Coordinate Descent on the Orthogonal Group for Recurrent Neural Network Training

Estelle Massart,[1, 2] **Vinayak Abrol,** [3]
[1] Mathematical Institute, University of Oxford, Oxford, UK
[2] National Physical Laboratory, Teddington, UK
[3] Infosys Centre for AI, IIIT Delhi, India
massart@maths.ox.ac.uk, abrol@iiitd.ac.in

Abstract

We address the poor scalability of learning algorithms for orthogonal recurrent neural networks via the use of stochastic coordinate descent on the orthogonal group, leading to a cost per iteration that increases linearly with the number of recurrent states. This contrasts with the cubic dependency of typical feasible algorithms such as stochastic Riemannian gradient descent, which prohibits the use of big network architectures. Coordinate descent rotates successively two columns of the recurrent matrix. When the coordinate (i.e., indices of rotated columns) is selected uniformly at random at each iteration, we prove convergence of the algorithm under standard assumptions on the loss function, stepsize and minibatch noise. In addition, we numerically show that the Riemannian gradient has an approximately sparse structure. Leveraging this observation, we propose a variant of our proposed algorithm that relies on the Gauss-Southwell coordinate selection rule. Experiments on a benchmark recurrent neural network training problem show that the proposed approach is a very promising step towards the training of orthogonal recurrent neural networks with big architectures.

1 Introduction

Exploding or vanishing gradients are key issues affecting the training of deep neural networks (DNNs), and are particularly problematic when training recurrent neural networks (RNNs) (Bengio, Simard, and Frasconi 1994; Pascanu, Mikolov, and Bengio 2013), an architecture that endows the network with some memory and has been proposed for modeling sequential data (see, e.g., (Giles, Kuhn, and Williams 1994)). In recurrent neural networks, the signal propagation is described by the following pair of equations

$$\left\{\begin{array}{lll} h(t+1) & = & \phi(W_{\text{in}}x(t+1) + Wh(t)), \\ y(t+1) & = & W_{\text{out}}h(t+1) + b_{\text{out}}, \end{array}\right. \tag{1}$$

for $t = 0, 1, \dots,$ with input $x(t) \in \mathbb{R}^{d_{\text{in}}}$, hidden state $h(t) \in \mathbb{R}^d$, and output $y(t) \in \mathbb{R}^{d_{\text{out}}}$. The mapping ϕ is a pointwise nonlinearity, and $W \in \mathbb{R}^{d\times d}$, $W_{\text{in}} \in \mathbb{R}^{d\times d_{\text{in}}}$, $W_{\text{out}} \in \mathbb{R}^{d_{\text{out}}\times d}$ and $b_{\text{out}} \in \mathbb{R}^{d_{\text{out}}}$ are model parameters. The repeated multiplication of the hidden state by the recurrent matrix W makes these architectures particularly sensitive to exploding or vanishing gradients.

A recently proposed remedy against exploding and vanishing gradients in RNNs imposes the recurrent weight matrix W to be orthogonal/unitary, see, e.g., (Arjovsky, Shah, and Bengio 2016). However, for large models, enforcing this constraint comes with substantial computational costs, that prohibit the use of large architectures. Several solutions have been proposed to alleviate these costs, typically relying on some parametrization of the orthogonal group, see below. Unfortunately, most of these works are focused on algorithmic contributions and contain no convergence results.

As the orthogonal group admits a Riemannian manifold structure, the problem can be studied through the lens of Riemannian optimization. For example, stochastic Riemannian gradient descent has been applied to orthogonal RNN training in (Wisdom et al. 2016). This algorithm scales poorly with architecture size; its cost per iteration evolves cubically with the number of recurrent states. We propose here a stochastic extension of the Riemannian coordinate descent algorithm proposed in (Shalit and Chechik 2014). For each mini-batch, instead of updating the full matrix W, we rotate a pair of columns of W only; this is equivalent to restricting the update of the matrix W to one coordinate of the tangent space to the orthogonal group per iteration, for a suitably chosen basis. As pointed out in (Shalit and Chechik 2014), this rotation can be efficiently implemented as a multiplication of the current iterate by a Givens matrix, at a cost that evolves linearly with d. Moreover, as we only require a subset of the partial derivatives at each iterations, the cost of the backpropagation routine could be theoretically reduced from a quadratic to a linear function of d. Our work is thus a major step towards the training of high-dimensional orthogonal recurrent neural networks. Moreover, the theory of Riemannian optimization allows us to derive a convergence analysis of our proposed algorithm.

Related works

Orthogonal/unitary RNNs. Unitary RNNs have been initially proposed in (Arjovsky, Shah, and Bengio 2016) to avoid exploding or vanishing gradients. Various algorithms have then been developed to train orthogonal RNNs. Typically, these works propose parametrizations of the orthogonal/unitary group that lead to computationally cheap operations, see (Arjovsky, Shah, and Bengio 2016; Wisdom et al. 2016; Jing et al. 2017; Hyland and Rätsch 2017; Mhammedi et al. 2017; Helfrich, Willmott, and Ye 2018; Lezcano-Casado and Martínez-Rubio 2019; Maduranga, Helfrich, and Ye 2019). However, most of these works are focused on algorithmic contributions and do not contain convergence analyses. A notable exception is the work (Lezcano-Casado and Martínez-Rubio 2019); their parametrization of the orthogonal group relies on its Lie group structure and is shown not to create any spurious minimizer nor critical points under some weak assumption on the training trajectory. In that case, their algorithm can be seen as a stochastic Riemannian gradient descent on the orthogonal group with a specific Riemannian metric different from the canonical one, and its convergence follows from the analysis of stochastic Riemannian gradient descent, see (Bonnabel 2013). Let us finally mention two recent toolboxes extending the PyTorch class *torch.optim* to parameters constrained to lie on some manifolds, including the orthogonal group: McTorch (Meghwanshi et al. 2018) and GeoTorch (Lezcano-Casado 2019).

