Journal of Machine Learning Research 1 (2000) 1-48 Submitted 4/00; Published 10/00

# FuSplit LBI Toolbox – Fudan Split Linearized Bregman Iteration Deep Network Pytorch Optimizer

**Chen Liu** 18210980009@fudan.edu.cn
**Yanwei Fu*** yanweifu@fudan.edu.cn
**Xiangyang Xue** xyxue@fudan.edu.cn
*School of Data Science, Fudan University*
**Yuan Yao** yuany@ust.hk
Hong Kong University of Science and Technology

**Editor:**

## Abstract

We present the Fudan Split Linearized Bregman Iteration toolbox (FuSplit LBI Toolbox), which offers a strong and versatile functionality for training of Deep Neural Networks (DNNs). The FuSplit LBI extends the Split Linearized Bregman Iteration (SLBI) algorithm in linear model (Zhao et al. (2018)) to learn the parameters of deep networks. The key novelty of our FuSplit LBI lies in model selection consistency of learning the structural sparsity of network, with the comparable computational cost to Stochastic Gradient Descendent (SGD) and SGD variants. In our recent technical report (Fu et al. (2019)), we found that an iterative regularization path with structural sparsity derived from SLBI, can help prune or grow the network structures. The FuSplit LBI Toolbox is based on Optimizer Class of Pytorch; and it can be used with Pytorch code for training DNNs seamlessly. The codes will be downloadable from our github: `http://yanweifu.github.io/FuSplitLBI/index.html`.

**Keywords:** Split Linear Bregman Iteration, Network Training, Model Compression

## 1. Introduction

To train the DNNs, one may apply the optimizers such as SGD and its variants. In contrast, this paper presents to train Deep Neural Networks (DNNs), a novel strong and versatile optimizer toolbox – Fudan Split Linearized Bregman Iteration toolbox (FuSplit-LBI Toolbox). The FuSplit-LBI extends to train deep networks, the Split Linear Bregman Iteration (SLBI) (Huang et al. (2016)) which is a powerful optimization algorithm that can catch the structure sparsity of learned model parameters.

As proposed in (Huang et al. (2016)), SLBI can learn the structure sparsity over the parameter set. The sparse constraint is given to split variables and then link the original variables and split variables. To illustrate the concrete process, we firstly make some statements for notations here. In this paper, $X$ and $Y$ denote the inputs and output variables respectively; $Z$ and $S$ represent the split, and sparse split variables of weights, individually. $N$ stands for the number of samples and $W$ demonstrates the weights of model. Thus the

*. Corresponding author.

optimization problem with the sparse constraints is defined as follows.

$$\underset{W}{\operatorname{argmin}} \frac{1}{2N} L(Y, X|W) + \beta \|W\|_1 \tag{1}$$

where we have the differentiable loss function $L$ , and $\beta$ indicates the coefficient of $L_1$ penalty term. As in Huang et al. (2016), the objective function $L_{slbi}$ of SLBI is defined as

$$\underset{W}{\operatorname{argmin}} \frac{1}{2N} L(Y, X|W) + \beta \|S - W\|_2^2 \tag{2}$$

Here the sparse constraints are enforced on $S$. The optimization process for Equation 2 is iteratively solved by the following equations. The loss function of Equation 2 is denoted as $L_{slbi}$; and $t$ refers to the number of iteration.

$$W^{t+1} = W^t - \kappa\alpha\nabla_W L_{slbi} \tag{3}$$

$$Z^{t+1} = Z^t - \alpha\nabla_Z L_{slbi} \tag{4}$$

$$S^{t+1} = \kappa \cdot prox_{\|\cdot\|_1}(Z^{t+1}) \tag{5}$$

Where $\alpha$, $\kappa$, and $prox_{\|\cdot\|_1}$ denote the learning rate, the coefficient and the proximal mapping, individually. In all previous works Zhao et al. (2018); Huang et al. (2016), the SLBI is utilized in optimizing the linear model set $W$, while this paper, for the first time, provides a versatile toolbox in learning the parameter set $W$ from DNNs.

