Second-order networks in PyTorch

Daniel Brooks, Olivier Schwander, Frédéric Barbaresco, Jean-Yves Schneider, Matthieu Cord

Second-order networks in PyTorch

Daniel Brooks[1,2], Olivier Schwander[2], Frédéric Barbaresco[1], Jean-Yves Schneider[1], and Matthieu Cord[2]

[1] Thales Land and Air Systems, Advanced Radar Concepts (Limours, FRANCE)
[2] Sorbonne Université, CNRS, LIP6 - Laboratoire d'Informatique de Paris 6, F-75005 (Paris, FRANCE)

Abstract. Classification of Symmetric Positive Definite (SPD) matrices is gaining momentum in a variety machine learning application fields. In this work we propose a Python library which implements neural networks on SPD matrices, based on the popular deep learning framework Pytorch.

Keywords: SPD matrix, covariance, second-order neural network, Riemannian machine learning

1 Introduction

Information geometry-based machine learning has recently been rapidly emerging in a broad spectrum of learning scenarios, and deep learning has been no exception. Notably, works such as [14], [15] and [13] introduce neural networks respectively operating on Lie groups, Grassmann spaces, and SPD matrices. Tthe natural representation of any temporally or spatially structured signal as a Gaussian process allows for a near universal possible interpretation of the signal as its temporal or spatial covariance, which is an SPD matrix, i.e. which belongs to the SPD Riemannian manifold, which we note $\mathcal{S}_*^+$. Previous works make use of the SPD representation in other contexts than deep learning: for instance, Riemannian metric learning on $\mathcal{S}_*^+$ is developed in [24], while [23] review kernel methods on $\mathcal{S}_*^+$, with a primary applicative focus on electro-encephalogram/cardiogram (EEG/ECG) classification. In a similar vein, [4] and [2] extend barycenter-based classification methods to the SPD Riemannian framework. On the other hand, [9] propose the usage of SPD matrices as a region descriptor in images, with applications in image segmentation. The work in [17] pushed the idea further by allowing the region covariance descriptor to be appended to a deep neural representation of an image, and by doing so introduced the first hints of automatic backpropagation in a Riemannian setting. Finally, the older theoretical developments in [7] notably allowed the extension of optimization methods to manifold-valued neural networks as later utilized in [13], [10] and [8]. Even more recent works, namely [1] and [25] have appended SPD neural networks to classical, Euclidean ones, by considering the second-order moments of the learnt feature representations as a suitable representation for the data.

In this environment of popularization of deep learning on SPD matrices, we propose *torchspdnet*, a Python library featuring many relevant modules necessary

to build a neural network operating on SPD matrices. We do so in the popular PyTorch framework [21]. While other libraries were proposed for general learning on manifolds (Geomstats [20]), deep learning on manifolds (McTorch [19]), optimization on manifolds (Manopt [5]) and SPD matrix manipulation (PyRiemann [3]), ours focusses exclusively on deep learning architectures for SPD matrices, providing seamless integration with any PyTorch development framework. In the following section we describe the core components of a SPD neural network, which we may call SPDNet. The third section deals with the optimization of a manifold-valued network. Finally, we show some use cases.

2 Second order networks

Here we describe the architecture of an SPDNet. We begin with the core building blocks, then show how to build a network using these blocks in various scenarios. Following the logic of most modern deep learning frameworks including PyTorch, the core building blocks, or layers of the network, are implemented as individual modules.

2.1 SPD layers

Similarly to a classical neural network, an SPDNet aims at building a hierarchical sequence of more compact and discriminative manifolds as illustrated in figure 1. Three main layers are introduced in [13], described below.

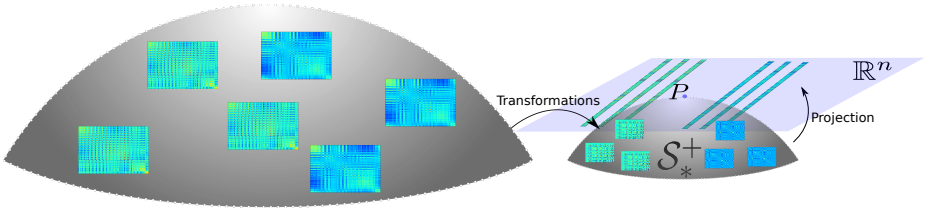

Fig. 1. Illustration of a generic SPD neural network. Successive bilinear layers followed by activations build a feature SPD manifold, which is then transformed to a Euclidean space to allow for classification.

