Training DNNs in Tiny Subspaces

Introductory Talk | Janik Philipps

Paper

Low Dimensional Landscape Hypothesis is True: DNNs can be Trained in Tiny Subspaces

Tao Li[1] Lei Tan[1] Qinghua Tao[2] Yipeng Liu[3] Xiaolin Huang*[1]
[1]Shanghai Jiao Tong University [2]Tsinghua University
[3]University of Electronic Science and Technology

Motivation: Deep neural networks depend on million of parameters causing severe problems

Models	# Parameter
VGG11_bn [41]	28.5M
EfficientNet-B0 [47]	4.14M
MobileNet [20]	3.3M
DenseNet121 [22]	7.0M
Inceptionv3 [46]	22.3M
Xception [7]	21.0M
GoogLeNet [45]	6.2M
ShuffleNetv2 [35]	1.3M
SequeezeNet [23]	0.78M
SEResNet18 [21]	11.4M
NasNet [53]	5.2M

- Potential overfitting of the data
- Only first order methods applicable
- Computationally and time intensive training
- Many training data needed
- Questionable solution quality

Motivation: The number of independent optimization variables may be smaller than the number of parameters

- Due to strong mutual relationships, **regarding each parameter** of deep neural networks **as an independent variable is too rough**
- The **gradients of parameters are strongly related** due to the training via backpropagation
- The parameters in the same layers also have **synergy correlations**

$$\downarrow$$

$$\boxed{\text{"DNNs can be trained in low-dimensional subspaces"}}$$

Motivation: Pioneering work achieved 90% accuracy in smaller spaces

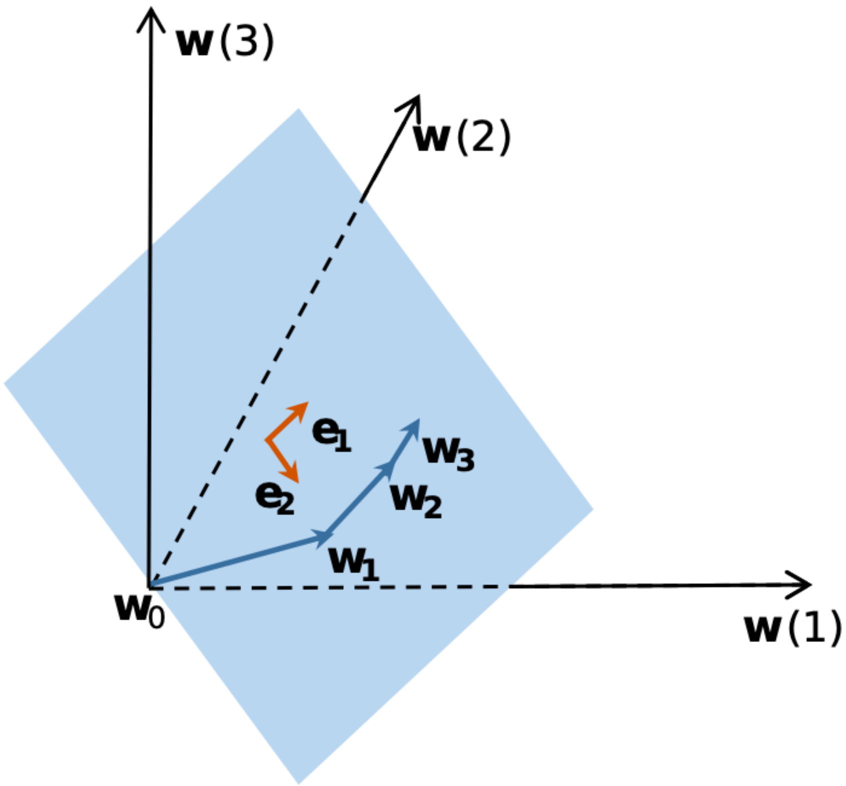
- There exists pioneering work on **training via random projections**
- For example, on CIFAR-10, LeNet with 62006 parameters could be optimized in 2900-dimensional subspaces with **90% accuracy of regular SGD training**.

$$\downarrow$$

Very promising but we can do even better!

