McTorch, a manifold optimization library for deep learning

Mayank Meghwanshi
Microsoft, India
mamegh@microsoft.com

Pratik Jawanpuria
Microsoft, India
pratik.jawanpuria@microsoft.com

Anoop Kunchukuttan
Microsoft, India
anoop.kunchukuttan@microsoft.com

Hiroyuki Kasai
The University of Electro-Communications, Japan
kasai@is.uec.ac.jp

Bamdev Mishra
Microsoft, India
bamdevm@microsoft.com

Abstract

In this paper, we introduce McTorch, a manifold optimization library for deep learning that extends PyTorch[1]. It aims to lower the barrier for users wishing to use manifold constraints in deep learning applications, i.e., when the parameters are constrained to lie on a manifold. Such constraints include the popular orthogonality and rank constraints, and have been recently used in a number of applications in deep learning. McTorch follows PyTorch's architecture and decouples manifold definitions and optimizers, i.e., once a new manifold is added it can be used with any existing optimizer and vice-versa. McTorch is available at `https://github.com/mctorch`.

1 Introduction

Manifold optimization refers to nonlinear optimization problems of the form

$$\min_{x \in \mathcal{M}} f(x), \tag{1}$$

where f is the loss function and the parameter search space $\mathcal{M}$ is a smooth *Riemannian* manifold [1]. Note that the Euclidean space is trivially a manifold. Conceptually, manifold optimization translates the constrained optimization problem (1) into an unconstrained optimization over the manifold $\mathcal{M}$, thereby generalizing many of the standard nonlinear optimization algorithms with guarantees [1, 5, 27, 28, 33, 17, 7]. A few ingredients of manifold optimization are the matrix representations of the tangent space (linearization of the search space at a point), an inner product (to define a metric structure to compute the Riemannian gradient and Hessian efficiently), and a notion of a straight line (a characterization of the geodesic curves to maintain strict feasibility of the parameters). Two popular examples[2] of smooth manifolds are i) the Stiefel manifold, which is the set of $n \times p$ matrices whose columns are orthonormal, i.e., $\mathcal{M} := \{\mathbf{X} \in \mathbb{R}^{n\times p} : \mathbf{X}^\top \mathbf{X} = \mathbf{I}\}$ [9] and ii) the symmetric positive definite manifold, which is the set of symmetric positive definite matrices, i.e., $\mathcal{M} := \{\mathbf{X} \in \mathbb{R}^{n\times n} : \mathbf{X} \succ \mathbf{0}\}$ [4].

[1]PyTorch is available at `https://pytorch.org/` and introduced in [25].
[2]A comprehensive list of manifolds is at `http://manopt.org/tutorial.html`.

Preprint. Work in progress.

Manifold optimization has gained significant interest in computer vision [18, 30], Gaussian mixture models [10], multilingual embeddings [14], matrix/tensor completion [8, 15, 16, 19, 21, 23, 31], metric learning [20, 32], phase synchronization [6, 34], to name a few.

Deep learning refers to machine learning methods with multiple layers of processing to learn effective representation of data. These methods have led to state-of-the-art results in computer vision, speech, and natural language processing. Recently, manifold optimization has been applied successfully in various deep learning applications [22, 2, 26, 11, 13, 24, 3].

On the practical implementation front, there exists popular toolboxes – Manopt [8], Pymanopt [29], and ROPTLIB [12] – that allow rapid prototyping without the burden of being well-versed with manifold-related notions. However, these toolboxes are more suitable for handling standard nonlinear optimization problems and not particularly well-suited for deep learning applications. On the other hand, PyTorch [25], a Python based deep learning library, supports tensor computations on GPU and provides dynamic tape-based auto-grad system to create neural networks. PyTorch provides a flexible format to define and train deep learning networks. Currently, however, PyTorch lacks manifold optimization support. The proposed McTorch[3] library aims to bridge this gap between the standard manifold toolboxes and PyTorch by extending the latter's functionality.

McTorch builds upon the PyTorch library for tensor computation and GPU acceleration, and derives manifold definitions and optimization methods from the toolboxes [8, 12, 29]. McTorch is well-integrated with PyTorch that allows users to use manifold optimization in a straightforward way.

2 Overview of McTorch

McTorch library has been implemented by extending a PyTorch fork to closely follow its architecture. All manifold definitions reside in the module `torch.nn.manifold` and are derived from the parent class `Manifold`, which defines the manifold structure (i.e., the expressions of manifold-related notions) that any manifold must implement. A few of these expressions are:

- `rand`: to get a random point on the manifold,
- `retr`: to retract a tangent vector onto the manifold,
- `egrad2rgrad`: to convert the back-propagated gradient to the Riemannian gradient.