## 2. Features of FuSplit-LBI Toolbox

The FuSplit-LBI Toolbox can be obtained from our project page[1]. This toolbox derives from Optimizer Class of Pytorch (Team (2017)); it can be directly utilized to replace the build-in Pytorch Optimizers such as SGD (Bottou (2010)) and Adam (Kingma and Ba (2014)). Currently this toolbox only supports training DNNs using Pytorch (Team (2017)), and the extension on other deep learning frameworks, e.g., TensorFlow (Abadi et al. (2016)) will be released as the future work. Our toolbox has very good compatibility: it only depends on two commonly used packages – Pytorch (Team (2017)) and Numpy (Oliphant (2006–)). With these two packages installed, one can facilitate FuSplit-LBI Toolbox in training DNNs either with CPUs or GPUs.

The names of layers are recorded as meta-data to help the learning process of our toolbox. Specifically, as an extension of Pytorch's (Team (2017)) Optimizer Class, FuSplit-LBI Toolbox is an improved version of the original Class. The original Class uses python list as input and ignores layer names, our toolbox adds parameter states to store layer names of model parameters. To this end, the FuSplit-LBI Toolbox can access the parameters of layers by their names. When analyzing the particular layers or pruning a series of layers, users can directly employ their names of corresponding layers. In our example codes, such a trick can significantly help facilitate the manipulating the layers of DNNs, especially the large depth networks such as ResNet (He et al. (2016)) or DenseNet (Huang et al. (2018)).

Further, the information of gradients can also be saved as the meta-data by default in our toolbox. In practice, this information is also very useful. For example, sometimes one

1. `http://yanweifu.github.io/FuSplitLBI/index.html`

| Dataset | *MNIST* | *Cifar10* | *ImageNet* | |
|---|---|---|---|---|
| Models | *Lenet* | *ResNet-20* | *AlexNet* | *ResNet-18* |
| *SGD*$^\star$ (Bottou (2010)) | 0.9921/– | 0.9125/– | 0.5655/0.7909 | 0.6976/0.8918 |
| *Adam*$^\star$ (Kingma and Ba (2014)) | 0.9919/– | 0.9086/– | –/– | 0.6770/0.8667 |
| *Our Toolbox* (Zhao et al. (2018)) | 0.9919/– | 0.9157/– | 0.5609/0.7886 | 0.6855/0.8785 |

Table 1: The Top-1/Top-5 accuracy comparison between SGD and our Split LBI on each dataset. $^\star$: we use the pytorch implementation. All models are trained with 100 epochs.

may be failed to train the DNNs; and the gradients may be the key checkpoints that one may want to analyze. In our FuSplit-LBI Toolbox, the important statistics of gradients can be recorded at the given interval for appointed layers. Then these data can be processed for the further usage.

Our FuSplit-LBI supports more advanced applications. For example, with the increase of model complexity of DNNs, more and more efforts are made on the model compression. One early proposed approach is the Biased Weight Decay (Hanson and Pratt (1989)). Recently, HashedNets model is proposed by (Chen et al. (2015)) to group the models into buckets and perform weights sharing. (Cheng et al. (2017)) gives a good summary of this area. In our toolbox, considering the speed and memory cost, our toolbox thus supports our recent network pruning algorithm (Fu et al. (2019)).

## 3. Experiments on FuSplit-LBI Toolbox

Our FuSplit-LBI Toolbox offers a strong and versatile functionality for training of Deep Neural Networks, with faster convergence rate, and better sparse constraints added.

To show this point, we conduct the experiments on three classical datasets: MNIST (LeCun et al. (2010)), Cifar10 (Krizhevsky et al. (2014)) and ImageNet (Deng et al. (2009)). We utilize the two typical networks – Lenet (LeCun et al. (2015)) and ResNet (He et al. (2016)). We show that our FuSplit-LBI Toolbox can converge faster than SGD (Bottou (2010)) with momentum. In particular the generalization ability of our trained model is similar as SGD (Bottou (2010)). Detailed comparison is given is Table 1. Note that on all these datasets, we strictly follow the standard supervised learning split specified in each dataset.