Many standard neural network architectures could be **well trained by only 40 independent variables** with almost the same performance.

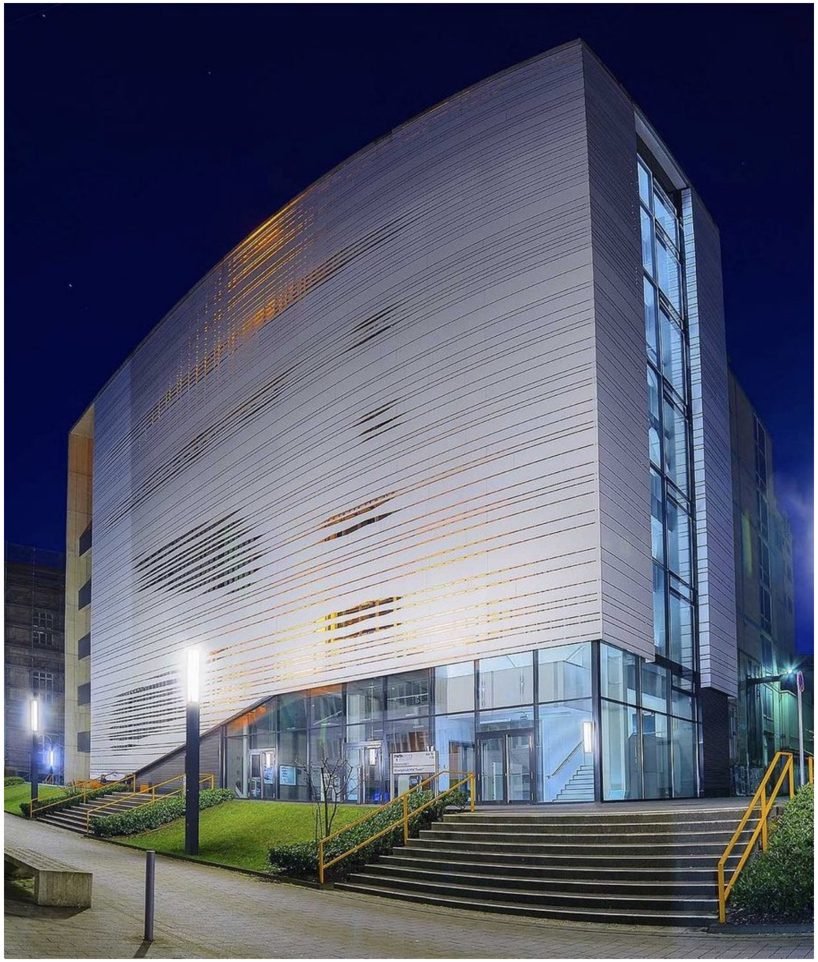
Approach in the paper



- There are **three parameters** $w(1)$, $w(2)$, $w(3)$ **to optimize**.
- The training trajectory $\{w_i\}_{i=0,\dots,t}$ could be in a **two-dimensional subspace** spanned by e_1 and e_2
- Training in the low-dimensional space can have **comparable performance** as training in the regular space