Orthogonal constraints in other DNN architectures. Orthogonal weights have also been used for other network architectures, including fully-connected, convolutional and residual neural networks (Ozay and Okatani 2016; Huang et al. 2018, 2020; Bansal, Chen, and Wang 2018; Jia et al. 2020). Using orthogonal weight matrices in DNNs preserves the energy of the signal propagated through the hidden units, and has been shown numerically to result in a lower test error than comparable unconstrained DNNs (Huang et al. 2018; Jia et al. 2020). Orthogonal initialization moreover leads to dynamical isometry (Pennington, Schoenholz, and Ganguli 2017), the desirable regime in which the spectrum of the input-output Jacobian is concentrated around the value one, shown to result in a faster training and better generalization, see (Pennington, Schoenholz, and Ganguli 2017; Hu, Xiao, and Pennington 2020; Murray, Abrol, and Tanner 2021; Abrol and Tanner 2020). For arbitrary DNN architectures, note however that orthogonal regularization is sometimes preferred over strict orthogonal constraints (see, e.g., (Jia et al. 2017; Yoshida and Miyato 2017; Bansal, Chen, and Wang 2018)).

Coordinate descent algorithms In unconstrained optimization, coordinate descent (CD) is a well-studied algorithm that minimizes successively the loss along a coordinate, typically using a coordinate-wise variant of a first-order method, and has been shown to achieve state-of-the-art results on a range of high-dimensional problems (Wright 2015). In particular, CD algorithms have recently been applied to DNN training, see, e.g., (Zeng et al. 2019; Palagi and Seccia 2019) and reference therein. When the objective is subject to orthogonality constraints, the Riemannian counterpart of CD, Riemannian coordinate descent (RCD), has been applied in (Ishteva, Absil, and Dooren 2013) to low-rank tensor approximation and in (Shalit and Chechik 2014) to sparse PCA and tensor decomposition. In particular, the latter provides an efficient implementation of coordinate updates in terms of multiplications by Givens matrices that we are extensively using in this work. Note however that these two papers consider the situation where the loss is exactly minimized along the coordinate direction at each iteration, an operation which is out of reach in our setting. This assumption has been recently relaxed in (Gutman and Ho-Nguyen 2021), where convergence results are provided for RCD with fixed stepsizes. To our knowledge, no work has explored the convergence of RCD with noisy gradients yet.

Contributions

We propose a stochastic Riemannian coordinate descent (SRCD) algorithm for training orthogonal RNNs. Our proposed algorithm is both scalable with the number of recurrent units (with cost per iteration scaling linearly with d), and comes with convergence guarantees unlike most existing works. More precisely, we prove the convergence of SRCD under standard assumptions on the (mini-batch) gradient noise, stepsize and objective, for coordinates selected uniformly at random at each iteration. Let us briefly mention that our algorithm is related to the one presented in (Jing et al. 2017); their work is mostly focused on algorithmic contributions, and does not highlight the link between their proposed algorithm and Riemannian coordinate descent nor discuss its convergence.

As a second contribution, we show numerically that the Riemannian gradient of the loss (with respect to the orthogonal parameter) has an approximately sparse representation in the basis considered for the tangent space, and propose a variant of SRCD in which the coordinate is selected using the Gauss-Southwell rule at each iteration. We finally illustrate numerically the behavior of the proposed algorithms on a benchmark problem. Implementation of the proposed algorithms can be found online[1].

2 Stochastic Riemannian coordinate descent on the orthogonal group

In this paper, without loss of generality, we address the optimization problem

$$\min_{X\in\mathbb{R}^{m\times n}, W\in\mathcal{O}_d} f(X,W), \tag{P}$$

where $\mathcal{O}_d := \{W \in \mathbb{R}^{d\times d} : W^\top W = I_d\}$ is the set of $d\times d$ orthogonal matrices. For the specific framework of RNN training, the variable X refers to the parameters W_{in}, W_{out}, b_{out} or any other unconstrained model parameter, while W is the recurrent matrix. We assume throughout the paper that the dimension of W is very large. Before introducing our algorithm, let us first summarize properties of the set of orthogonal matrices that we use in the rest of the paper.

Geometry of the manifold of orthogonal matrices

The set $\mathcal{O}_d$ of orthogonal matrices is a Riemannian manifold. Roughly speaking, a manifold is a topological set in which each neighbourhood can be set in one-to-one correspondence with an open set of the Euclidean space; in that sense, manifolds are sets that "locally look flat" (see (Absil, Mahony, and Sepulchre 2008) or (Boumal 2020) for a

[1] https://github.com/EMassart/OrthCDforRNNs

more formal definition). Typically, computations on manifolds are mostly done on the *tangent space* of the manifold at a given point, a first-order approximation of the manifold around that point. For the orthogonal group, the tangent space $\mathcal{T}_W\mathcal{O}_d$ at some point $W \in \mathcal{O}_d$ is given by:

$$\mathcal{T}_W\mathcal{O}_d = \{W\Omega : \Omega \in \mathbb{R}^{d\times d}, \Omega = -\Omega^\top\}. \tag{2}$$

