by the Cauchy–Schwarz inequality applied to the first and i-th basis vectors), so that the solution w above is well-defined and unique.

2.4 Intrinsic gradients

Using these intrinsic metrics allows to define an intrinsic gradient direction in parameter space. Given a dataset $\mathcal{D}$ of inputs x and corresponding targets y, the average loss function is

$$L_\theta := \mathbb{E}_{x \in \mathcal{D}} \ell_\theta(y) \tag{86}$$

where we put a subscript θ to make explicit its dependency on the parameters of the network. Given an intrinsic metric $\|\cdot\|$, the differential

$$G = -\frac{\partial L_\theta}{\partial \theta} \tag{87}$$

of the average loss with respect to the full parameter set θ, defines a gradient direction $\nabla_\theta L$ by the usual definition: it is the only tangent vector such that for any $\delta\theta$ we have

$$L_{\theta+\delta\theta} = L_\theta + \langle \nabla_\theta L, \delta\theta \rangle + O(\|\delta\theta\|^2) \tag{88}$$

where $\langle \cdot, \cdot \rangle$ is the scalar product associated with the norm $\|\cdot\|$. In a parametrization where $\|\cdot\|^2$ is given by a symmetric, positive definite matrix A, the gradient is given by

$$\nabla_\theta L = A^{-1} \frac{\partial L}{\partial \theta} = -A^{-1} G \tag{89}$$

The gradient $\nabla_\theta L$ is an intrinsic tangent vector on the parameter set.

Definition 25.
The natural gradient, unitwise natural gradient, backpropagated metric gradient, TSD gradient, unitwise TSD gradient, and their quasi-diagonal reductions, respectively, are the following update rule for θ:

$$\theta \leftarrow \theta - \eta \nabla_\theta L \tag{90}$$

where $\nabla_\theta L$ is the gradient of the average loss function L computed in the natural metric, unitwise natural metric, backpropagated metric, TSD metric, unitwise TSD metric, and their quasi-diagonal reductions, respectively.

The algorithms of Section 1 are the application of these updates to ordinary neural networks, written out with $[0;1]$-valued activities and sigmoid activation function. More details on how this works out are given below (Section 2.5).

This update is intrinsic only under all *affine* reparametrizations of the parameter θ. Indeed, even if the tangent vector $\nabla_\theta L$ giving the direction of the gradient is fully intrinsic, adding a tangent vector to a parameter θ is not an intrinsic operation (if two parametrizations differ by a non-affine transformation, then the additions will not amount to the same). On the other hand, the ideal limit when the learning rate η tends to 0 is intrinsic: the trajectories of the differential equation

$$\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = -\nabla_{\theta(t)} L \tag{91}$$

are intrinsic trajectories in parameter space for the unitwise natural gradient and backpropagated metric.

For the quasi-diagonal algorithms, invariance is always restricted to affine reparametrizations, since this is the setup in which they are well-defined.

2.5 The Fisher matrix for neural networks

Applying the general definitions above to ordinary neural networks with sigmoid activation function leads to the algorithms of Section 1. This is mostly by direct computation and we do not reproduce it fully. Let us however discuss in more detail the case of the Fisher metric.

For each input x, the network defines a probability distribution on the outputs y. This probability distribution depends on the parameters of the network. Thus, for each input x, we can define a *datum-wise* Fisher matrix on the parameter set:

$$F(x)_{w_{ij}w_{i'j'}} = \mathbb{E}_{y|x} \frac{\partial \ell(y)}{\partial w_{ij}} \frac{\partial \ell(y)}{\partial w_{i'j'}} \tag{92}$$

The dataset together with the network define a probability distribution on pairs (x, y), by first choosing at random an input x in the dataset, then running the network on this input. The Fisher matrix associated with this distribution on pairs (x, y) is the average of the datum-wise Fisher matrix over the dataset

$$F = \mathbb{E}_{x \in \mathcal{D}} F(x) \tag{93}$$

(see [AN00], Section 8.2), or more explicitly

$$F_{w_{ij}w_{i'j'}} = \mathbb{E}_{x \in \mathcal{D}} \mathbb{E}_{y|x} \frac{\partial \ell(y)}{\partial w_{ij}} \frac{\partial \ell(y)}{\partial w_{i'j'}} \tag{94}$$

Exact Fisher matrix versus one-sample Fisher matrix. One possible way to train neural networks using the natural gradient is to estimate the Fisher matrix by taking a random input x in the dataset, taking a random output y for this input, and add the term $\frac{\partial \ell(y|x)}{\partial w_{ij}} \frac{\partial \ell(y|x)}{\partial w_{i'j'}}$ to the current estimate of the Fisher matrix (with a discount factor for older contributions).

An important variant uses for y the target value for input x, instead of taking y as a random sample given by the activations of the output layer:

$$\hat{F}_{w_{ij}w_{i'j'}} = \mathbb{E}_{x \in \mathcal{D}} \frac{\partial \ell(y(x))}{\partial w_{ij}} \frac{\partial \ell(y(x))}{\partial w_{i'j'}} \tag{95}$$

with y the target for x: this coincides with the *tensor square differential metric* of Definition 14. Hopefully, when the network converges towards the desired targets, these two variants coincide. This variant has been present for a long time in studies on natural gradient (as is clear, e.g., from Equation (14) in [APF00]) and is elaborated upon in [RMB07]. As pointed out in [PB13], the two variants are often confused.

The natural gradient is intrinsic in these two variants. The second variant, contrary to the natural gradient, depends on the targets and not only on the network and dataset, and is thus not naturally interpreted as a Riemannian metric for neural networks.

Both the full natural gradient and its "one-sample" variant give rise to a unitwise version and to a quasi-diagonal version (Section 2.3).

In Section 4 we compare performance of the unitwise natural gradient and unitwise one-sample/TSD natural gradient.

Exact Fisher matrix computation. Our framework allows to compute the exact Fisher matrix instead of using a single value for y, by using the Fisher modulus and backpropagation transfer rates. The latter can be computed by doing n_{out} backpropagations for each input. This is of course more convenient than summing over the (in the Bernoulli case) $2^{n_{\text{out}}}$ possible outcomes for y.