BiMap The bilinear mapping (BiMap) layer transforms an input matrix $X^{(l-1)}$ of size $n^{(l-1)}$ at layer $(l-1)$ into an SPD matrix $X^{(l)}$ of size $n^{(l)}$ at layer (l) using a basis change matrix $W^{(l)}$, required to be full-rank, which in turn constrains $n^{(l)} \leq n^{(l-1)}$. In practice $W^{(l)}$ is in fact constrained to be semi-orthogonal:

$$X^{(l)} = {W^{(l)}}^T X^{(l-1)} W^{(l)} \text{ with } W^{(l)} \in \mathcal{O}(n^{(l-1)}, n^{(l)}) \tag{1}$$

In the equation above, $\mathcal{O}(n^{(l-1)}, n^{(l)})$ is the manifold of semi-orthogonal rectangular matrices, also called Stiefel manifold, and $X^{(l-1)} = U^{(l-1)}\Sigma^{(l-1)}U^{(l-1)^T}$ designates the eigenvalue decomposition of $X^{(l-1)}$

ReEig The transformation layer is followed by an activation, in this case a rectified eigenvalues (ReEig) layer:

$$X^{(l)} = U^{(l-1)} \max(\Sigma^{(l-1)}, \epsilon I_{n^{(l-1)}}) U^{(l-1)^T} \text{ with } P^{(l-1)} = U^{(l-1)}\Sigma^{(l-1)}U^{(l-1)^T} \tag{2}$$

The ReEig layer also makes use of an eigenvalue decomposition as it operates directly on the eigenvalues, with ϵ being a fixed threshold set to a default value of $1e-4$.

LogEig After a succession of transformations and activations, the final feature manifold is then transformed via a logarithmic mapping to a Euclidean space (LogEig layer) to perform the actual classification:

$$X^{(l)} = vec(\ U^{(l)} \log(\Sigma^{(l)}) U^{(l)^T}\ ) \text{ , with } P^{(l)} = U^{(l)}\Sigma^{(l)}U^{(l)^T} \tag{3}$$

The LogEig layer is justified in the Log-Euclidian Metric (LEM) framework, independently introduced in [22] and [11], which shows a correspondence from the manifold $\mathcal{S}_*^+$ to the Euclidean space $\mathcal{S}^+$ of symmetric matrices through the matrix logarithm. The *vec* operator denotes matrix vectorization.

3 Training

The main difficulties of learning an SPDNet lie both in the backpropagation through structured Riemannian functions [16] [6], and in the manifold-constrained optimization [7].

3.1 Structured derivatives

Manifold-valued functions, such as the LogEig and ReEig layers, require a generalization of the chain rule, key to the backpropgation algorithm. Both these layers can be represented in a unified fashion as a non-linear function f acting directly on the eigenvalues of the input matrix $X^{(l-1)} = U^{(l-1)}\Sigma^{(l-1)}U^{(l-1)^T}$. Then, the backpropagation goes as follows: given the succeeding gradient $\frac{\partial L^{(l)}}{\partial X^{(l)}}$, the output gradient $\frac{\partial L^{(l-1)}}{\partial X^{(l-1)}}$ is:

$$\frac{\partial L^{(l-1)}}{\partial X^{(l-1)}} = U\left(L \odot (U^T(\frac{\partial L^{(l)}}{\partial X^{(l)}})U)\right)U^T \tag{4}$$

In the previous equation, the Loewner matrix of finite differences L is defined as:

$$L_{ij} = \begin{cases} \frac{f(\sigma_i)-f(\sigma_j)}{\sigma_i-\sigma_j} & \text{if } \sigma_i \neq \sigma_j \\ f'(\sigma_i) & \text{otherwise} \end{cases} \tag{5}$$