This is a D-dimensional vector space, with $D = d(d-1)/2$; consistently, $\mathcal{O}_d$ is a D-dimensional manifold. Riemannian manifolds are manifolds whose tangent spaces are endowed with a Riemannian metric (a smoothly varying inner product, providing tangent vectors with a notion of length). As often for the manifold of orthogonal matrices (Absil, Mahony, and Sepulchre 2008), we choose as Riemannian metric the classical Euclidean metric: for each tangent space $\mathcal{T}_W\mathcal{O}_d$, and for all $\xi_W, \zeta_W \in \mathcal{T}_W\mathcal{O}_d$, the Riemannian metric between ξ_W and ζ_W is given by:

$$\langle \xi_W, \zeta_W \rangle := \operatorname{tr}\left(\xi_W^\top \zeta_W\right), \tag{3}$$

the usual Frobenius inner product. In this paper, we therefore use the notation $\langle\cdot,\cdot\rangle$ to refer both to the Riemannian metric, and the Euclidean inner product. Considering this metric, it can easily be checked that the tangent space $\mathcal{T}_W\mathcal{O}_d$ is generated by the orthonormal basis (Shalit and Chechik 2014):

$$\mathcal{B}_W := \{\eta_i\}_{i=1}^{D}, \text{ where } \eta_i := WH_{j,l}, \tag{4}$$

with j, l two indices[2] such that $1 \le j < l \le d$ and $i = \sum_{k=1}^{j-1}(d-k) + (l-j)$, and where the set of matrices $\{H_{j,l}\}$, with $1 \le j < l \le d$ is an orthonormal basis for the vector space of $d \times d$ skew-symmetric matrices:

$$H_{j,l} := \frac{1}{\sqrt{2}}\left(e_j e_l^\top - e_l e_j^\top\right), \tag{5}$$

with $e_j \in \mathbb{R}^d$ the vector whose elements are all zero, except the jth component that is equal to one. Let us emphasize that each tangent space is thus endowed with a norm function, and that this norm function coincides with the Frobenius norm:

$$\|\xi_W\| = \langle \xi_W, \xi_W\rangle^{\frac{1}{2}} = \left(\operatorname{tr}\left(\xi_W^\top \xi_W\right)\right)^{\frac{1}{2}}. \tag{6}$$

The Riemannian gradient is the counterpart of the Euclidean gradient on manifolds. Given an arbitrary function $h : \mathcal{O}_d \to \mathbb{R}$, the Riemannian gradient of h at $W \in \mathcal{O}_d$ is the unique vector $\nabla h(W) \in \mathcal{T}_W\mathcal{O}_d$ that satisfies:

$$\langle \xi_W, \nabla h(W)\rangle = \mathrm{D}h(W)[\xi_W] \qquad \forall \xi_W \in \mathcal{T}_W\mathcal{O}_d,$$

i.e., its inner product with any tangent vector ξ_W gives the directional derivative of h along the direction spanned by the tangent vector ξ_W, written here $\mathrm{D}h(W)[\xi_W]$. On the orthogonal group, the Riemannian gradient is simply computed[3] as

$$\nabla h(W) = P_{\mathcal{T}_W\mathcal{O}_d}(\nabla^e h(W)),$$

where $\nabla^e h(W)$ is the Euclidean gradient of h, where h is seen as a function from $\mathbb{R}^{d\times d}$ to $\mathbb{R}$, and $P_{\mathcal{T}_W\mathcal{O}_d}(\cdot)$ is the orthogonal projection on the tangent space $\mathcal{T}_W\mathcal{O}_d$:

$$P_{\mathcal{T}_W\mathcal{O}_d}(M) = W\left(\frac{W^\top M - M^\top W}{2}\right) \tag{7}$$

for all $M \in \mathbb{R}^{d\times d}$. Given the basis (4), one has a Riemannian counterpart to the notion of partial derivative. We define the ith Riemannian partial derivative of h, for the basis (4) of the tangent space $\mathcal{T}_W\mathcal{O}_d$, by:

$$\nabla_i h(W) = \langle \nabla h(W), \eta_i\rangle, \tag{8}$$

where η_i is the ith coordinate vector of the basis (4). This Riemannian partial derivative can be obtained easily from the Euclidean gradient $\nabla^e h(W)$ (typically computed using backpropagation in DNN training):

$$\nabla_i h(W) = \operatorname{tr}\left(H_{j,l}^\top W^\top \nabla h(W)\right) = \operatorname{tr}\left(H_{j,l}^\top W^\top \nabla^e h(W)\right), \tag{9}$$

where $j, l \in \{1, \ldots, d\}$ are defined after (4), and where the second equality comes from the fact that the Frobenius inner product of a skew-symmetric matrix ($H_{j,l}$) and a symmetric matrix ($W^\top(\nabla h(W) - \nabla^e h(W))$) is zero.

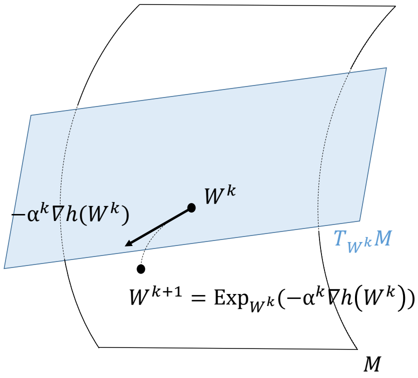

Figure 1: Graphical illustration of a Riemannian gradient descent iteration on a manifold M. At each iteration, we first compute the Riemannian gradient at the current iterate W^k and then update the iterate by "moving on the manifold in the direction opposite to the gradient", using the exponential map.

A last tool that is required in this work is the exponential map, an operator that allows to move on the manifold in a given direction. The exponential map $\mathrm{Exp}_W(\xi_W)$ returns the point obtained by evaluating at time $t = 1$ the geodesic (curve with no acceleration, i.e., straight lines in the Euclidean space), starting at W, with initial velocity ξ_W. For the manifold of orthogonal matrices, the exponential map is given by

$$\mathrm{Exp}_W(\xi_W) = W\ \mathrm{expm}\left(W^\top \xi_W\right),\ \forall \xi_W \in \mathcal{T}_W\mathcal{O}_d, \tag{10}$$

with $\mathrm{expm}\,()$ the matrix exponential. Figure 1 provides an abstract representation of a Riemannian gradient descent iteration.