Remember the backpropagation transfer rates $J_k^{k_{\text{out}}}$ from Definition 2. The latter simply implements the general Definition 18 for ordinary neural networks. In Section 1, the unitwise natural gradient was obtained from these transfer rates through the Fisher modulus. Here we reproduce the corresponding formula for all terms of the Fisher matrix, not only the terms of the unitwise Fisher matrix incoming to a given unit, so we introduce a Fisher modulus indexed by pairs of units.

Proposition 26 (Exact Fisher matrix for neural networks). *Let x be an input for the network. Compute the transfer rates $J_k^{k_{\text{out}}}$ as in Definition 2. Depending on output layer interpretation, set for each pair of units k and k':*

$$\begin{cases} \Phi_{kk'}(x) := \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} \frac{J_k^{k_{\text{out}}} J_{k'}^{k_{\text{out}}}}{a_{k_{\text{out}}}(1-a_{k_{\text{out}}})} & \textit{(Bernoulli)} \\ \Phi_{kk'}(x) := \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} J_k^{k_{\text{out}}} J_{k'}^{k_{\text{out}}} & \textit{(square-loss)} \\ \Phi_{kk'}(x) := \frac{4}{S} \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} J_k^{k_{\text{out}}} J_{k'}^{k_{\text{out}}} & \textit{(classification)} \\ \qquad - \frac{4}{S^2} \left(\sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} a_{k_{\text{out}}} J_k^{k_{\text{out}}} \right) \left(\sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} a_{k_{\text{out}}} J_{k'}^{k_{\text{out}}} \right) & \\ \qquad \textit{where } S := \sum_{k_{\text{out}} \in \mathcal{L}_{\text{out}}} a_{k_{\text{out}}}^2 & \end{cases} \tag{96}$$

Then the entry of the datum-wise Fisher matrix $F(x)$ associated with parameters w_{ik} and $w_{jk'}$ (including biases with $i = 0$ or $j = 0$) is

$$F(x)_{w_{ik} w_{jk'}} = a_i a_j a_k (1 - a_k) a_{k'} (1 - a_{k'}) \Phi_{kk'} \tag{97}$$

and consequently the same entry in the Fisher matrix is

$$F_{w_{ik} w_{jk'}} = \mathbb{E}_{x \in \mathcal{D}}\, a_i a_j a_k (1 - a_k) a_{k'} (1 - a_{k'}) \Phi_{kk'} \tag{98}$$

The proof is given in the Appendix and is a more or less straightforward application of the results of the previous section, together with an explicit computation of the Fisher metric on the output in the Bernoulli, square-loss, and classification interpretations.

So it is possible to compute the full Fisher matrix by performing n_{out} independent backpropagations for each sample input. The Fisher matrix F, being the average of $F(x)$ over the dataset, may be approximated by the standard online or small-batch techniques using samples from the dataset.

For a network with only one hidden layer, this simplifies and no additional backpropagations are needed. Indeed, the backpropagation transfer rates of the input layer are never used, and on the hidden layer are given by

$$J_k^{k_{\text{out}}} = w_{kk_{\text{out}}} a_{k_{\text{out}}} (1 - a_{k_{\text{out}}}) \tag{99}$$

from which the Fisher modulus can be immediately computed. This is the case treated in [Kur94] (for the Bernoulli interpretation).

3 Some properties of unitwise algorithms and their quasi-diagonal approximations

3.1 Performance improvement at each step

A common feature of all gradient-based algorithms in any metric is that the objective function improves at each step provided the learning rate is small enough. Consequently this holds for the unitwise natural gradient, backpropagated metric gradient, and their quasi-diagonal reductions.

Proposition 27.
Suppose that training has not reached a local optimum, i.e., that the gradient vector G of Section 1 does not vanish. Suppose that the metric considered is non-degenerate (i.e., respectively, that the matrices $F^{(k)}$ or $M^{(k)}$ are invertible, or that the denominators in the quasi-diagonal algorithms do not vanish), so that the algorithms considered are well-defined. Suppose that the chosen interpretation ω of the output layer depends smoothly on the output layer activities.

Then there exists a value η_C of the learning rate such that, for any learning rate $\eta < \eta_C$, the value of the loss function strictly decreases after one step of the unitwise natural gradient, backpropagated metric gradient, or their quasi-diagonal reductions.

As usual, the value of η_C depends on the current state of the network and thus may change over the course of training.

3.2 Invariance properties

The algorithms presented in Section 1 are the implementation of the gradients and metrics defined in Section 2, written out using $[0; 1]$-valued activities and the logistic activation function. We could have written them out, for instance, using $[-1; 1]$-valued activities and the tanh activation function, and

the learning trajectory would be the same—provided, of course, that the initialization was done so that both implementations of the network behave the same at startup. We present a more precise formulation of this property.

Imagine that the inputs of the network are subjected to simple transformations such as scaling ($a_i \leftarrow \alpha_i a_i$ for i in the input layer) or 0/1 inversion ($a_i \leftarrow 1 - a_i$). There is a simple way to change the parameters of subsequent units so that the final activation of the network stays the same, namely, $w_{ij} \leftarrow w_{ij}/\alpha_i$ for scaling and $w_{ij} \leftarrow -w_{ij}$, $w_{0j} \leftarrow w_{0j} + w_{ij}$ for 0/1 inversion. So clearly the expressivity of a neural network is not sensitive to such changes.

However, training will behave differently. For instance, if we apply one step of backpropagation training to the scaled inputs with the scaled network, the coefficiens of units which have been scaled down ($\alpha_i < 1$) will evolve more slowly and conversely for $\alpha_i > 1$. The final output of the network after the update will be different. (Hence the common practice of rescaling the activities of units.) The same goes for 0/1 inversion in a slightly more complicated way: evolution of the bias depends on the activity of input units, and the weights from input units with activity close to 0 will evolve faster than those with activity close to 1, as seen on Figure 1.

We would like the following invariance for a training procedure: If we start with two networks N and N' which are fed inputs x and x' with x' obtained from a simple transformation of x, and if the parameters of N' are set such that initially its output is the same as N, then we would like the outputs of N and N' to stay the same after one step of the training procedure.

This is *not* satisfied by backpropagation. However, for any affine transform of the activities of any unit, this is satisfied by the natural gradient, unitwise natural gradient, backpropagated metric gradient, and their quasi-diagonal reductions.