To facilitate creation of a manifold-constrained parameter, PyTorch's native `Parameter` class is modified to accept an extra argument on initialization to specify the manifold type and size. `Parameter` can be initialized to a random point on the manifold or a particular value provided by the user. `Parameter` also holds the attribute `rgrad` (which stands for the Riemannian gradient) that gets updated with every back-propagated gradient step.

The existing optimizers in the module `torch.optim` are modified to support updates on the manifold. An optimization step is a function of parameter's current value, gradient, and optimizer state. In the manifold optimization, the gradient is the Riemannian gradient and the update is with the retraction operation.

To use manifolds in PyTorch layers (in `torch.nn.Module`), we have added the property `weight_manifold` to the linear and convolutional layers which constrains the weight tensor of the layer to a specified manifold. As the shape of weight tensor is calculated using the inputs to the layer, we have added `ManifoldShapeFactory` to create a manifold object for a given tensor shape such that it obeys the initialization conditions of that manifold.

All the numerical methods are implemented using the tensor functions of PyTorch and support both CPU and GPU computations. As the implementation modifies and appends to the PyTorch code, all the user facing APIs are similar to PyTorch. McTorch currently supports:

- **Manifolds**: `Stiefel`, `PositiveDefinite`,
- **Optimizers**: `SGD`, `Adagrad`, `ConjugateGradient`,
- **Layers**: `Linear`, `Conv1d`, `Conv2d`, `Conv3d`.

[3]McTorch stands for **M**anifold-**c**onstrained **Torch**.

3 McTorch Usage

Orthogonal weight normalization in multi-layer perceptron [11]. An example showing creation and optimization of McTorch module with the Stiefel manifold.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

# A McTorch module using manifold constrained linear layers.
class OrthogonalWeightNormalizationNet(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size):
        super(OrthogonalWeightNormalizationNet, self).__init__()
        layer_sizes = [input_size] + hidden_sizes + [output_size]
        self.layers = []

        for i in range(1, len(layer_sizes)):
            self.layers.append(nn.Linear(in_features=layer_sizes[i-1],
                                         out_features=layer_sizes[i],
                                         weight_manifold=nn.Stiefel))
            # ReLU for middle layers
            if i != len(layer_sizes)-1:
                self.layers.append(nn.ReLU())
            # LogSoftmax at the output layer
            else:
                self.layers.append(nn.LogSoftmax(dim=1))
        self.model = nn.Sequential(*self.layers)

    def forward(self, x):
        return self.model(x)

# Create module object.
model = OrthogonalWeightNormalizationNet(input_size=1024,
        hidden_sizes=[128, 128, 128, 128, 128], output_size=68)