[2] The following simply characterizes our numbering of the basis elements: the first coordinate vector is associated to the pair ($j = 1, l = 2$), the second to the pair ($j = 1, l = 3$) etc until ($j = d-1, l = d$).

[3] This follows from the fact that the orthogonal group is an embedded submanifold of $\mathbb{R}^{d\times d}$, see (Absil, Mahony, and Sepulchre 2008, Chap. 3) for more information.

Algorithm 1: SRGD: Stochastic Riemannian Gradient Descent

```
1: Let $\{\alpha^k\}$ be a sequence of stepsizes. Set $k = 0$, and initialize the unconstrained and orthogonal variables $X^0 \in \mathbb{R}^{m \times n}, W^0 \in \mathcal{O}_d$.
2: while not converged do
3:     Compute the (stochastic) gradients $\tilde{g}_X^k$ and $\tilde{g}_W^k$
4:     Update the unconstrained variable: $X^{k+1} = X^k - \alpha^k \tilde{g}_X^k$
5:     Update the constrained variable: $W^{k+1} = \mathrm{Exp}_{W^k}(-\alpha^k \tilde{g}_W^k)$
6:     $k := k + 1$
7: end while
8: return solution
```

Algorithm 2: SRCD: Stochastic Riemannian Coordinate Descent

```
1: Let $\{\alpha^k\}$ be a sequence of stepsizes. Set $k = 0$, and initialize the unconstrained and orthogonal variables $X^0 \in \mathbb{R}^{m \times n}, W^0 \in \mathcal{O}_d$.
2: while not converged do
3:     Compute the (stochastic) gradient $\tilde{g}_X^k$
4:     Update the unconstrained variable: $X^{k+1} = X^k - \alpha^k \tilde{g}_X^k$
5:     Select a coordinate $i^k \in \{1, \dots, D\}$ of the tangent space $\mathcal{T}_{W^k}(\mathcal{O}_d)$
6:     Compute the (stochastic) partial derivative $\tilde{g}_{W,i_k}^k$ using (9)
7:     Update the orthogonal variable: $W^{k+1} = \mathrm{Exp}_{W^k}(-\alpha^k \tilde{g}_{W,i_k}^k)$. Due to the structure of $\tilde{g}_{W,i_k}^k$, this step simply involves the multiplication of $W^k$ by a Givens matrix, see (12), and has a computational cost evolving linearly with $d$.
8:     $k := k + 1$
9: end while
```

Stochastic Riemannian Gradient Descent (SRGD)

Building on the previous discussion, we now recall in Algorithm 1 the celebrated stochastic Riemannian gradient descent algorithm, initially proposed and analysed in (Bonnabel 2013), that we apply here to (P). We use this algorithm as a comparison point throughout this paper. Slightly departing from the notation introduced above, and in order to simplify the notation, we use hereafter the notation g_X^k and g_W^k to refer respectively to the gradient of f with respect to the unconstrained variable X and the (Riemannian) gradient of f with respect to W at iteration k, and $\tilde{g}_X^k$ and $\tilde{g}_W^k$ for their stochastic counterparts. In other words, the full exact and stochastic (Riemannian) gradient of f are simply

$$g^k = \begin{pmatrix} g_X^k \\ g_W^k \end{pmatrix} \qquad \text{and} \qquad \tilde{g}^k = \begin{pmatrix} \tilde{g}_X^k \\ \tilde{g}_W^k \end{pmatrix}.$$

In the case when f is a sum of a large number of functions, as in our RNN training application, $\tilde{g}_X^k$ and $\tilde{g}_W^k$ can be seen as the approximations of g_X^k and g_W^k computed over a mini-batch. At each iteration, the stochastic gradients $\tilde{g}_X^k$ and $\tilde{g}_W^k$ are computed, and the iterates X^k, W^k are updated. Note that we evaluate the exponential map (10) at each iteration, involving a matrix exponential whose cost evolves cubically with d (PyTorch matrix exponential implementation currently relies on Taylor/ Padé approximants and the scaling-squaring trick, see (Bader, Blanes, and Casas 2019)).

Proposed algorithm: Stochastic Riemannian Coordinate Descent (SRCD)

Algorithm 2 presents our proposed SRCD algorithm. At each iteration, the unconstrained parameters are updated using stochastic gradient descent, exactly as in Algorithm 1; the modification is in the update rule for the orthogonal parameter. Here, the iterate is only updated along one coordinate of the tangent space, for the basis (4). This amounts to rotating a pair of columns of the iterate W^k of an angle that depends on the stepsize and the component of the gradient along that coordinate. The stochastic gradient $\tilde{g}_W^k$ is thus replaced by the following tangent vector, which is aligned with the i^kth coordinate vector η_{i^k} of the tangent space $\mathcal{T}_{W^k}\mathcal{O}_d$:

$$\tilde{g}_{W,i^k}^k = \langle \tilde{g}_W^k, \eta_{i^k} \rangle \eta_{i^k}, \tag{11}$$

where $\eta_{i^k} = W^k H_{j^k,l^k}$ for some given j^k, l^k defined below (4).

Note that, since the matrix H_{j^k,l^k} has a very special structure, see (5), the exponential map in Line 7 can be written as a multiplication by a Givens matrix. Indeed, writing $\theta^k = \langle \tilde{g}_W^k, \eta_{i^k} \rangle$, there holds

$$\mathrm{Exp}_{W^k}(\theta^k \eta_{i^k}) = \mathrm{Exp}_{W^k}(\theta^k W^k H_{j^k,l^k}) = W^k G_{j^k,l^k}(\theta^k), \tag{12}$$

with

$$G_{j^k,l^k}(\theta^k) := \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cos(\theta^k) & \cdots & \sin(\theta^k) & \vdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -\sin(\theta^k) & \cdots & \cos(\theta^k) & \vdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \tag{13}$$

a Givens matrix (Shalit and Chechik 2014). Right-multiplying the iterate W^k with $G_{j^k,l^k}(\theta^k)$ has the effect of rotating clockwise the j^kth and l^kth columns of W^k, with rotation angle θ^k. In particular, since Givens matrices are sparse (having only 4 nonzero elements), the evaluation of the right-hand side of (12) has a cost of about $6d$ flops, which is much lower than the cost of the matrix exponential that is used when updating the W variable in SRGD.