The sigmoid and tanh networks correspond to each other by the following rewriting, thanks to $\tanh(x) = 2\,\mathrm{sigm}(2x) - 1$: if $a_k = \mathrm{sigm}(\sum_{i\to k} w_{ik} a_i + w_{0k})$ and $a'_k = \tanh(\sum_{i\to k} w'_{ik} a'_i + w'_{0k})$ (and interpretation of the output layer in the tanh case is done by putting back the activities in $[0;1]$ via $a' \mapsto 1/2 + a'/2$), then the two networks will behave the same if we set $w_{ik} = 4w'_{ik}$ ($i \neq 0$) and $w_{0k} = 2w'_{0k} - 2\sum_{i\neq 0} w'_{ik}$.

Definition 28.
Let k be an input or internal unit. Call (α, β, γ)-affine reparametrization of unit k *the following operation: Replace the activation of unit k*

$$a_k = f^k_{\theta_k}(a_{i_1}, \ldots, a_{i_{n_k}}) \tag{100}$$

where $\theta_k = (w_{0k}, (w_{ik})_{i\to k})$, with

$$a'_k = \alpha f^k_{\gamma\,\theta'_k}(a_{i_1}, \ldots, a_{i_{n_k}}) + \beta \tag{101}$$

where $\theta'_k = (w'_{0k}, (w'_{ik})_{i\to k})$. Send a'_k instead of a_k to the next layer of the network, with weights modified as follows:

$$w'_{kj} = w_{kj}/\alpha, \qquad w'_{0j} = w_{0j} - w_{kj}\beta/\alpha \tag{102}$$

for all units j such that $k \to j$, and $w'_{ik} = w_{ik}/\gamma$ for all units i with $i \to k$ (including $i = 0$), so that the final outputs before and after the reparametrization are the same.

The passage from sigm to tanh consists in applying the $(2, -1, 2)$-reparametrization to all units. We have restricted the definition to non-output units to simplify notation; for output units a corresponding reparametrization of the output interpretation has to be done.

The following is an immediate consequence of the intrinsic definition of the algorithms. It is only part of the invariance properties of the objects from Section 2; we limited ourselves to this case to simplify notation.

Proposition 29 (Invariance under affine reparametrization).
Consider a network obtained from an initial network by applying any number of (α, β, γ)-affine reparametrizations to any number of units (where α, β and γ may depend on the unit).

Then, after one step of the unitwise natural gradient, backpropagated metric gradient, unitwise TSD gradient, or their quasi-diagonal reductions, the final outputs of the non-reparametrized and reparametrized networks are the same.

Consequently, the learning trajectories, and performance, of the two networks with these corresponding initializations are the same.

This may look obvious, but we should keep in mind that this property is *not* satisfied by backpropagation or quasi-Hessian methods.

In particular, these algorithms are insensitive to shifting and scaling of all units in the network. Traditionally, it is recommended to normalize the activities on input units so that they average to 0 over the dataset and have a prescribed variance: the algorithms here automatically do the same in an implicit way, for all (not only input) units. As a consequence, units with low activation levels get updated as fast as highly activated units. (Note that as discussed after the definition of the quasi-diagonal algorithms, these averages and variances are computed according to non-uniform weights on the dataset given by the Fisher modulus or backpropagated modulus.)

Still the invariance above only applies if the two networks considered have corresponding initializations. For instance, if the initial weights are random with a variance set to 1 whatever the data, obviously the initial behavior of the network will be sensitive to scaling of its input. So *these methods do not remove the need for traditional recommendations for initializing the*

weights (either by normalizing the data and then taking weights of size 1, or by taking initial weights depending on the variance or covariance matrix of the data).

The unitwise natural gradient, backpropagated metric gradient, and unitwise TSD gradient (but not their quasi-diagonal reductions) have a further, more interesting invariance property: invariance under affine recombination of the signals a unit receives from its various incoming units. For instance, if we start with zero weights, an internal unit will evolve in the same way if it receives f and $f + \varepsilon g$ (where f and g are seen as function of the input x) as if it receives f and g. This is especially useful if g is correlated to the desired output.

Proposition 30 (Invariance under affine recombination of incoming signals).
Consider a neural network and define a new one in the following way. Let k be a non-input unit in the network, with n_k incoming units, and let $\varphi : \mathbb{R}^k \to \mathbb{R}^k$ be an invertible affine map. Define a new network by replacing the activation function at unit k

$$a_k = f^k_{\theta_k}(a_{i_1}, \dots, a_{i_{n_k}}) \tag{103}$$

with

$$a_k = f^k_{\varphi^*(\theta_k)}(\varphi(a_{i_1}, \dots, a_{i_{n_k}})) \tag{104}$$

still parametrized by θ_k, where $\varphi^*(\theta_k)$ results from applying the dual[6] *inverse affine transformation φ^* to θ_k, so that initially the responses of the original and reparametrized networks are the same.*

Then, after one step of the unitwise natural gradient, backpropagated metric gradient, or unitwise TSD gradient, with respect to θ_k, the final outputs of the non-reparametrized and reparametrized networks are the same.

Consequently, the learning trajectories, and performance, of the two networks with these corresponding initializations are the same.

Once more, this is not simply φ in one place cancelling out φ^{-1} in another: indeed, backpropagation or quasi-Hessian methods do not have this property, and neither do the quasi-diagonally-reduced algorithms.

Proof. This comes as a consequence of the best-fit interpretation (Proposition 31) below.

It also follows from the intrinsic constructions by noting that, unlike the quasi-diagonal reductions, the construction of these gradients never breaks

[6] θ_k is an affine form over the n_k-tuple of incoming activities. φ^* is defined, axiomatically, by the property that applying $\varphi^*(\theta_k)$ to the activities transformed by φ, is the same as applying θ_k to the untransformed activities. Decomposing $\theta_k = (w_{0k}, (w_{ik})_{i \to k})$, the affine matrix defining φ^* is the transpose of the inverse of the affine matrix defining φ.

down the n_k-tuple of incoming activities into its components from each incoming unit; thus, contrary to the quasi-diagonal reductions, we could have written the unitwise natural and backpropagated metrics in a setting where activation functions are given by $a_k = f^k_{\theta_k}(g(a_{i_1}, \ldots, a_{i_{n_k}}))$ where g is a fixed, parameterless map with values in a manifold. □

3.3 Best-fit interpretation

The unitwise natural gradient, backpropagated metric gradient, and unitwise TSD gradient (but not their quasi-diagonal reductions) share an interpretation as a least-squares regression problem at each unit. Namely, the backpropagated value $b_k(x, y)$ on input x and target y indicates how the activity of unit k should change on input x. Seeing b_k as a function of the input x, unit k has to use the activities of incoming units i (also seen as functions of x) and combine them using the weights w_{ik}, to match $b_k(x, y)$ as close as possible for each x. This idea is presented in [Kur94] in a more specific setting. This is also relevant to the behavior of the algorithms when the matrices F and M defining the metrics are singular or close to singular, as we will see.