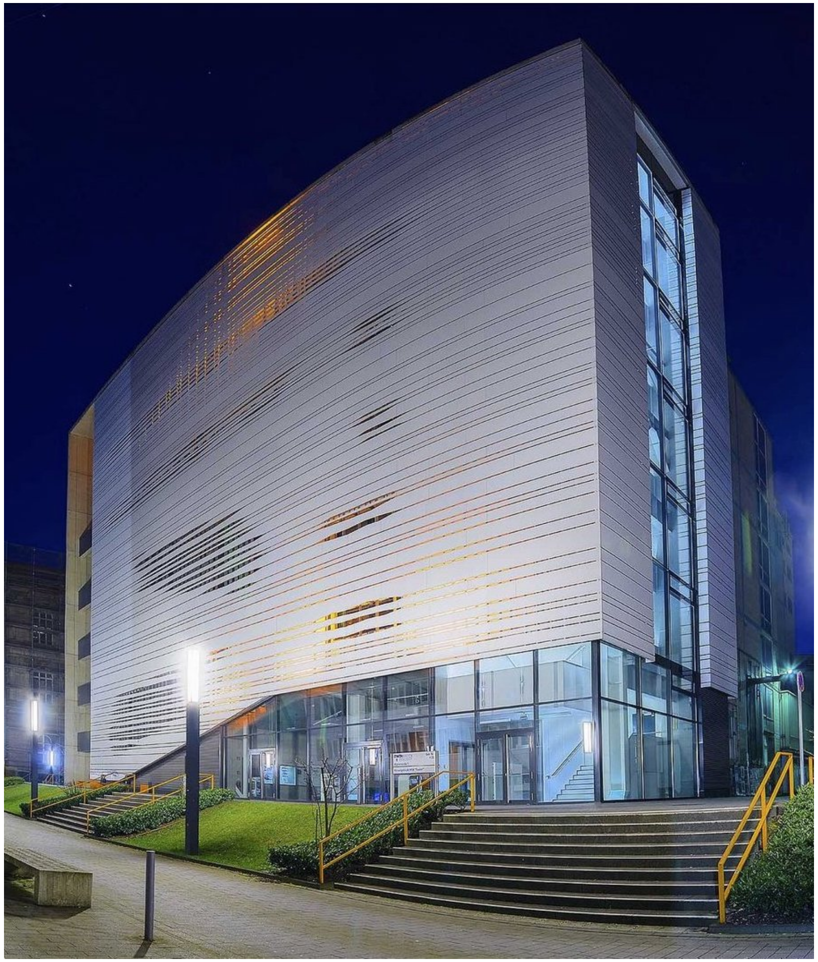
RWTH AACHEN UNIVERSITY

Agenda



- Dynamic Linear Dimensionality Reduction (DLDR)
- DLDR-based Quasi-Newton Algorithm
- Numerical Experiments

Agenda



- **Dynamic Linear Dimensionality Reduction (DLDR)**
- DLDR-based Quasi-Newton Algorithm
- Numerical Experiments

Dynamic Linear Dimensionality Reduction

$\boxed{\text{Assumption: The layer width is unlimited.}}$

- Under infinite-width limit, a wide neural network estimator can be approximated by a linear model under gradient descent, such that

$$f^{lin}(x, w_t) \approx f(x, w_0) + \nabla_w f(\mathcal{X}, w_0)(w_t - w_0)$$

- We formulate the gradient flow of a single-output neural network:

$$\dot{w}_t = -\nabla_w f(\mathcal{X}, w_t)^T \nabla_{f_t(\mathcal{X}, w_t)} \mathcal{L}$$

- Using the linearized model, the dynamics of gradient flow are governed by

$$\dot{w}_t = -\nabla_w f(\mathcal{X}, w_0)^T \nabla_{f^{lin}(\mathcal{X}, w_t)} \mathcal{L}$$

Dynamic Linear Dimensionality Reduction

- Applying Singular Value Decomposition on $\nabla_w f(\mathcal{X}, w_0)$ yields

$$\nabla_w f(\mathcal{X}, w_0) = U_0 \Sigma_0 V_0^T \in \mathbb{R}^{m \times n}$$

- By definition of the Neural Tangent Kernel, we have

$$\Theta_0 = \nabla_w f(\mathcal{X}, w_0) \nabla_w f(\mathcal{X}, w_0)^T = U_0 \Sigma_0 \Sigma_0^T U_0^T$$

- Based on the properties of NTK in infite-width limit, we can approximate Σ_0 by a low-rank matrix $\tilde{\Sigma}_0 \in \mathbb{R}^{d \times d}$ containing the first d largest singular values, such that

$$\Sigma_0 \approx \tilde{U}_0 \tilde{\Sigma}_0 \tilde{V}_0^T$$

- Thus we derive the approximations

$$\nabla_w f(\mathcal{X}, w_0) \approx U_0 \tilde{U}_0 \tilde{\Sigma}_0 \tilde{V}_0^T V_0^T$$

$$\dot{w}_t \approx -V_0 \tilde{V}_0 \left(\tilde{\Sigma}_0 \tilde{U}_0^T U_0^T \nabla_{f^{lin}(\mathcal{X}, w_t)} \mathcal{L} \right)$$