Algorithm 2 also requires a strategy for selecting the coordinate at each iteration. Two well-known strategies in the Euclidean setting are uniform sampling, where i^k is sampled independently and uniformly among $\{1, \dots, D\}$ at each iteration (with $D = d(d-1)/2$ the dimension of the manifold), and the Gauss-Southwell rule, where i^k is the coordinate associated to the fastest local variation of the objective:

$$i^k = \underset{i \in \{1,\dots,D\}}{\operatorname{argmax}} \|g^k_{W,i}\|. \tag{14}$$

We compare both strategies numerically in Section 3.

Convergence analysis

Our convergence analysis heavily relies on the convergence analysis of stochastic gradient descent in the Euclidean setting (see (Bottou, Curtis, and Nocedal 2018, Chap. 4)). The main contribution of the convergence analysis is a careful extension to the Riemannian setting, and to coordinates updates for one of the variables. First, let us introduce the following smoothness assumption on the function f, following (Boumal, Absil, and Cartis 2018).

Assumption 1 *The function $f : \mathbb{R}^{m\times n} \times \mathcal{O}_d \to \mathbb{R}$ is L-smooth, i.e., satisfies for all $(X, W) \in \mathbb{R}^{m\times n} \times \mathcal{O}_d$ and $(\nu, \mu) \in \mathbb{R}^{m\times n} \times \mathcal{T}_W\mathcal{O}_d$:*

$$|f(X + \nu, \operatorname{Exp}_W(\mu)) - f(X, W) - \langle g_X(X, W), \nu\rangle$$
$$- \langle g_W(X, W), \mu\rangle| \leq \frac{L}{2}\left(\|\nu\|^2 + \|\mu\|^2\right).$$

Assumption 1 is guaranteed to be satisfied if the Euclidean gradient of the function f (when the latter is seen as a function on $\mathbb{R}^{m\times n} \times \mathbb{R}^{d\times d}$) is Lipshitz-continuous, with Lipshitz constant L. This follows from Lemma 7 in (Boumal, Absil, and Cartis 2018), noticing that the proof of that lemma still holds when the manifold is a product manifold of a compact manifold (here the orthogonal group) and a Euclidean space[4].

Our second assumption is classical when analysing stochastic gradient descent in the Euclidean setting, see, e.g., (Bottou, Curtis, and Nocedal 2018, Chap. 4). Let us write hereafter $\mathcal{F}^k = \{\tilde{g}^0_X, \tilde{g}^0_W, i^0, \dots, \tilde{g}^{k-1}_X, \tilde{g}^{k-1}_W, i^{k-1}\}$, the σ-algebra generated by the random variables before iteration k.

Assumption 2 *The gradients and iterates of the algorithm satisfy the conditions:*

1. *The sequence of iterates $\{(X^k, W^k)\}_{k\in\mathbb{N}}$ is contained in an open set over which the value of the function f is lower bounded by some f_{inf},*
2. *There exist $\mu_X, \mu_W > 0$ such that for all $k \in \mathbb{N}$ and $Z \in \{X, W\}$,*
$$\langle g^k_Z, \mathbb{E}\left[\tilde{g}^k_Z|\mathcal{F}^k\right]\rangle \geq \mu_Z\|g^k_Z\|^2.$$
3. *There exists $C_X, C_W \geq 0$ and $M_X, M_W \geq 0$ such that, for all $k \in \mathbb{N}$ and for $Z \in \{X, W\}$,*
$$\mathbb{E}\left[\|\tilde{g}^k_Z\|^2|\mathcal{F}^k\right] \leq C_Z + M_Z\|g^k_Z\|^2.$$

Under these assumptions, we prove the following result, which is analogous to (Bottou, Curtis, and Nocedal 2018, Thm. 4.10).

Theorem 1 (Convergence result) *Under Assumptions 1 and 2, the sequence of iterates $\{(X^k, W^k)\}$ generated by Algorithm 2, with coordinate i^k selected for each k uniformly at random among $\{1, \dots, D\}$, with $D = d(d-1)/2$ the dimension of $\mathcal{O}_d$, and using a sequence of stepsizes α^k that satisfies the Robbins-Monro conditions (Robbins and Monro 1951):*

$$\lim_{k\to\infty} \sum_{i=0}^{k} \alpha^k = \infty \qquad \lim_{k\to\infty} \sum_{i=0}^{k} (\alpha^k)^2 < \infty \tag{15}$$

satisfies

$$\mathbb{E}\left[\frac{1}{\sum_{k=0}^{K}\alpha^k}\sum_{k=0}^{K}\alpha^k\|g^k\|^2\right] \to 0 \qquad \text{as } K \to \infty, \tag{16}$$

where $g^k = [g^{k\top}_X\, g^{k\top}_W]^\top$ is the (Riemannian) gradient of the objective at iterate (X^k, W^k).