Proposition 31 (Intrinsic gradients as best fit to b). *Let k be a non-input unit in the network. For x in the dataset $\mathcal{D}$, let $b_k(x)$ be the backpropagated value* (4) *obtained on input x and the corresponding target y.*

Consider the solution $\lambda = (\lambda_i)$ to the following weighted least-squares problem:

$$\lambda = \arg\min \left\{ \sum_{x \in \mathcal{D}} \left(\sum_i \lambda_i a_i(x) - \frac{b_k(x)}{a_k(x)(1 - a_k(x))\Phi_k(x)} \right)^2 W_x \right\} \qquad (105)$$

where i runs over the incoming units to k (including $i = 0$ with $a_0 \equiv 1$), $\Phi_k(x)$ is the Fisher modulus (Definition 3), and the weights are

$$W_x := a_k(x)^2 (1 - a_k(x))^2 \Phi_k(x) \qquad (106)$$

Then the unitwise natural gradient step (17) *is given by λ, namely, the update is $w_{ik} \leftarrow w_{ik} + \eta \lambda_i$ at each unit k.*

The same holds for the backpropagated metric gradient using the backpropagated modulus m_k (Definition 7) instead of the Fisher modulus Φ_k.

The same holds for the unitwise TSD gradient using $b_k(x)^2$ instead of the Fisher modulus Φ_k.

Thus, the gradient step depends on the *linear span* of the incoming activities $(a_i(x))_{i \to k}$, seen as functions over the dataset (which, by the way, proves Proposition 30 above). This is why the gradient step is the same whether

the unit receives signals $f(x)$ and $g(x)$ or $f(x)$ and $f(x)+\varepsilon g(x)$. Thus, these algorithms perform an implicit orthonormalization of the incoming signals at each unit (not only input units) in the network.

Proof. A direct application of the well-known formula for the solution of the weighted least-squares problem (105), with the choice of weight (106), yields exactly the updates (16) and (26). □

Non-invertibility and regularization of the matrices. In several situations the matrices F and M used to define the unitwise natural update and backpropagated metric update can be singular.

This is the case, for instance, if one input unit is uniformly set to 0 over all elements in the dataset: obviously such a unit is not informative, and the corresponding term will vanish both in the metric and in the gradient. This is also the case when, e.g., two units incoming to the same unit are perfectly correlated. Correlation in the activation profiles happens systematically in case the size of the dataset is smaller than the number of incoming parameters at a given unit.

The linear regression viewpoint limits, in theory, the seriousness of these issues: this only means the linear regression problem has several solutions (one can add any quantity to a non-informative weight), and any of them will do as an update. Indeed, for instance, the matrix M in Definition 9 is of the form $AA^\top$, and M^{-1} is applied to the vector G which is of the form AY, thus G always lies in the image of A and thus the linear system is underdetermined, not overdetermined. From the gradient ascent viewpoint this means the matrix M will be singular but the gradient term $\partial L/\partial w$ will vanish in the corresponding directions.

Numerically, however, the issue must be dealt with. One can use the Moore–Penrose pseudoinverse of M (or F in the Fisher matrix case), obtained by adding $\varepsilon.\mathrm{Id}$ to M or to F with very small ε. This is a standard regularization technique. It has the advantage of producing a well-defined update when $\varepsilon \to 0$, asymptotically independent of ε.

Thus, if a formal definition is needed, one can decide to use the Moore–Penrose pseudoinverse for M^{-1} and F^{-1} in the definition of the updates. However, this formally breaks the invariance properties: the Moore–Penrose pseudoinverse selects, among the several possible solutions (λ_i) to an underdetermined least squares problem, the one with smallest norm $\sum \lambda_i^2$, and this is not intrinsic.

4 An experimental comparison

To test the influence of the different methods, we chose a very simple problem in which a perfect solution is expected to be found. A sparsely connected

network with 5 layers of size 100, 30, 10, 30, and 100 was built, and 16 random length-100 binary strings were fed to the input layer, with the target equal to the input. Ideally the network learns to encode each of the 16 samples using 4 bits on the middle layer (thus with room to spare) and uses the bottom layer parameters to rewrite the output from this.

The sparsely connected network is built at random in each instance as follows. Each of the 100 units in the input layer is linked to 5 randomly selected nodes in the first hidden layer. Each of the 30 units in the first hidden layer is linked to 5 random nodes in the middle hidden layer. The scheme is reversed for the bottom part of the model: each of the 100 output units is linked to 5 random nodes in the last hidden layer, and each unit in the last hidden layer is linked to 5 random nodes of the middle hidden layer.

For each instance, the dataset is made of 16 random binary strings of length 100. The target for each input is identical to the input. We use Bernoulli interpretation of the output.

Note that this setting is adverse for the unitwise and quasi-diagonal natural gradients, which require a small output layer; this must be remembered in the comparisons below.

To test the influence of parametrization on non-invariant algorithms, and to check invariance of the invariant ones, each algorithm was implemented both using $\mathrm{sigm}(x)$ and $\tanh(x)$ as the activation function.

The methods tested are: backpropagation; unitwise natural gradient; quasi-diagonal natural gradient; unitwise TSD gradient (i.e., "one-sample" natural gradient, see Section 2.5); backpropagated metric gradient; quasi-diagonal backpropagated metric gradient; diagonal Gauss–Newton (equivalent to keeping only the diagonal terms in the quasi-diagonal backpropagated metric gradient).

Since the sample size is small, the algorithms were run in batch mode.