Dynamic Linear Dimensionality Reduction

$\boxed{\text{How to find the low-dimensional subspace?}}$

- 1) Sample t steps of parameters during the training, namely, $\{w_1, w_2, \ldots, w_t\}$.
- 2) Centralize these as $\bar{w} = \frac{1}{t}\sum_{i=1}^{t} w_i$ and let $W = [w_1 - \bar{w}, \ldots, w_t - \bar{w}]$.
- 3) Find a d-dimensional subspace spanned by $P = [e_1, e_2, \ldots, e_d]$ to cover W.

The third step is to find a subspace that the distance of W and $P^T W$ is minimized.

- We consider the SVD $W = U\Sigma V^T$ with $U = [u_1, \ldots, u_n]$, $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_t)$ and $V = [v_1, \ldots, v_t]$. The first d columns of U are the independent variables.
- Compute v_i for $i = 1, \ldots, d$ by the spectral decomposition of $W^T W$ and derive

$$Wv_i = \sigma_i u_i, \quad i = 1, \ldots, d.$$

Dynamic Linear Dimensionality Reduction

Algorithm 1 Dynamic Linear Dimensionality Reduction (DLDR)

1: Sample parameter trajectory $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_t\}$ along the training;
2: $\overline{\mathbf{w}} = \frac{1}{t}\sum_{i=1}^{t} \mathbf{w}_i$;
3: $W = [\mathbf{w}_1 - \overline{\mathbf{w}}, \mathbf{w}_2 - \overline{\mathbf{w}}, \ldots, \mathbf{w}_t - \overline{\mathbf{w}}]$;
4: Perform spectral decomposition on $W^\top W$ and obtain the largest d eigenvalues $[\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2]$ with the corresponding eigenvectors $[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d]$;
5: $\mathbf{u}_i = \frac{1}{\sigma_i} W \mathbf{v}_i$;
6: Return $[\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d]$ as the orthonormal bases.