# Optimize with the Adagrad algorithm.
optimizer = torch.optim.Adagrad(params=model.parameters(), lr=1e-2)
for epoch in range(10):
    optmizer.zero_grad()
    data, target = get_next_batch()
    output = model(data)
    loss = F.nll_loss(output, target)
    cost.backward()
    optimizer.step()
```

In the above example, a new PyTorch module is defined by inheriting from `nn.Module`. It requires defining an `__init__` function to set up layers and a `forward` function to define forward pass of the module. The backward pass for back-propagation of gradients is automatically computed. In the layer definition, `Linear` layers with `Stiefel` (orthogonal) manifold are initialized. It also adds ReLU nonlinearity between layers and softmax nonlinearity on the output. The forward function of the model does a forward pass on sequence of layers. To optimize, `torch.optim.Adagrad` is initialized, which is followed by multiple epochs of forward and backward passes of the module on batched training data.

4 Roadmap and Conclusion

We are actively working on adding support for more manifolds and optimization algorithms. We are also curating a collection of code examples and benchmarks to showcase various uses of manifold optimization in deep learning research and applications.

McTorch is released under the BSD-3 Clause license and all the codes and examples are available on the GitHub repository of the project at `https://github.com/mctorch/mctorch`.

References

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.

[2] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *ICML*, 2016.

[3] V. Badrinarayanan, B. Mishra, and R. Cipolla. Symmetry-invariant optimization in deep networks. Technical report, arXiv preprint arXiv:1511.01754, 2015.

[4] R. Bhatia. *Positive definite matrices*, volume 24. Princeton university press, 2009.

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

[6] N. Boumal. Nonconvex phase synchronization. *SIAM Journal on Optimization*, 26(4):2355–2377, 2016.

[7] N. Boumal, P.-A. Absil, and C. Cartis. Global rates of convergence for nonconvex optimization on manifolds. *IMA Journal of Numerical Analysis*, 2018.

[8] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15(Apr):1455–1459, 2014.

[9] A. Edelman, T.A. Arias, and S.T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[10] R. Hosseini and S. Sra. Matrix manifold optimization for gaussian mixtures. In *NIPS*, 2015.

[11] L. Huang, X. Liu, B. Lang, A. W. Yu, Y. Wang, and B. Li. Orthogonal weight normalization: Solution to optimization over multiple dependent Stiefel manifolds in deep neural networks. In *AAAI*, 2017.

[12] W. Huang, P.-A. Absil, K. A. Gallivan, and P. Hand. ROPTLIB: an object-oriented C++ library for optimization on Riemannian manifolds. Technical Report FSU16-14.v2, Florida State University, 2016.

[13] Z. Huang, J. Wu, and L. Van Gool. Building deep networks on Grassmann manifolds. In *AAAI*, 2018.

[14] P. Jawanpuria, A. Balgovind, A. Kunchukuttan, and B. Mishra. Learning multilingual word embeddings in latent metric space: a geometric approach. Technical report, arXiv preprint arXiv:1808.08773, 2018.

[15] P. Jawanpuria and B. Mishra. A unified framework for structured low-rank matrix learning. In *ICML*, 2018.

[16] H. Kasai and B. Mishra. Low-rank tensor completion: a riemannian manifold preconditioning approach. In *ICML*, 2016.

[17] H. Kasai, H. Sato, and B. Mishra. Riemannian stochastic recursive gradient algorithm. In *ICML*, 2018.

[18] A. Kovnatsky, K. Glashoff, and M. M. Bronstein. Madmm: a generic algorithm for non-smooth optimization on manifolds. In *ECCV*, 2016.

[19] D. Kressner, M. Steinlechner, and B. Vandereycken. Low-rank tensor completion by Riemannian optimization. *BIT Numerical Mathematics*, 2013. Doi: 10.1007/s10543-013-0455-z.

[20] G. Meyer, S. Bonnabel, and R. Sepulchre. Linear regression under fixed-rank constraints: a Riemannian approach. In *ICML*, 2011.

[21] B. Mishra, G. Meyer, S. Bonnabel, and R. Sepulchre. Fixed-rank matrix factorizations and Riemannian low-rank optimization. *Computational Statistics*, 29(3–4):591–621, 2014.

[22] M. Nickel and D. Kiela. Learning continuous hierarchies in the Lorentz model of hyperbolic geometry. In *ICML*, 2018.

[23] M. Nimishakavi, P. Jawanpuria, and B. Mishra. A dual framework for low-rank tensor completion. In *NIPS*, 2018.

[24] M. Ozay and T. Okatani. Training CNNs with normalized kernels. In *AAAI*, 2018.

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *The NIPS workshop on the future of gradient-based machine learning software & techniques*, 2017.

[26] S. K. Roy, Z. Mhammedi, and M. Harandi. Geometry aware constrained optimization techniques for deep learning. In *CVPR*, 2018.

[27] H. Sato and T. Iwai. A new, globally convergent Riemannian conjugate gradient method. *Optimization: A Journal of Mathematical Programming and Operations Research*, 64(4):1011–1031, 2013.

[28] H. Sato, H. Kasai, and B. Mishra. Riemannian stochastic variance reduced gradient. *arXiv preprint: arXiv:1702.05594*, 2017.

[29] J. Townsend, N. Koep, and S. Weichwald. Pymanopt: A Python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research*, 17(137):1–5, 2016.

[30] R. Tron and K. Daniilidis. The space of essential matrices as a Riemannian quotient manifold. *SIAM Journal Imaging Sciences*, 10(3):1416–1445, 2017.

[31] B. Vandereycken. Low-rank matrix completion by Riemannian optimization. *SIAM Journal on Optimization*, 23(2):1214–1236, 2013.

[32] P. Zadeh, R. Hosseini, and S. Sra. Geometric mean metric learning. In *ICML*, 2016.

[33] H. Zhang, S. J. Reddi, and S. Sra. Riemannian svrg: Fast stochastic optimization on Riemannian manifolds. In *NIPS*, 2016.

[34] Y. Zhong and N. Boumal. Near-optimal bounds for phase synchronization. *SIAM Journal on Optimization*, 28(2):989–1016, 2018.