Proof 1 *See supplementary material.*

3 Numerical experiments

We consider in this paper the copying memory task, which consists in remembering a sequence of letters from some alphabet $\mathcal{A} = \{a_k\}_{k=1}^N$. The model is given a sequence

$$(\underbrace{a_{s_1}, a_{s_2}, \dots, a_{s_K}}_{\text{Sequence to remember}}, \underbrace{b, b, \dots, b}_{L\text{ blank symbols}}, \underbrace{c}_{\text{Start symbol}}, \underbrace{b, \dots, b}_{K\text{-1 blank symbols}}),$$

with $a_{s_i} \in \mathcal{A}$ for all $i \in \{1, \dots, K\}$, and b, c two additional letters that do not belong to the alphabet (respectively the "blank" and "start" symbols). The whole input sequence has total length $L + 2K$; the first K elements are the elements to remember. Once the model reads the "start" symbol, it should start replicating the sequence it has memorized. For the input sequence given above, the output of the network should thus be the sequence $(b, \dots, b, a_{s_1}, a_{s_2}, \dots, a_{s_K})$, made of $L + K$ replications of the blank letter followed by the sequence the model was asked to memorize. We rely on https://github.com/Lezcano/expRNN for the PyTorch implementation of the model architecture and of the copying memory task. In particular, the alphabet $\mathcal{A}$ comprises of 9 different elements, $L = 1000$, $K = 10$, the batchsize is equal to 128, and the recurrent matrix is an orthogonal matrix of size 190×190. The loss for this problem is the cross-entropy.

Almost sparsity of the recurrent gradient. Let us first illustrate that the gradient of the loss with respect to the orthogonal parameter (the recurrent weight matrix) has an approximately sparse representation in the basis (4). Figure 2 illustrates the repartition of the Riemannian partial derivatives (8) (in absolute value, and computed over a minibatch), both at the initialization and after 500 training iterations. This figure indicates that a few partial derivatives have absolute value about two orders of magnitude larger than the bulk of partial derivatives. This observation supports the choice of the Gauss-Southwell coordinate selection strategy proposed in this paper. In this experiment, at initialization, 0.1% (resp.

[4] Actually, we just need the gradient of the function to be Lipshitz continuous on $\mathbb{R}^{m\times n} \times \operatorname{Conv}(\mathcal{O}_d) \subset \mathbb{R}^{m\times n} \times \mathbb{R}^{d\times d}$, where $\operatorname{Conv}(\mathcal{O}_d)$ is the convex hull of $\mathcal{O}_d$.

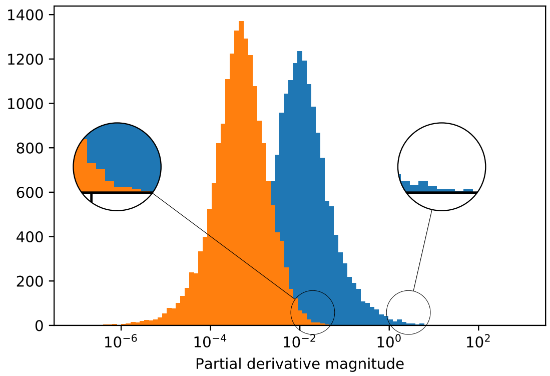

Figure 2: Histograms of the magnitude of the partial derivatives of the loss for the copying memory problem, both at initialization (blue) and after 500 iterations (orange). Note that a few partial derivatives are dominating by a couple of orders of magnitude.

4.2%) of the coordinates represent 95% (resp. 99%) of the norm of the Riemannian gradient. After 500 iterations, 9% (resp. 28.4%) of the coordinates represent 95% (resp. 99%) of the norm of the Riemannian gradient.

Comparison of the algorithms. To illustrate further the benefits of our approach, we compare here SRGD (as a baseline) with two variants of SRCD; selecting the coordinate either uniformly at random (SRCD-U) at each iteration, or according to the Gauss-Southwell rule (SRCD-GS), see (14). For the sake of completeness, we also consider a block version of SRCD-GS, in which a few coordinates are selected at each iteration (the block size was here set to 0.5% of the total number of coordinates in the tangent space).[5] Though typically state-of-the-art algorithms addressing this problem rely on adaptive stepsizes, we compare here these algorithms using a fixed stepsize, so that the gap between the training loss of SRGD and SRCD-GS or SRCD-U gives us a measure of how harmful the restriction of the gradient to one/some coordinates is for the optimization. Figure 3 shows the evolution of the loss for the different algorithms, using a fixed stepsize set to $2 \cdot 10^{-4}$. Overall, our numerical experiments illustrate a good initial behavior of our proposed algorithms compared to SRGD, with a fast decrease of the loss over the first iterations. Though SRCD-GS outperforms SRCD-U over the first iterations, these two methods perform similarly over a larger number of iterations (the curve for SRCD-U has been averaged over ten runs). Updating at each iteration a block of coordinates improves significantly the performance, and provides a decrease of the loss that is very close to SRGD. Note also that, following our discussion on the approximately sparse representation of the stochastic Riemannian gradient in the basis (4), it might be interesting to increase progressively the number of coordinates during the optimization; this is left for future research.

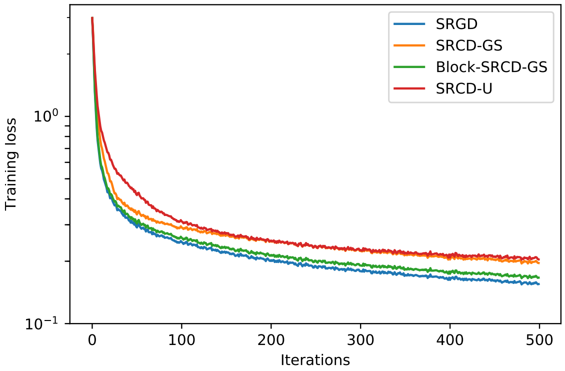

Figure 3: Comparison of stochastic Riemannian gradient descent (SRGD) and stochastic Riemannian coordinate descent (SRCD) on the copying memory problem. Note that a decrease of the loss comparable to SRGD can be obtained by restricting the update to one/some coordinates in the tangent space, resulting in a cost per iteration that evolves linearly with d, instead of cubically for SRGD.