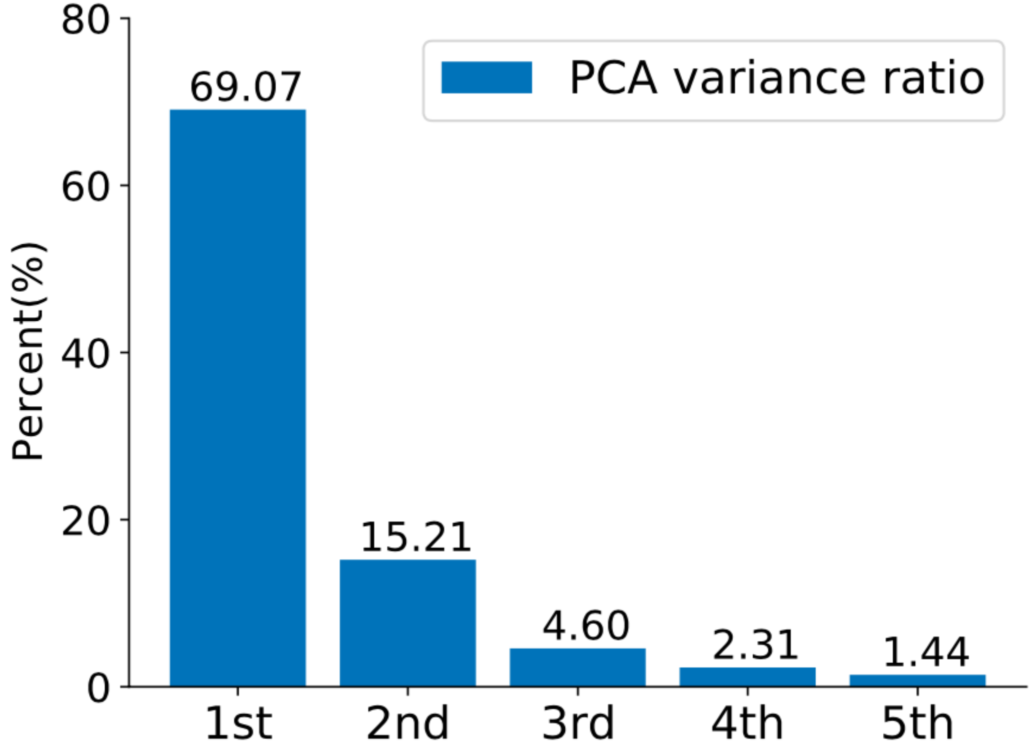
Dynamic Linear Dimensionality Reduction

Setting: learning rate 0.1, batch size 128, SGD with momentum of 0.9, 3 trials total

dimension reduction method	dimension	test accuracy
—	78330	83.84±0.34
Li *et al.* [31]	7982	58.35±0.04
Gressmann *et al.* [15]	7982	70.26±0.02
DLDR (ours)	**15**	**85.18±0.20**