Regularization. The algorithms were taken directly from Section 1. To all methods except backpropagation, we added a regularization term of $10^{-4}\,\mathrm{Id}$ to the various matrices involved, to stabilize numerical inversion. This value is not so small; values such as 10^{-7} seemed to affect performance. This is probably due to the small sample size (16 samples): each sample contributes a rank-1 matrix to the various metrics. Larger sample sizes would probably need less regularization.

Initialization. For the tanh activation function, all the weights were initialized to a centered Gaussian random variable of variance 1, and the biases set to 0. For the sigmoid activation function, the initialization was the corresponding one (using Eqs. 41 and 42) so that initially the responses of the networks are the same: namely, each weight was set to a centered Gaussian random variable of variance 4, and then the bias at unit k was set to

$-\sum_{i\, i\to k} w_{ik}/2$. This initialization has the property that if the incoming signals to a unit are independent, centered about 1/2 (sigmoid) or 0 (tanh) and of variance σ with σ not too large, then the output of the unit is also centered of variance $\approx \sigma$. (The factor 4 in the sigmoid case compensates for the derivative 1/4 of the sigmoid function at 0.)

Learning rate. A simple adaptive method was used for the learning rate. All methods based on gradients in a metric have a guarantee of improvement at each step if the learning rate is small enough. So in the implementation, if a step was found to make the loss function worse (in a batch mode, thus summed over all samples), the step was cancelled and the learning rate was divided by 2. If the step improves the loss function, the learning rate is increased by a factor 1.1. The initial learning rate was set to 0.01; in practice the initial value of the learning rate is quickly forgotten and has little influence.

Unfortunately this scheme only makes sense in batch mode, but it has the advantage of automatically selecting learning rates that suit each method, thus placing all methods on an equal footing.

Execution time and number of iterations. First, $10,000$ steps of backpropagation were performed on the whole dataset, in batch mode. The resulting running time was set aside and converted to an equivalent number of iterations for all of the other algorithms. This is a very rough way to proceed, since the running times can depend on the implementation details[7], and vary from run to run (because floating point operations do not take the same time depending on the numbers they are operating on, especially when both very small and very large values are involved.

Most of all, the different methods scale in different ways with the network, and so the network used here may not be representative of other situations. In particular this auto-encoder setting with 100 output units puts the unit-wise natural gradient and quasi-diagonal natural gradient at a disadvantage (since on the same time budget they must performed n_{out} backpropagations per sample), compared to, e.g., a classification setting.

Nevertheless, we give in Table 4 the numbers of iterations giving roughly equal running time for each method.

The natural gradient was also tested (using the exact full Fisher matrix as obtained from Proposition 26). The computational cost is very high and only 10 iterations take place in the alloted time, too few for convergence. Thus we do not report the associated results.

Results. In Table 2, we report the average loss per sample, in bits, and its standard deviation, at the end of the allocated number of training iterations.

[7] We tried to implement each method equally carefully.

Method	Number of iterations
Backpropagation (sigmoid)	10,000
Backpropagation (tanh)	10,000
Unitwise natural gradient	2,100 to 2,300
Quasi-diagonal natural gradient	2,800 to 3,100
Backpropagated metric	4,200 to 4,300
Quasi-diagonal backpropagated metric	7,400 to 7,500
Diagonal Gauss–Newton	7,700 to 7,800
Unitwise TSD gradient	4,000 to 4,100

Table 1: Number of iterations resulting in approximately equal execution times for our problem

Method	Average loss (bits) $\pm$ std-dev	
	sigmoid	tanh
Backpropagation	35.9 ± 2.1	24.7 ± 2.2
Unitwise natural gradient	0.9 ± 1	1.4 ± 1.8
Quasi-diagonal natural gradient	3.5 ± 1.2	3.4 ± 1.6
Backpropagated metric	0.8 ± 0.8	0.3 ± 0.5
Quasi-diagonal backpropagated metric	1.9 ± 1.2	1.5 ± 1.3
Diagonal Gauss–Newton	11.6 ± 2.5	3.5 ± 2.0
Unitwise TSD gradient	24.7 ± 2.5	28.5 ± 3.4

Table 2: Average loss per sample (bits) after an execution time equivalent to $10,000$ backpropagation passes, computed over 20 independent runs

These values can be interpreted as representing the average number of bits of an output that the model did not learn to predict correctly (out of 100). The results of the implementations using sigmoid and tanh activation are reported separately.

Performance as a function of time is plotted in Figure 3.

The statistics were made using 20 independent runs for each method.

Interpretation. These results are mainly illustrative: the situation considered here may not be representative because of the small sample size and network size involved.

Still, it is clear that for problems of this size, the more elaborate algorithms are very competitive. Only the tanh implementation of the diagonal Gauss–Newton method comes close to the invariant algorithms (while its performance in sigmoid implementation is not as good).

As can be expected, the invariant algorithms have similar performance in sigmoid or tanh implementation: trajectories match each other closely

