How to Implement Physics-based Learning

Michael Kellman [1] **Michael Lustig** [1] **Laura Waller** [1]

1. Introduction

Computational imaging systems (*e.g.* tomographic systems, computational optics, magnetic resonance imaging) jointly design software and hardware to retrieve information which is not traditionally accessible. Generally, such systems are characterized by how the information is encoded (forward process) and decoded (inverse problem) from the measurements. Critical aspects of computational imaging systems, such as experimental design and image priors, can be optimized through deep networks formed by *unrolling* the iterations of classical model-based reconstructions (Gregor & LeCun, 2010; Sun et al., 2016; Kellman et al., 2019a). Termed physics-based learning, this paradigm is able to incorporate knowledge we understand, such as the physics-based forward model and optimization methods, while relying on data-driven learning for parameters that we do not. A complete review the field can be found here (Ongie et al., 2020).

The goal of this tutorial is to explain step-by-step how to implement physics-based learning for the rapid prototyping of a computational imaging system. We provide a basic overview of physics-based learning, the construction of a physics-based network, and its reduction to practice. Specifically, we advocate exploiting the auto-differentiation functionality (Griewank & Walther, 2008) twice, once to build a physics-based network and again to perform physics-based learning. Thus, the user need only implement the forward model process for their system, speeding up prototyping time. We provide an open-source Pytorch (Paszke et al., 2017) implementation of a physics-based network and training procedure for a generic sparse recovery problem (Sec. 3).

2. Background

Computational imaging systems are described by how sought information is encoded to and decoded from a set of measurements. The encoding of information, $\mathbf{x}$, into measurements, $\mathbf{y}$, is given by

$$\mathbf{y} = \mathcal{A}(\mathbf{x}) + \mathbf{n}, \tag{1}$$

where $\mathcal{A}$ describes the forward model process that characterizes the formation of measurements and $\mathbf{n}$ is random system noise. The image reconstruction from a set of measurements, *i.e.* decoding, can be structured using an inverse problem formulation,

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}} \underbrace{\|\mathcal{A}(\mathbf{x}) - \mathbf{y}\|^2}_{\mathcal{D}(\mathbf{x};\mathbf{y})} + \mathcal{P}(\mathbf{x}), \tag{2}$$

where $\mathbf{x}$ is the sought information, $\mathcal{D}(\cdot)$ is the data consistency penalty (commonly ℓ_2 distance between the measurements and the estimated measurements), and $\mathcal{P}(\cdot)$ is the signal prior (*e.g.* sparsity, total variation). This optimization problem often requires a non-linear and iterative solver. Proximal gradient descent can efficiently solve the optimization problem in the case of a non-linear signal prior (Parikh & Boyd, 2014). Methods such as Alternating Direction Method of Multipliers (ADMM) (Boyd et al., 2011) and Half-Quadratic Splitting (HQS) (Geman & Yang, 1995) can be efficient solvers when multiple constraints are placed on the image reconstruction and the forward model process is linear.

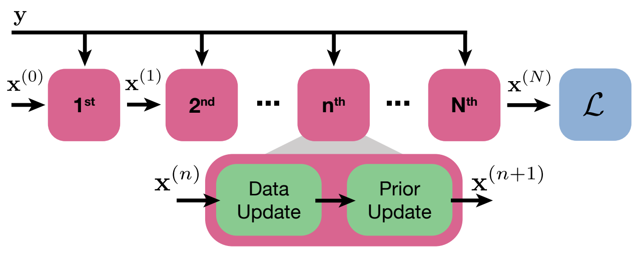

Figure 1. Unrolled Physics-based network composed of N unrolled decoder iterations. Each layer is comprised of a data consistency update (*e.g.* gradient update) and a prior update (*e.g.* proximal update). The measurements, $\mathbf{y}$, and initialization, $\mathbf{x}^{(0)}$, serve as the network's inputs and the final estimate, $\mathbf{x}^{(N)}$, as the network's output. Finally, the final estimate is fed into some loss function, $\mathcal{L}$, for training.

The physics-based network (Fig. 1) is then formed by unrolling N iterations of the optimization algorithm into the

[1]Electrical Engineering and Computer Sciences, University of California, Berkeley, USA. Correspondence to: Michael Kellman <kellman@berkeley.edu>.

layers of a network. The inputs to the network are the measurements and initialization and the output is an estimate of the information after N iterations of the optimizer.

3. Implementation

For the purposes of demonstration, we implement physics-based learning on a compressed sensing problem – underdetermined sparse recovery from linear Gaussian random measurements. The sparse signals have normally-distributed amplitude values. The learnable parameters in the physics-based network will be the measurement matrix, the sparsity prior penalty, and the step size of the optimizer. Specifically, we solve the ℓ_1 relaxation of the sparse recovery problem,

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2 + \lambda\|\mathbf{x}\|_1, \quad (3)$$

where λ is the sparsity prior penalty that trades off the penalty of the prior with the data consistency term, and $\mathbf{A} \in \mathbb{R}^{J\times K}$ is the linear measurement matrix.

We use the proximal gradient descent algorithm (Alg. 1) to solve the optimization problem (Eq. 3) and to form the architecture of the physics-based network.