To train ResNet-18 on ImageNet-2012 dataset, the model trained by FuSplit-LBI toolbox can achieve the Top-1 accuracy of 67.76 by 40 epochs, which is 2% lower than the best ResNet-18 results report in He et al. (2016) optimized by 120 epochs. This indicates that our toolbox have a much faster convergence than SGD. In Fig. 1c, we give a comparison on validation data of models trained by 30 epochs. It clearly shows faster convergence rate of our toolbox within the same number of training epochs as SGD.

Our toolbox will save the values of $Z^{t+1}$ and $S^{t+1}$, which requires extra $7\% - 8\%$ more GPU usage than that of SGD. In Table 2, we give a comparison of time and memory cost between SGD (Bottou (2010)) and our toolbox. Results in Table 2 are reported by using


(a) Loss Curve (b) Training Accuracy Curve (c) Validation Accuracy Curve

Figure 1: The left one shows the training loss curve by SGD and SLBI; the middle figure shows training accuracy; and the right one shows the validation accuracy.

| *Optimizer* | *Time Per 1k Iterations* | *GPU Memory Usage* |
|---|---|---|
| *SGD* | 750s | 10132MB |
| *Our Toolbox* | 780s | 10874MB |

Table 2: The time and memory cost of SGD and our toolbox on ResNet-18 trained on ImageNet. All models are trained with 100 epochs.

the ResNet-18 (He et al. (2016)) on ImageNet (Deng et al. (2009)) with the batch size of 256.

FuSplit-LBI Toolbox can get better sparsity than Lasso (Tibshirani (1996)) and weight decay. The visualizations of layers are shown in Fig. 3. The tuning process of FuSplit-LBI Toolbox is also user-friendly. In most circumstances, only the learning rate $\alpha$ and balance coefficient $\mu$ should be tuned. In our experiments, $\mu$ is determined by the epochs of training, the iterations per epoch and model complexity. To train the Lenet on MNIST, we set $\alpha = 0.1$ and $\mu = 100$.

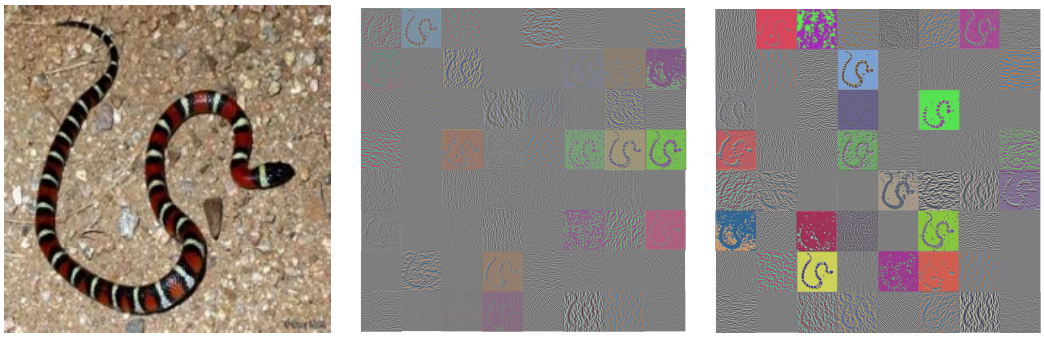
**Momentum based SLBI.** Inspired by the variants of SGD, we, for the first time, apply the momentum term to the Split LBI algorithm in this paper. An implementation is also provided in our toolbox. Particularly, our Toolbox adds the momentum in the training process, and updates the model parameters. The update formulas are shown as below.

$$v^{t+1} = \nabla_W L_{slbi} + \rho v^t \tag{6}$$
$$W^{t+1} = W^t - \kappa\alpha v^{t+1} L_{slbi} \tag{7}$$
$$Z^{t+1} = Z^t - \alpha \nabla_Z L_{slbi} \tag{8}$$
$$S^{t+1} = \kappa \cdot prox_{\|\cdot\|_1}(Z^{t+1}) \tag{9}$$

Where $\rho$ is the momentum factor, empirically setting as 0.9.