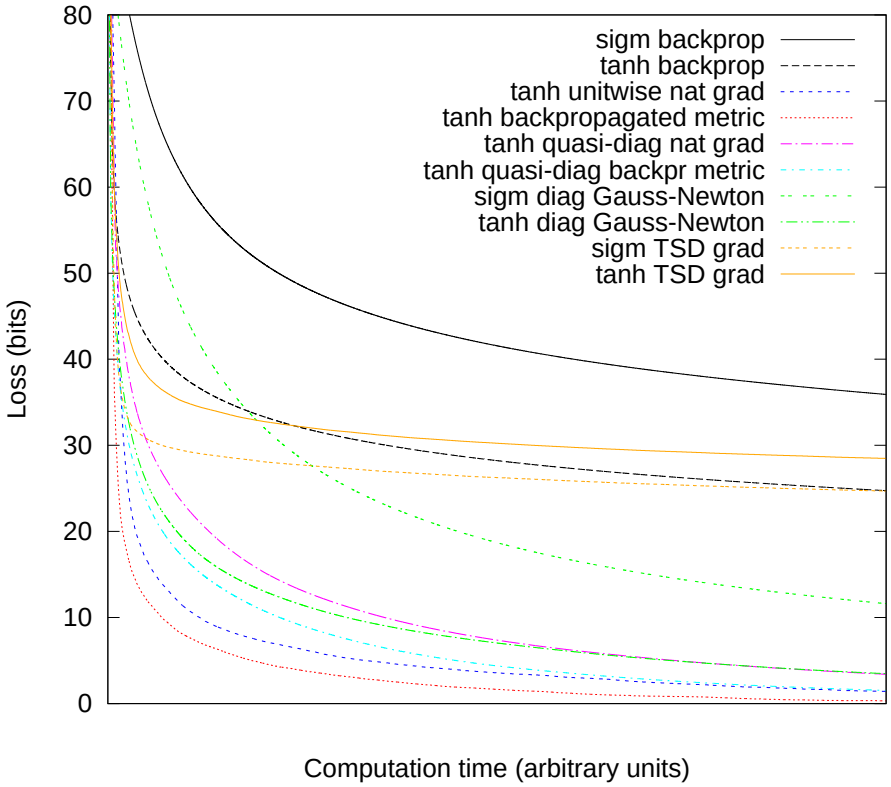

Figure 3: Performance over time of all algorithms involved. For better readability the trajectories of the invariant algorithms have been plotted only in tanh implementation (Figure 2 shows them in sigmoid implementation for completeness).

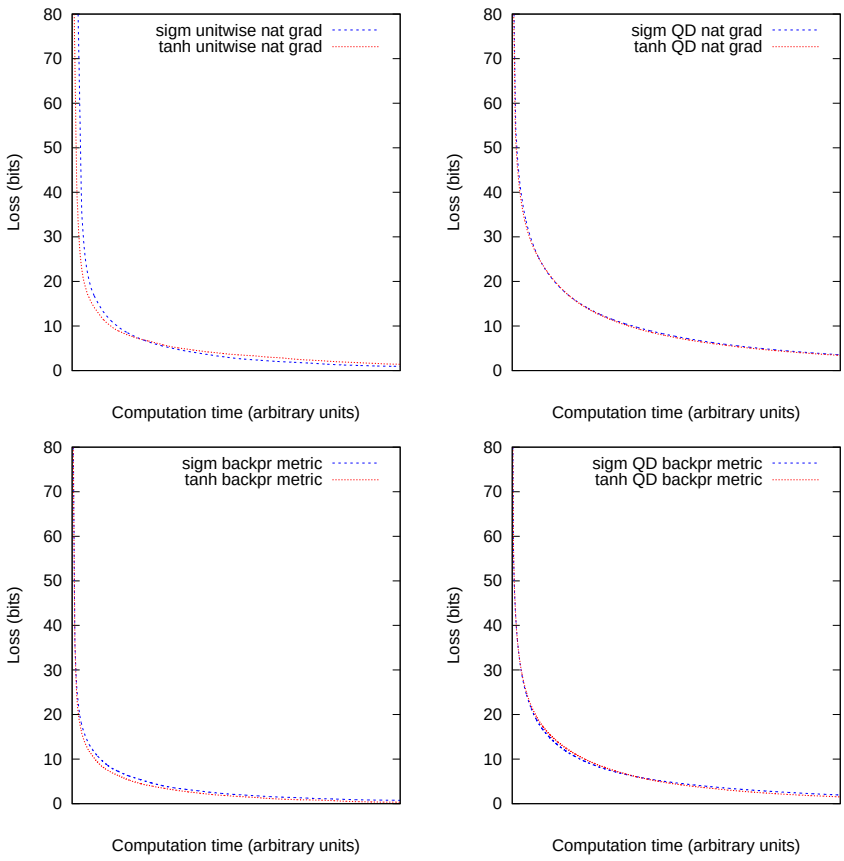

Figure 4: Comparison of the trajectories of the invariant algorithms in tanh and sigmoid implementations

(Figure 4). The variations are caused, first, by random initialization of the dataset and weights in each run, and second, by the inclusion of the regularization terms $\varepsilon \,\mathrm{Id}$, which breaks invariance. If the effect of the latter is isolated, by having the same initialization in tanh and sigmoid implementations, the trajectories coincide for a the first few iterations but then start to differ; however, overall performance is not affected.

In this setting, the natural gradient, in its unitwise and quasi-diagonal versions, seems to perform slightly worse than the backpropagated metric methods. Remember, however, that this is an effect of the large output layer size combined with a given computation time budget. Per iteration instead of computation time, the natural gradient methods perform better than the backpropagated metric methods, and we expect them to be more competitive for smaller output sizes, e.g., in classification tasks.

The relatively bad performance of the unitwise TSD gradient is a surprise.

This method is invariant, although not naturally interpreted as an intrinsic Riemannian metric, and is meant to approximate the natural gradient. In the related context of recurrent networks [Oll13] it was found to perform quite well. Analysis of the runs shows that the learning rates (adjusted as described above) decrease steadily through optimization. This may be related to poor invertibility of the matrices involved. An indication in this direction comes from the difference in performance in the sigmoid and tanh implementation: this algorithm is invariant and the only difference can come from non-invariant regularization kicking in. The small size of the dataset in our setting is not enough to explain this problem, as it does not seem to affect the other methods. Too low rank of the matrices may be related to the large dimensionality of the output layer compared to the number of samples in our experiment (in contrast to [RMB07] or [Oll13]): Indeed, reasoning on the full (whole-network) Fisher matrix, the TSD gradient contributes a matrix of rank 1 for each data sample (see (95)); on the other hand, the exact Fisher matrix contributes a sum of n_{out} matrices of rank 1 for each data sample as can be seen from (96)–(97). Thus from a theoretical viewpoint the quality of the TSD approximation likely depends on output dimensionality.

Lack of invariance is striking for some algorithms, such as the diagonal Gauss–Newton method, the performance of which is very different in the sigmoid and tanh interpretations (Figure 5). The quasi-diagonal backpropagated metric method only differs from diagonal Gauss–Newton by the inclusion of a small number of non-diagonal terms in the update. This change brings the sigmoid and tanh implementations in line with each other, improving performance with respect to the best of the two Gauss–Newton implementations. In settings where the activities of units (especially internal units, since the input can always be centered) are not as well centered as here, we expect the quasi-diagonal backpropagated metric method to outperform the tanh diagonal Gauss–Newton implementation even more clearly. So, arguably, the quasi-diagonal backpropagated metric is "the invariant way" to write the diagonal Gauss–Newton algorithm.

In conclusion, although this experiment is a small-scale one, it clearly emphasizes the interest of using invariant algorithms.