Comparison in terms of computational cost. Table 1 presents the cost per iteration/update (with CUDA synchronization) using different optimizers for the copying memory problem (see supplementary material for experimental details). Though we would have expected SRCD to be significantly faster than SRGD, the computational cost of both methods is actually very close. Table 1 indicates that the cost per iteration is indeed dominated by the cost of the backpropagation step (computing the Euclidean gradients with respect to the parameters) and that the cost of the parameter update is negligible in comparison. Actually, we didn't even notice any significant increase of computational time when imposing orthogonal constraints on the recurrent matrix via stochastic Riemannian gradient descent, compared to vanilla stochastic gradient descent. Though this might seem surprising given the large amount of literature aiming to decrease the computational cost required to preserve orthogonality, this is in line with a discussion in (Lezcano-Casado and Martínez-Rubio 2019). Writing b the batch-size and l the length of the input sequence, the authors of (Lezcano-Casado and Martínez-Rubio 2019) mention that the cost of backpropagation (restricted to the recurrent matrix) is $\mathcal{O}(bld^2)$ instead of $\mathcal{O}(d^3)$ for approximating to a satisfactory accuracy a matrix exponential using the scaling-squaring trick and/or Padé approximants. There follows that, when $bl > d$, backpropagation dominates the cost per iteration. As expected, further experiments indicate that the cost of the matrix exponential evaluation and the cost of the backpropagation become more and more comparable as the size of the recurrent matrix increases, when keeping the

[5] Note that, in order for the update rule of the orthogonal parameter to be expressible in terms of Givens matrices, the selected coordinates have to correspond to disjoint pairs of columns. As this experiment simply aims to compare the methods in terms of accuracy, the matrix exponential was used to avoid here this constraint.

batchsize and length of the sequence fixed, so that, for very large architectures, the use of SRGD would become prohibitive. Note also that, while in our implementations we run the full backpropagation algorithm (as standard deep learning libraries face difficulties to compute a subset of partial derivatives instead of the whole gradient), theoretically the cost of backpropagation could be decreased from $\mathcal{O}(bld^2)$ to $\mathcal{O}(bld)$, reducing substantially the computational cost per iteration of our proposed approach compared to the numbers given in Table 1.

Table 1: Average run-time cost per iteration/update of the iterates for the copying memory problem using the different optimizers considered in this paper

Process	Optimizer			
	SGD	SRGD	SRCD-GS	SRCD-U
loss.backward() + optim.step()	0.3125	0.3311	0.3191	0.3429
optim.step()	.00164	.00197	.00177	.00223

To conclude, the proposed algorithm is very promising for big architectures (large number of recurrent states), since the gap in accuracy per iteration between SRGD and SRCD illustrated on Fig. 3 is expected to be largely compensated by the computational cost savings per iteration. A detailed study is out of the scope of this paper and we defer it to future work.

4 Conclusions

We have proposed SRCD, a new algorithm for orthogonal RNN training with computational cost per iteration in $\mathcal{O}(d)$, in contrast with the $\mathcal{O}(d^3)$ cost of SRGD. We proved the convergence of our proposed algorithm under typical assumptions on the training problem (Lipshitz smoothness of the objective, classical assumptions on the gradient noise and stepsizes satisfying the Robbins-Monro conditions), for coordinates selected uniformly at random in $\{1, \ldots, D\}$, with $D = d(d-1)/2$ the manifold dimension. We have also shown numerically that the Riemannian gradient has an approximately sparse representation in the basis (4), and leveraged this observation by proposing a Gauss-Southwell coordinate selection rule. Future research could aim to endow the proposed optimizer with an adaptive stepsize such as Adam or RMSProp, following the recent attempts (Kasai, Jawanpuria, and Mishra 2019; Bécigneul and Ganea 2019; Lezcano-Casado 2020) for developing adaptive stepsizes on manifolds.

Acknowledgements

The first author's work is supported by the National Physical Laboratory, UK.

References

Abrol, V.; and Tanner, J. 2020. Information-Bottleneck under Mean Field Initialization. In *International Conference on Machine Learning (ICML) Workshop on Uncertainty & Robustness in Deep Learning*. Vienna, Austria.

Absil, P.-A.; Mahony, R.; and Sepulchre, R. 2008. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press. ISBN 978-0-691-13298-3.

Arjovsky, M.; Shah, A.; and Bengio, Y. 2016. Unitary Evolution Recurrent Neural Networks. In *International Conference on Machine Learning (ICML)*, 1120–1128. New York, USA.

Bader, P.; Blanes, S.; and Casas, F. 2019. Computing the Matrix Exponential with an Optimized Taylor Polynomial Approximation. *Mathematics*, 7(12).

Bansal, N.; Chen, X.; and Wang, Z. 2018. Can We Gain More from Orthogonality Regularizations in Training Deep CNNs? In *International Conference on Neural Information Processing Systems (NeurIPS)*, 4266–4276. Montréal, Canada.

Bécigneul, G.; and Ganea, O.-E. 2019. Riemannian Adaptive Optimization Methods. In *International Conference on Learning Representations (ICLR)*. New Orleans, USA.

Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on neural networks*, 5(2): 157–166.

Bonnabel, S. 2013. Stochastic gradient descent on Riemannian manifolds. *IEEE Transactions Automatic Control*, 58(9): 2217–2229.

Bottou, L.; Curtis, F. E.; and Nocedal, J. 2018. Optimization Methods for Large-Scale Machine Learning. *SIAM Review*, 60(2): 223–311.

Boumal, N. 2020. An introduction to optimization on smooth manifolds. Available online.

Boumal, N.; Absil, P.-A.; and Cartis, C. 2018. Global rates of convergence for nonconvex optimization on manifolds. *IMA Journal of Numerical Analysis*, 39(1): 1–33.