3.2 Constrained optimization

In the specific case of the BiMap layer, the transformation matrix W is constrained to the Stiefel manifold. The Euclidean gradient $\frac{\partial \mathcal{L}}{\partial G}$ of the loss function $\mathcal{L}$ does not respect the geometry of the manifold: as such the gradient descent is ill-defined. $\frac{\partial \mathcal{L}}{\partial G}$. The correct Riemannian gradient is obtained by tangent projection $\Pi\mathcal{T}_W$ on the manifold at W. The update is then obtained by computing the geodesic on the manifold from W towards the Riemannian gradient, also called exponential mapping $Exp_W(X)$. We illustrate this process in figure 2. Both the tangent projection and geodsic are known on the Stiefel manifold [7]: ction Exp_W have a closed form [7]:

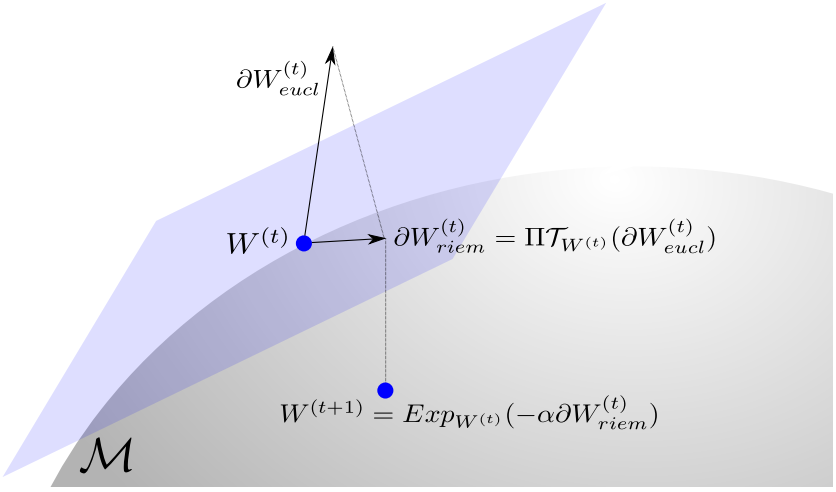

Fig. 2. Illustration of manifold-constrained gradient update. The Euclidean gradient is projected to the tangent space, then mapped to the manifold.

$$\begin{aligned} \Pi\mathcal{T}_W(X) &= X - WW^T X \\ Exp_W(X) &= Orth(W + X) \end{aligned} \tag{6}$$

The operator $Orth$ represents the orthonormalization of a free family of vectors, i.e. the Q matrix in the QR decomposition.

3.3 Summary

The library we propose seamlessly integrates orthogonally-constrained optimization on $\mathcal{S}_*^+$: the code for setting up the learning of a model in PyTorch is only modified in the usage of the *MixOptimizer* class, which mixes a conventional optimizer with the Riemannian ones:

```
import torch.nn as nn
from mixoptimizer import MixOptimizer
...
model=... #define the model
...
l=nn.CrossEntropyLoss()
opt=MixOptimizer(model.parameters(),lr=lr,momentum=0.9,weight_decay=5e-4) #defin
...
l.backward()
opt.step() #in the training loop, compute gradients and update weights as usuall
```

4 Use cases

Here we show how to use the library in practice. Following the PyTorch logic, elementary functions are defined in *torchspdnet.functional* and high-level modules in *torchspdnet.nn*.

4.1 Basic SPDNet model

Here we give the most basic use case scenario: given input covariance data of size 20×20, we build an SPDNet which reduces its size to 15 then 10 through two BiMaps and a ReEig activation, followed by the LogEig and vectorization. Finally, a standard fully-connected layer allows for classification over the 3 classes

```
import torch.nn as nn
import torchspdnet.nn as nn_spd

model=nn.Sequential(
        nn_spd.BiMap(1,1,20,15),
        nn_spd.ReEig(),
        nn_spd.BiMap(1,1,15,10),
        nn_spd.LogEig(),
        nn_spd.Vectorize(),
        nn.Linear(10**2,3)
    )
```

Note that our implementation of the BiMap module supports an arbitrary number of channels, represented by the additional parameters all set to 1 in this example.