Appendix

Proof of Proposition 26

The Fisher metric depends, of course, on the interpretation of the output layer as a probability distribution ω. The final output ω is a probability distribution over the values of y in the target space, parametrized by the activities a_k of the output units k. If the output activities change by δa_k, the probability distribution ω will change as well. The norm of this change

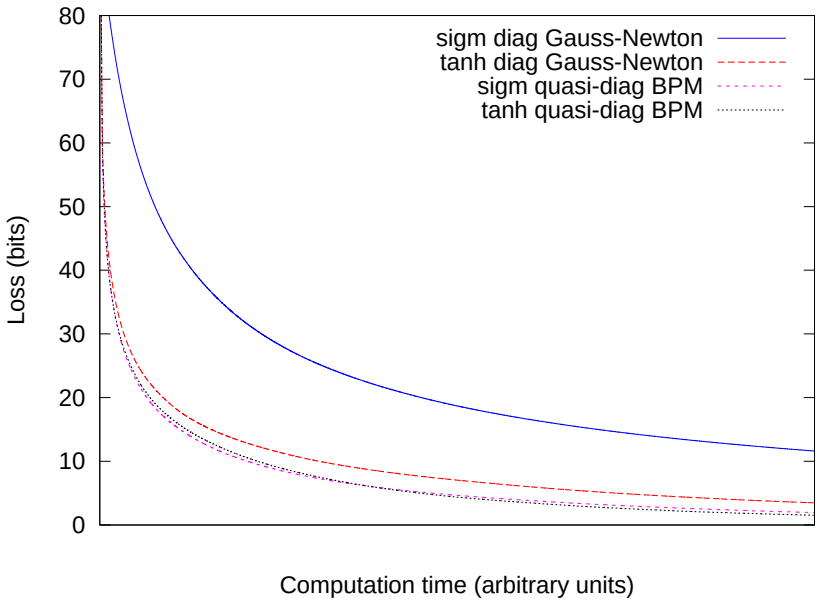

Figure 5: Effect of introducing a few non-diagonal terms: Comparison of the diagonal Gauss–Newton and the quasi-diagonal backpropagated metric methods

in Fisher metric is

$$\|\delta\omega\|^2_{\text{nat}} = \mathbb{E}_{y\sim\omega}(\delta \ln \omega(y))^2 \tag{107}$$

$$= \sum_{k,k'\in\mathcal{L}_{\text{out}}} \mathbb{E}_{y\sim\omega} \frac{\partial \ln \omega(y)}{\partial a_k} \frac{\partial \ln \omega(y)}{\partial a_{k'}} \delta a_k \delta a_{k'} \tag{108}$$

thus stemming from the matrix

$$I_{kk'} := \mathbb{E}_{y\sim\omega} \frac{\partial \ln \omega(y)}{\partial a_k} \frac{\partial \ln \omega(y)}{\partial a_{k'}} \tag{109}$$

over the output layer.

In the Bernoulli interpretation, for each component k of the output layer, the random variable y_k is a Bernoulli variable with parameter a_k. In the square-loss interpretation, each y_k is a Gaussian random variable with mean a_k and variance 1. In the classification interpretation, y is a discrete random variable which takes value k with probability $\omega_k = a_k^2/(\sum_{j\in\mathcal{L}_{\text{out}}} a_j^2)$.

Let us compute the Fisher metric in the space ω in each case. In the Bernoulli case, we have $\omega(y) = \prod_{k\in\mathcal{L}_{\text{out}}}(\mathbb{1}_{y_k=1}a_k + \mathbb{1}_{y_k=0}(1-a_k))$. Consequently

$$\frac{\partial \ln \omega(y)}{\partial a_k} = \frac{\mathbb{1}_{y_k=1}}{a_k} - \frac{\mathbb{1}_{y_k=0}}{1-a_k} = \frac{y_i - a_k}{a_k(1-a_k)} \tag{110}$$

Since under the distribution ω we have $\mathbb{E}y_k = a_k$ and $\operatorname{Var} y_k = a_k(1-a_k)$ (with y_k and y_j independent for $k \neq j$) we find

$$I_{kk'} = \frac{\mathbb{1}_{k=k'}}{a_k(1-a_k)} \tag{111}$$

for k and k' in the output layer.

In the Gaussian case we have $\omega(y) = \prod_k \frac{e^{-(y_i-\omega_k)^2/2}}{\sqrt{2\pi}}$ so that $\frac{\partial \ln \omega(y)}{\partial \omega_k} = y_i - \omega_k$. Since under the distribution ω we have $\mathbb{E}y_k = \omega_k$ and $\operatorname{Var} y_k = 1$ we find $I = \mathrm{Id}$ hence

$$I_{kk'} = \mathbb{1}_{k=k'} \tag{112}$$

for k and k' in the output layer.

In the classification case the probability to have $y = j$ is $\omega(y) = a_j^2/S$ with $S = \sum_{i \in \mathcal{L}_{\text{out}}} a_i^2$. Thus $\partial \ln \omega(y)/\partial \omega_k = 2(\frac{\mathbb{1}_{k=j}}{a_j} - \frac{a_k}{S})$. Taking the expectation over y we find

$$I_{kk'} = 4 \sum_j \frac{a_j^2}{S} \left(\frac{\mathbb{1}_{j=k}}{a_j} - \frac{a_k}{S} \right) \left(\frac{\mathbb{1}_{j=k'}}{a_j} - \frac{a_{k'}}{S} \right) \tag{113}$$

$$= 4 \frac{a_k^2}{S} \frac{1}{a_k} \frac{1}{a_{k'}} \mathbb{1}_{k=k'} - 4 \frac{a_k^2}{S} \frac{1}{a_k} \frac{a_{k'}}{S} - 4 \frac{a_{k'}^2}{S} \frac{1}{a_{k'}} \frac{a_k}{S} + 4 \left(\sum_j \frac{a_j^2}{S} \right) \frac{a_k}{S} \frac{a_{k'}}{S} \tag{114}$$

$$= \frac{4}{S} \mathbb{1}_{k=k'} - \frac{4 a_k a_{k'}}{S^2} \tag{115}$$

These give the expression of the Fisher matrix $I_{kk'}$ for k and k' in the output layer. This is enough to compute the full Fisher matrix, as follows.