**Visualization of learned parameters of SLBI**. To further validate our toolbox , we offer an example for training the AlexNet(Krizhevsky et al. (2012)) on ImageNet Dataset.

The first convolutional filters are trained and visualized in Figure 2. This example proves the capability of our toolbox on complex tasks.



(a) Input picture (b) Visualization for SGD (c) Visualization for SLBI

Figure 2: The left one is the input image, the middle one and the right one are visualization of first convolutional layer of AlexNet using gradient backpropogation(Springenberg et al. (2014)). It shows that the filters learned by our toolbox have clearer observed patterns than those of learned by SGD. We use the parameter: $\kappa = 1, \mu = 500, lr = 1e-2, \rho = 0.9$.

## 4. Example: Training of Lenet on MNIST Dataset

To illustrate how to use FuSplit-LBI Toolbox, we will present one demo on training Lenet (LeCun et al. (2015)) on MNIST (LeCun et al. (2010)). It is a widely used dataset with 60,000 grayscale handwritten digits that belong to 10 classes (0 - 9).

This dataset is already split into train set and test set with 50,000 and 10,000 samples respectively. Lenet is a classical DNN proposed by Yann Lecun. It has 3 convolution layers and 2 fully connected layers. The construction of Lenet(LeCun et al. (2015) )is easy, so we will not show details about the model.

To load the package and initialize it, only two lines are needed.

```
from slbi import SLBI
optimizer = SLBI(model.parameters(), lr=1e-1, kappa=1, mu=100)
```

Where slbi is our python file. It is install-free, so usage is not affected by which platform users work on. As mentioned in Sec. 2, we add the name tags to corresponding layers; and this functionality should be initialized as follows:

```
name_list = [ ]
for name, p in model.named_parameters():
    name_list.append(name)
optimizer.assign_name(name_list)
```

By using this function the corresponding name will be saved in the *parameter_state* with dict name "*name*". When this step is done, we can perform the training of the pre-defined model. The example is shown as follows.

```
if (iter + 1) % record_interval == 0:
    optimizer.step(record=True, path='lenet/')
else:
    optimizer.step(record=False, path='lenet/')
```

This record function will write the statistics into txt files for further analysis. In the txt file, the statistics are recorded with layers' names, so it is convenient to get interested ones. By the way, if the users have some other demands for model analysis, the record function can be modified just a little to satisfy customized demand.

When one training epoch ends, we can update the prune order, the prune order is required by the prune function. So if users want to prune the model, this update function must be called after training one epoch.

```
optimizer.update_order(ep)
```

Optionally, if users want to test the results of the split variables, we can use the following codes. The second line aims at recovering the original parameters.

```
optimizer.use_w_star()
optimizer.recover()
```

Finally, to prune the layers to reduce the number of parameters as Fu et al. (2019), the following code should be invoked.

```
optimizer.prune_layer(percent, layer_name, prune_bias)
```

Here *percent* means the percent to be pruned with the value range of $[0\%, 100\%]$. The parameters will be pruned according to descending order of the values obtained from *optimizer.update_order.layer_name* means the layer to prune. It can be a list or a string. The names should be in accordance with the names assigned by using *optimizer.assign_name*. *prune_bias* determines whether the corresponding biases are also set to be zeros.

## 5. Summary

In this paper, we introduce our Fudan Split Linearized Bregman Iteration toolbox (FuSplit LBI Toolbox), which implements a series of SLBI functions in efficiently and effectively learning the model parameters in Pytorch. In particular, we, for the first time, present such a toolbox that facilitates training DNNs by SLBI and SLBI with momentum. The codes and example models will be released to the communities.