4.2 First-order and second-order combined

In a more complex example, an SPDNet acts upon the features maps of a convolutional network. For an image recognition task, these features may come from a

pre-trained deep network but nothing keeps from training the whole network in an end-to-end fashion or to fine-tune the parameters. Here we describe the combination of a pre-trained ResNet-18 [12] on the CIFAR10 [18] challenge and of SPDNet layers. We call such a model a second-order neural network (SOCNN).

```
import torch.nn as nn
import torchspdnet.nn as nn_spd
from resnet import ResNet18

class SOCNN(nn.Module):
    def __init__(self):
        super(__class__,self).__init__()

        self.model_fo=ResNet18() #first-order model
        self.model_fo.load_state_dict(th.load('pretrained/ResNet18.pth')['state_

        self.connection=nn.Conv2d(512,256,kernel_size=(1,1)) #convolutional con

        self.model_so=nn.Sequential( #second-order model
            nn_spd.BiMap(1,1,256,128),
            nn_spd.ReEig(),
            nn_spd.BiMap(1,1,128,64),
        ).to(self.device_so)

        self.dense=nn.Sequential(
            nn.Linear(64**2,1024),
            nn.Linear(1024,10)
        )

    def forward(self,x):
        x_fo=self.model_fo(x)
        x_co=self.connection(x_fo)
        x_sym=nn_spd.CovPool()(x_co.view(x_co.shape[0],x_co.shape[1],-1))
        x_so=self.model_so(x_sym)
        x_vec=nn_spd.LogEig()(x_so).view(x_so.shape[0],x_so.shape[-1]**2)
        y=self.dense(x_vec)
        return y
```

5 Conclusion

We have proposed a PyTorch library for deep learning on SPD matrices. We hope its versatility and natural integration in any PyTorch workflow will allow future projects to more readily make use of the potential of exploiting covariance structure in data at any level.

References

1. Acharya, D., Huang, Z., Paudel, D.P., Gool, L.V.: Covariance Pooling for Facial Expression Recognition p. 8
2. Barachant, A., Bonnet, S., Congedo, M., Jutten, C.: Multiclass BrainComputer Interface Classification by Riemannian Geometry. IEEE Transactions on Biomedical Engineering **59**(4), 920–928 (Apr 2012). https://doi.org/10.1109/TBME.2011.2172210, http://ieeexplore.ieee.org/document/6046114/
3. barachant, a.: Python package for covariance matrices manipulation and Biosignal classification with application in Brain Computer interface: alexandrebarachant/pyRiemann (Feb 2019), https://github.com/alexandrebarachant/pyRiemann, original-date: 2015-04-19T16:01:44Z
4. Barachant, A., Bonnet, S., Congedo, M., Jutten, C.: Classification of covariance matrices using a Riemannian-based kernel for BCI applications. Neurocomputing **112**, 172–178 (Jul 2013). https://doi.org/10.1016/j.neucom.2012.12.039, https://hal.archives-ouvertes.fr/hal-00820475
5. Boumal, N., Mishra, B., Absil, P.A., Sepulchre, R.: Manopt, a Matlab Toolbox for Optimization on Manifolds. Journal of Machine Learning Research **15**, 1455–1459 (2014), http://jmlr.org/papers/v15/boumal14a.html
6. Brodski, M., Dalecki, J., dus, O., Iohvidov, I., Kren, M., Ladyenskaja, O., Lidski, V., Ljubi, J., Macaev, V., Povzner, A., Sahnovi, L., muljan, J., Suharevski, I., Uralceva, N.: Thirteen Papers on Functional Analysis and Partial Differential Equations, American Mathematical Society Translations: Series 2, vol. 47. American Mathematical Society (Dec 1965). https://doi.org/http://dx.doi.org/10.1090/trans2/047, http://www.ams.org/home/page/
7. Edelman, A., Arias, T., Smith, S.: The Geometry of Algorithms with Orthogonality Constraints. SIAM Journal on Matrix Analysis and Applications **20**(2), 303–353 (Jan 1998). https://doi.org/10.1137/S0895479895290954, https://epubs.siam.org/doi/abs/10.1137/S0895479895290954
8. Engin, M., Wang, L., Zhou, L., Liu, X.: DeepKSPD: Learning Kernel-matrix-based SPD Representation for Fine-grained Image Recognition. arXiv:1711.04047 [cs] (Nov 2017), http://arxiv.org/abs/1711.04047, arXiv: 1711.04047
9. Faulkner, H., Shehu, E., Szpak, Z.L., Chojnacki, W., Tapamo, J.R., Dick, A., Hengel, A.v.d.: A Study of the Region Covariance Descriptor: Impact of Feature Selection and Image Transformations. In: 2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA). pp. 1–8 (Nov 2015). https://doi.org/10.1109/DICTA.2015.7371222
10. Gao, Z., Wu, Y., Bu, X., Jia, Y.: Learning a Robust Representation via a Deep Network on Symmetric Positive Definite Manifolds. arXiv:1711.06540 [cs] (Nov 2017), http://arxiv.org/abs/1711.06540, arXiv: 1711.06540
11. Harris, W.F.: The average eye. Ophthalmic and Physiological Optics **24**(6), 580–585 (2004). https://doi.org/10.1111/j.1475-1313.2004.00239.x, https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1475-1313.2004.00239.x
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778. IEEE, Las Vegas, NV, USA (Jun 2016). https://doi.org/10.1109/CVPR.2016.90, http://ieeexplore.ieee.org/document/7780459/