Giles, C. L.; Kuhn, G. M.; and Williams, R. J. 1994. Dynamic recurrent neural networks: Theory and applications. *IEEE Transactions on Neural Networks*, 5(2): 153–156.

Gutman, D. H.; and Ho-Nguyen, N. 2021. Coordinate Descent Without Coordinates: Tangent Subspace Descent on Riemannian Manifolds. *Mathematics of Operations Research (Accepted)*.

Helfrich, K. E.; Willmott, D.; and Ye, Q. 2018. Orthogonal Recurrent Neural Networks with Scaled Cayley Transform. In *35th International Conference on Machine Learning (ICML)*. Stockholm, Sweden.

Hu, W.; Xiao, L.; and Pennington, J. 2020. Provable benefit of orthogonal initialization in optimizing deep linear networks. In *International Conference on Learning Representations (ICLR)*. Virtual.

Huang, L.; Liu, L.; Zhu, F.; Wan, D.; Yuan, Z.; Li, B.; and Shao, L. 2020. Controllable Orthogonalization in Training DNNs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6428–6437. Virtual.

Huang, L.; Liu, X.; Lang, B.; Yu, A. W.; Wang, Y.; and Li, B. 2018. Orthogonal Weight Normalization: Solution to Optimization over Multiple Dependent Stiefel Manifolds in Deep Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI)*. New Orleans, USA.

Hyland, S. L.; and Rätsch, G. 2017. Learning Unitary Operators with Help From u(n). In *AAAI Conference on Artificial Intelligence (AAAI)*. San Francisco, USA.

Ishteva, M.; Absil, P.-A.; and Dooren, P. V. 2013. Jacobi Algorithm for the Best Low Multilinear Rank Approximation of Symmetric Tensors. *SIAM Journal on Matrix Analysis and Applications*, 34(2): 651–672.

Jia, K.; Li, S.; Wen, Y.; Liu, T.; and Tao, D. 2020. Orthogonal Deep Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(4): 1352 – 1368.

Jia, K.; Tao, D.; Gao, S.; and Xu, X. 2017. Improving training of deep neural networks via Singular Value Bounding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3994–4002. Honolulu, USA.

Jing, L.; Shen, Y.; Dubcek, T.; Peurifoy, J.; Skirlo, S.; LeCun, Y.; Tegmark, M.; and Soljačić, M. 2017. Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNNs. In *International Conference on Machine Learning (ICML)*, 1733–1741. Sydney, Australia.

Kasai, H.; Jawanpuria, P.; and Mishra, B. 2019. Riemannian adaptive stochastic gradient algorithms on matrix manifolds. In *International Conference on Machine Learning (ICML)*, 3262–3271. Long Beach, USA.

Lezcano-Casado, M. 2019. Trivializations for gradient-based optimization on manifolds. In *International Conference on Neural Information Processing Systems (Neurips)*. Vancouver, Canada.

Lezcano-Casado, M. 2020. Adaptive and Momentum Methods on Manifolds Through Trivializations. arxiv preprint 2010.04617.

Lezcano-Casado, M.; and Martínez-Rubio, D. 2019. Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group. In *International Conference on Machine Learning (ICML)*, 3794–3803. Long Beach, Calif., USA.

Maduranga, K. D. G.; Helfrich, K. E.; and Ye, Q. 2019. Complex Unitary Recurrent Neural Networks Using Scaled Cayley Transform. In *AAAI Conference on Artificial Intelligence (AAAI)*. Honolulu, TH, USA.

Meghwanshi, M.; Jawanpuria, P.; Kunchukuttan, A.; Kasai, H.; and Mishra, B. 2018. McTorch, a manifold optimization library for deep learning. Technical report, arXiv preprint arXiv:1810.01811.

Mhammedi, Z.; Hellicar, A.; Rahman, A.; and Bailey, J. 2017. Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections. In *International Conference on Machine Learning (ICML)*, 2401–2409. Sydney, Australia.

Murray, M.; Abrol, V.; and Tanner, J. 2021. Activation function design for deep networks: linearity and effective initialisation. *Applied and Computational Harmonic Analysis (Accepted)*.

Ozay, M.; and Okatani, T. 2016. Optimization on Submanifolds of Convolution Kernels in CNNs. arxiv preprint 1610.07008.

Palagi, L.; and Seccia, R. 2019. Block layer decomposition schemes for training deep neural networks. *Journal of Global Optimization*, 77: 97–124.

Pascanu, R.; Mikolov, T.; and Bengio, Y. 2013. On the difficulty of training recurrent neural networks. In *Internationl Conference on Machine Learning (ICML)*, III–1310–III–1318. Atlanta, USA.

Pennington, J.; Schoenholz, S. S.; and Ganguli, S. 2017. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In *Conference on Neural Information Processing Systems (NeurIPS)*. Long Beach, USA.

Robbins, H.; and Monro, S. 1951. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3): 400–407.

Shalit, U.; and Chechik, G. 2014. Coordinate-descent for learning orthogonal matrices through Givens rotations. In *International Conference on Machine Learning (ICML)*, 548–556. Beijing, China.

Wisdom, S.; Powers, T.; Hershey, J. R.; Roux, J. L.; and Atlas, L. 2016. Full-Capacity Unitary Recurrent Neural Networks. In *International Conference on Neural Information Processing Systems (NeurIPS)*. Barcelona, Spain.

Wright, S. J. 2015. Coordinate descent algorithms. *Mathematical Programming*, 151: 3—34.

Yoshida, Y.; and Miyato, T. 2017. Spectral Norm Regularization for Improving the Generalizability of Deep Learning. arxiv preprint 1705.10941.

Zeng, J.; Lau, T. T.-K.; Lin, S.; and Yao, Y. 2019. Global Convergence of Block Coordinate Descent in Deep Learning. In *International Conference on Machine Learning (ICML)*, 7313–7323. Long Beach, Calif., USA.