Let x be an input for the network. Given a variation $\delta\theta$ of the network parameters θ, let δa_i be the resulting variation of unit i, and let $\delta\omega$ be the resulting variation of the final output ω. We have $\delta\omega = \sum_{k \in \mathcal{L}_{\text{out}}} \frac{\partial \omega}{\partial a_k} \delta a_k$. The datum-wise Fisher metric on θ is

$$\|\delta\theta\|_{\text{nat}}^2 = \|\delta\omega\|_{\text{nat}}^2 \tag{116}$$

$$= \sum_{k,k' \in \mathcal{L}_{\text{out}}} I_{kk'} \delta a_k \delta a_{k'} \tag{117}$$

For each k in the output layer, we have $\delta a_k = \sum_i \frac{\partial a_k}{\partial \theta_i} \delta\theta_i$ where the sum runs over all units i in the network. For each i we have $\frac{\partial a_k}{\partial \theta_i} = \frac{\partial a_k}{\partial a_i} \frac{\partial a_i}{\partial \theta_i} = J_i^k \frac{\partial a_i}{\partial \theta_i}$. Plugging this into the above yields

$$\|\delta\theta\|_{\text{nat}}^2 = \sum_i \sum_{i'} \sum_{k \in \mathcal{L}_{\text{out}}} \sum_{k' \in \mathcal{L}_{\text{out}}} I_{kk'} J_i^k J_{i'}^{k'} \frac{\partial a_i}{\partial \theta_i} \frac{\partial a_{i'}}{\partial \theta_{i'}} \tag{118}$$

so that the term of the Fisher matrix corresponding to $\delta\theta_i$ and $\delta\theta_{i'}$ is $\sum_{k \in \mathcal{L}_{\text{out}}} \sum_{k' \in \mathcal{L}_{\text{out}}} I_{kk'} J_i^k J_{i'}^{k'} \frac{\partial a_i}{\partial \theta_i} \frac{\partial a_{i'}}{\partial \theta_{i'}}$.

For standard neural networks we have $\delta\theta_i = (\delta w_{ji})_{j,\, j\to i}$ and the sigmoid activation function yields $\frac{\partial a_i}{\partial w_{ji}} = a_j a_i(1-a_i)$.

Plugging into this the expression for $I_{kk'}$ yields the results in Proposition 26.

Proof of Remark 15

Let v be an infinitesimal variation of the parameter θ. Let v_i be the coordinates of v in some coordinate system. At first order, the increment in the average loss function along v is $\mathbb{E}_{x\in\mathcal{D}} \sum_i \frac{\partial \ell_x}{\partial \theta_i} v_i$.

Let us abbreviate $\partial_i \ell_x = \frac{\partial \ell_x}{\partial \theta_i}$. The matrix defining the TSD metric is $M_{ij} = \mathbb{E}_{x\in\mathcal{D}} \partial_i \ell_x \partial_j \ell_x$. The corresponding gradient direction is $M^{-1}\mathbb{E}_{x\in\mathcal{D}} \partial \ell_x$.

Let $m = \mathbb{E}_{x\in\mathcal{D}} \sum_i \partial_i \ell_x v_i$ be the change in loss function associated with v. The variance, when x runs over the dataset, of the gain in the loss function for x is $\mathbb{E}_{x\in\mathcal{D}} \left((\sum_i \partial_i \ell_x v_i) - m\right)^2 = \mathbb{E}_{x\in\mathcal{D}} (\sum_i \partial_i \ell_x v_i)^2 - m^2$. For fixed average gain m, this is minimal when $\mathbb{E}_{x\in\mathcal{D}} (\sum_i \partial_i \ell_x v_i)^2$ is minimal.

This is a smooth convex function of v, whose minimum we have to find over the hyperplane $\{v,\, \mathbb{E}_{x\in\mathcal{D}} \sum_i \partial_i \ell_x v_i = m\}$. The minimum of a positive-definite quadratic functional $\sum_{ij} A_{ij} v_i v_j$ over a hyperplane $\sum_i B_i v_i = m$, is found at $v = \lambda A^{-1} B$ for some constant λ. Here we have $B_i = \mathbb{E}_{x\in\mathcal{D}} \partial_i \ell_x$, and expanding $\mathbb{E}_{x\in\mathcal{D}} (\sum_i \partial_i \ell_x v_i)^2 = \mathbb{E}_{x\in\mathcal{D}} ((\sum_i \partial_i \ell_x v_i)(\sum_j \partial_j \ell_x v_j)) = \sum_{ij} \mathbb{E}_{x\in\mathcal{D}} \partial_i \ell_x \partial_j \ell_x v_i v_j$ yields $A_{ij} = \mathbb{E}_{x\in\mathcal{D}} \partial_i \ell_x \partial_j \ell_x = M_{ij}$. Consequently, for any value of m, the optimal v is a multiple of the TSD gradient direction $M^{-1}\mathbb{E}_{x\in\mathcal{D}} \partial \ell_x$.

References

[AAHO11] Ludovic Arnold, Anne Auger, Nikolaus Hansen, and Yann Ollivier. Information-Geometric Optimization algorithms: A unifying picture via invariance principles. Preprint, arXiv:1106.3708v2, 2011.

[AGS05] Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows in metric spaces and in the space of probability measures*. Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 2005.

[Ama98] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Comput.*, 10:251–276, February 1998.

[AN00] Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 2000. Translated from the 1993 Japanese original by Daishi Harada.

[APF00] Shun-ichi Amari, Hyeyoung Park, and Kenji Fukumizu. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12(6):1399–1409, 2000.

[GHL87] S. Gallot, D. Hulin, and J. Lafontaine. *Riemannian geometry*. Universitext. Springer-Verlag, Berlin, 1987.

[Kur94] Takio Kurita. Iterative weighted least squares algorithms for neural networks classifiers. *New Generation Comput.*, 12(4):375–394, 1994.

[LBOM96] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer, 1996.

[Oll13] Yann Ollivier. Efficient Riemannian training of recurrent neural networks for learning symbolic data sequences. Preprint, arXiv:1306.0514, 2013.

[PB13] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. Preprint, http://arxiv.org/abs/1301.3584, 2013.

[RMB07] Nicolas Le Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *NIPS*. Curran Associates, Inc., 2007.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.