Algorithm 1 Proximal Gradient Descent

Inputs $\mathbf{x}^{(0)}$-initialization, α-step size, N-number of iterations, $\mathbf{y}$-measurements
Output $\mathbf{x}^{(N)}$-final estimate of image

```
1: $n \leftarrow 0$
2: for $n < N$ do
3:     $\mathbf{z}^{(n)} \leftarrow \mathbf{x}^{(n)} - \alpha\nabla_{\mathbf{x}}\mathcal{D}(\mathbf{x}^{(n)}; \mathbf{y})$
4:     $\mathbf{x}^{(n+1)} \leftarrow \texttt{soft\_thr}_{\alpha\lambda}(\mathbf{z}^{(n)})$
5:     $n \leftarrow n + 1$
6: end for
```

Physics-based network

With some extra Pytorch specific flags and functionalities, the physics-based network (Fig. 2) mirrors the basic structure of Alg. 1. In implementation, it requires all learnable parameters as input (measurement matrix, step size, and sparsity penalty), initialization, ground truth, and number of iterations (or depth of network). As output, the network returns the final estimate of the sparse recovery optimization.

The specific Pytorch flags and functionalities that enable us to properly use the automatic differentiator for sparse recovery and for physics-based learning are contained in lines 14, 15, and 26. Lines 14 and 15 set up the initialization to the network, first in line 14 detaching it from any previous automatic differentiation graphs and second in line 15 setting its *requires_grad* field to be True. This will allow the automatic differentiation to track operations related to $\mathbf{x}$ for taking derivatives of $\mathcal{D}(\mathbf{x}; \mathbf{y})$ with respect to $\mathbf{x}$. Line 26 sets

```
 1 def pbnet(A, alpha, lamb, x0, xt, K, testFlag=True):
 2     # pbnet - Physics-based Network
 3     # args in -
 4     #   A - measurement matrix
 5     #   alpha - step size
 6     #   lamb - sparsity penalty
 7     #   x0 - initialization
 8     #   xt - ground truth sparse vector
 9     #   K - number of layers (i.e. iterations)
10     #   testFlag - disables training (default True)
11     # args out -
12     #   x - output of network (final estimate)
13 
14     x = x0.detach().clone()
15     x.requires_grad = True
16 
17     y_meas = Aop(A, xt)
18     if testFlag: y_meas = y_meas.detach()
19 
20     for kk in range(K):
21         y_est = torch.matmul(A,x)
22         res = y_est - y_meas
23         loss_dc = torch.sum(res**2)
24         g = torch.autograd.grad(loss_dc,
25                                 x,
26                                 create_graph = not testFlag)
27 
28         x = x - alpha*g[0]           # gradient update
29         x = softthr(x, lamb*alpha)   # proximal update
30     return x
```

Figure 2. Implementation of a physics-based network for compressed sensing sparse recovery using automatic differentiation to compute gradients with respect to $\mathbf{x}$. The network is setup to learn the measurement matrix, step size, and sparsity penalty. Soft thresholding is used as the proximal operator.

the *create_graph* flag of the automatic differentiator to True. This tells Pytorch to store these operations so that they can be traced through by a second automatic differentiator for computing derivatives with respect to learnable parameters at training time.

While some physics-based networks take in measurements as input, this is often not possible when learning a system's experimental design. Here, in line 17, the measurements are synthesized using the learnable measurement matrix and ground truth sparse vector.

Physics-based Learning

The mechanics of training a physics-based network are similar to that of training any neural net or convolutional neural network in that it relies on a dataset, an optimizer, and automatic differentiation to compute gradients. In our particular application, our dataset consists of sparse vectors with amplitudes that are normally distributed. As for the optimizer, we use the adaptive moment estimation (Adam) (Kingma & Ba, 2014) algorithm to perform the gradient updates. For researchers that interested in learning for large-scale computational imaging systems, automatic differentiation will require more memory than available on commercial graphical processing units. To enable learning at these scales, vanilla automatic differentiation can be replaced with memory-efficient techniques (Kellman et al., 2019b).

4. Remarks

In this tutorial we overview the implementation of physics-based learning in Pytorch with a simple open-source implementation that provides a minimum working example for those looking to optimize their own system. The concepts discussed here generalize to vast sets of applications and optimization algorithms.

References

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Geman, D. and Yang, C. Nonlinear image recovery with half-quadratic regularization. *IEEE transactions on Image Processing*, 4(7):932–946, 1995.

Gregor, K. and LeCun, Y. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 399–406, 2010.

Griewank, A. and Walther, A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2008. ISBN 0898716594, 9780898716597.

Kellman, M., Bostan, E., Repina, N., and Waller, L. Physics-based learned design: Optimized coded-illumination for quantitative phase imaging. *IEEE Transactions on Computational Imaging*, 5(3):344–353, 2019a.

Kellman, M., Tamir, J., Boston, E., Lustig, M., and Waller, L. Memory-efficient learning for large-scale computational imaging. *arXiv preprint arXiv:1912.05098*, 2019b.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ongie, G., Jalal, A., Metzler, C. A., Baraniuk, R. G., Dimakis, A. G., and Willett, R. Deep learning techniques for inverse problems in imaging. *arXiv preprint arXiv:2005.06001*, 2020.

Parikh, N. and Boyd, S. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Sun, J., Li, H., Xu, Z., et al. Deep ADMM-Net for compressive sensing MRI. In *Advances in Neural Information Processing Systems*, pp. 10–18, 2016.