13. Huang, Z., Van Gool, L.J.: A Riemannian Network for SPD Matrix Learning. In: AAAI. vol. 1, p. 3 (2017)
14. Huang, Z., Wan, C., Probst, T., Van Gool, L.: Deep Learning on Lie Groups for Skeleton-based Action Recognition. arXiv:1612.05877 [cs] (Dec 2016), http://arxiv.org/abs/1612.05877, arXiv: 1612.05877
15. Huang, Z., Wu, J., Van Gool, L.: Building Deep Networks on Grassmann Manifolds. arXiv:1611.05742 [cs] (Nov 2016), http://arxiv.org/abs/1611.05742, arXiv: 1611.05742
16. Ionescu, C., Vantzos, O., Sminchisescu, C.: Matrix Backpropagation for Deep Networks with Structured Layers. In: 2015 IEEE International Conference on Computer Vision (ICCV). pp. 2965–2973. IEEE, Santiago, Chile (Dec 2015). https://doi.org/10.1109/ICCV.2015.339, http://ieeexplore.ieee.org/document/7410696/
17. Ionescu, C., Vantzos, O., Sminchisescu, C.: Training Deep Networks with Structured Layers by Matrix Backpropagation. arXiv:1509.07838 [cs] (Sep 2015), http://arxiv.org/abs/1509.07838, arXiv: 1509.07838
18. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images p. 60
19. Meghwanshi, M., Jawanpuria, P., Kunchukuttan, A., Kasai, H., Mishra, B.: McTorch, a manifold optimization library for deep learning. arXiv:1810.01811 [cs, stat] (Oct 2018), http://arxiv.org/abs/1810.01811, arXiv: 1810.01811
20. Miolane, N., Mathe, J., Donnat, C., Jorda, M., Pennec, X.: geomstats: a Python Package for Riemannian Geometry in Machine Learning. arXiv:1805.08308 [cs, stat] (May 2018), http://arxiv.org/abs/1805.08308, arXiv: 1805.08308
21. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch (Oct 2017), https://openreview.net/forum?id=BJJsrmfCZ
22. Pennec, X., Fillard, P., Ayache, N.: A Riemannian Framework for Tensor Computing. International Journal of Computer Vision **66**(1), 41–66 (Jan 2006). https://doi.org/10.1007/s11263-005-3222-z, http://link.springer.com/10.1007/s11263-005-3222-z
23. Yger, F.: A review of kernels on covariance matrices for BCI applications. In: 2013 IEEE International Workshop on Machine Learning for Signal Processing (MLSP). pp. 1–6 (Sep 2013). https://doi.org/10.1109/MLSP.2013.6661972
24. Yger, F., Sugiyama, M.: Supervised LogEuclidean Metric Learning for Symmetric Positive Definite Matrices. arXiv:1502.03505 [cs] (Feb 2015), http://arxiv.org/abs/1502.03505, arXiv: 1502.03505
25. Yu, K., Salzmann, M.: Second-order Convolutional Neural Networks. arXiv:1703.06817 [cs] (Mar 2017), http://arxiv.org/abs/1703.06817, arXiv: 1703.06817