**Acknowledgement**. We thanks for the helps from Yuming Sun, Hangyu Lin, Donghao Li, Shun Zhang, and Zuyuan Zhong.

## References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

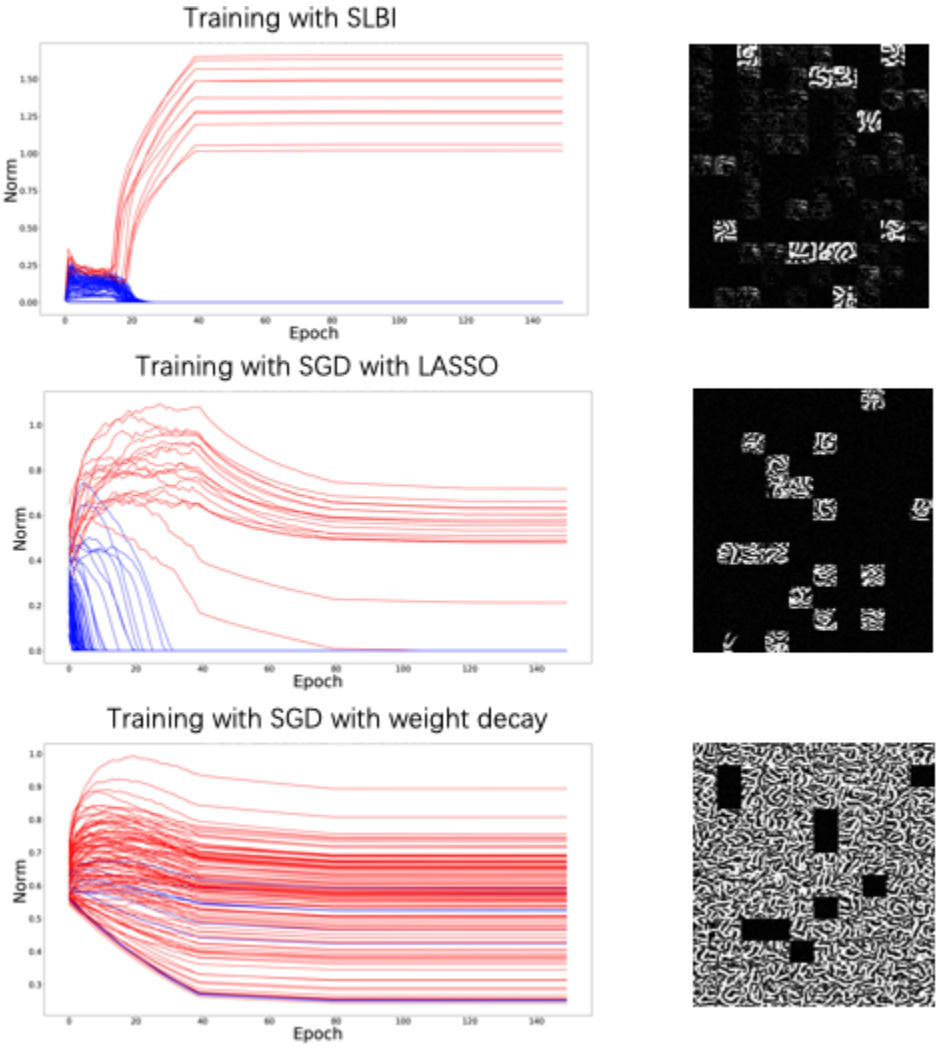



Figure 3: The learned structural sparsity of our FuSplit-LBI. Comparison between different optimizers on MNIST datasets. Results come from Fu et al. (2019).

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Yanwei Fu, Donghao Li, Xinwei Sun, Shun Zhang, Yizhou Wang, and Yuan Yao. $S^2$-lbi: Stochastic split linearized bregman iterations for parsimonious deep learning. In *in submission*, 2019.

Stephen José Hanson and Lorien Y Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*, pages 177–185, 1989.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Chendi Huang, Xinwei Sun, Jiechao Xiong, and Yuan Yao. Split lbi: An iterative regularization path with structural sparsity. In *Advances In Neural Information Processing Systems*, pages 3369–3377, 2016.

Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2752–2761, 2018.

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

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2:18, 2010.

Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20, 2015.

Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. URL `http://www.numpy.org/`.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Pytorch Core Team. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration, 2017.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Bo Zhao, Xinwei Sun, Yanwei Fu, Yizhou Wang, and Yuan Yao. Msplit lbi: Realizing feature selection and dense estimation simultaneously in few-shot and zero-shot learning. In *ICML*, 2018.