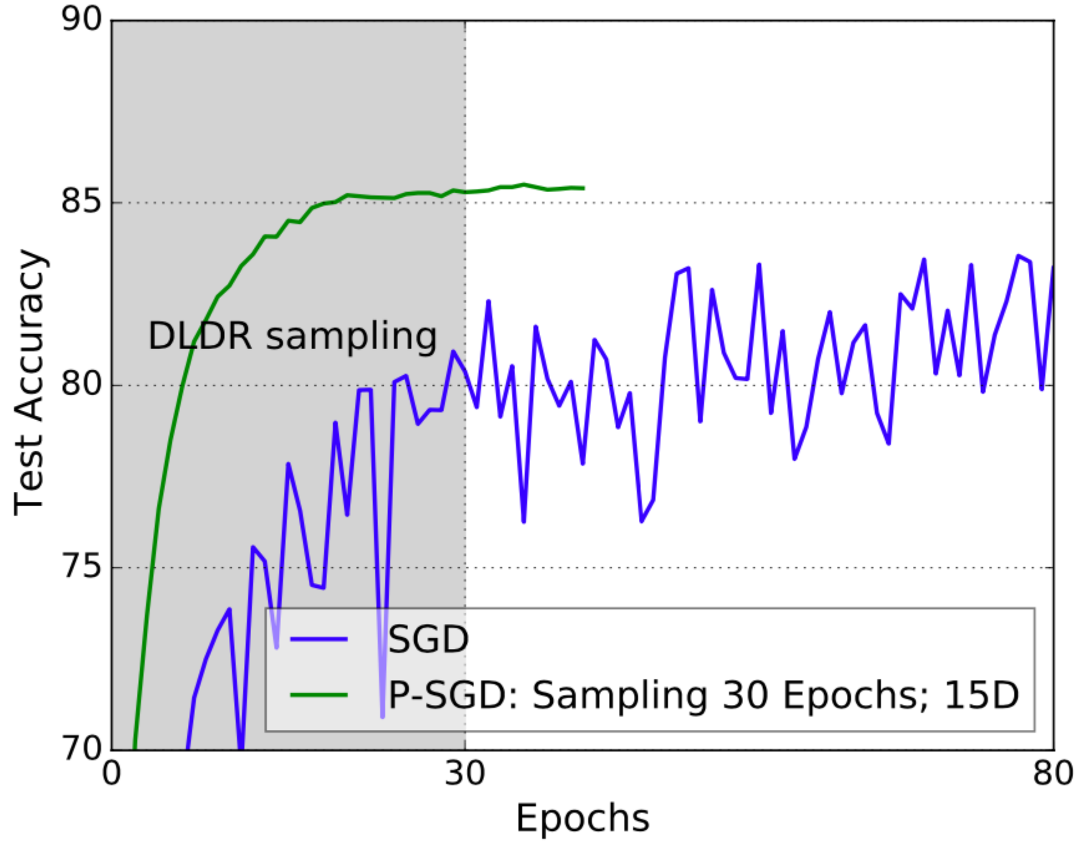
Test accuracy of ResNet8 on CIFAR-10

Dynamic Linear Dimensionality Reduction



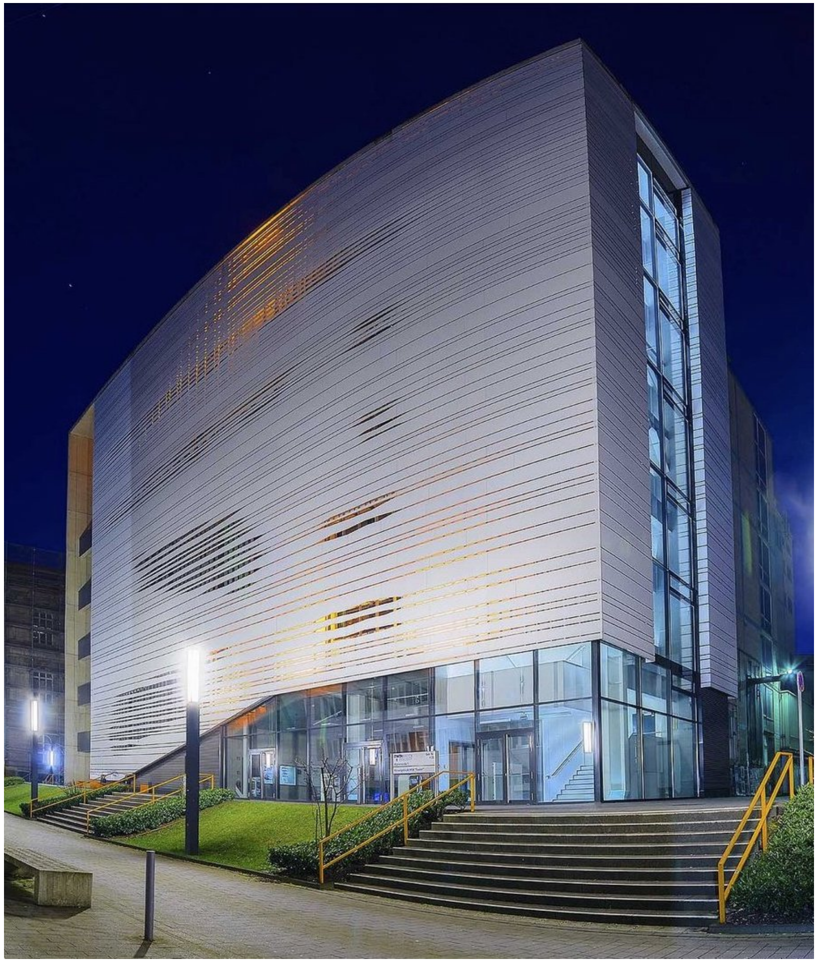
PCA ratio of the first 30 epochs training trajectory in the principal directions

Dynamic Linear Dimensionality Reduction



Training performance of SGD and P-SGD (in 15D subspace)

Agenda



- Dynamic Linear Dimensionality Reduction (DLDR)
- **DLDR-based Quasi-Newton Algorithm**
- Numerical Experiments

DLDR-based Quasi-Newton Algorithm

- Working in the low-dimensional space has the advantage that we can approximate the Hessian matrix as
$$H \approx PH_0P^T$$
- We use the BFGS algorithm with rank two correction update to approximate the inverse Hessian matrix as
$$\begin{aligned} B_{k+1} &= V_k^T B_k V_k + \rho_k \tilde{s}_k \tilde{s}_k^T \\ y_k &= \tilde{g}_{k+1} - \tilde{g}_k \\ \rho_k &= (y_k^T \tilde{s}_k)^{-1} \\ V_k &= I - \rho_k y_k \tilde{s}_k^T \end{aligned}$$

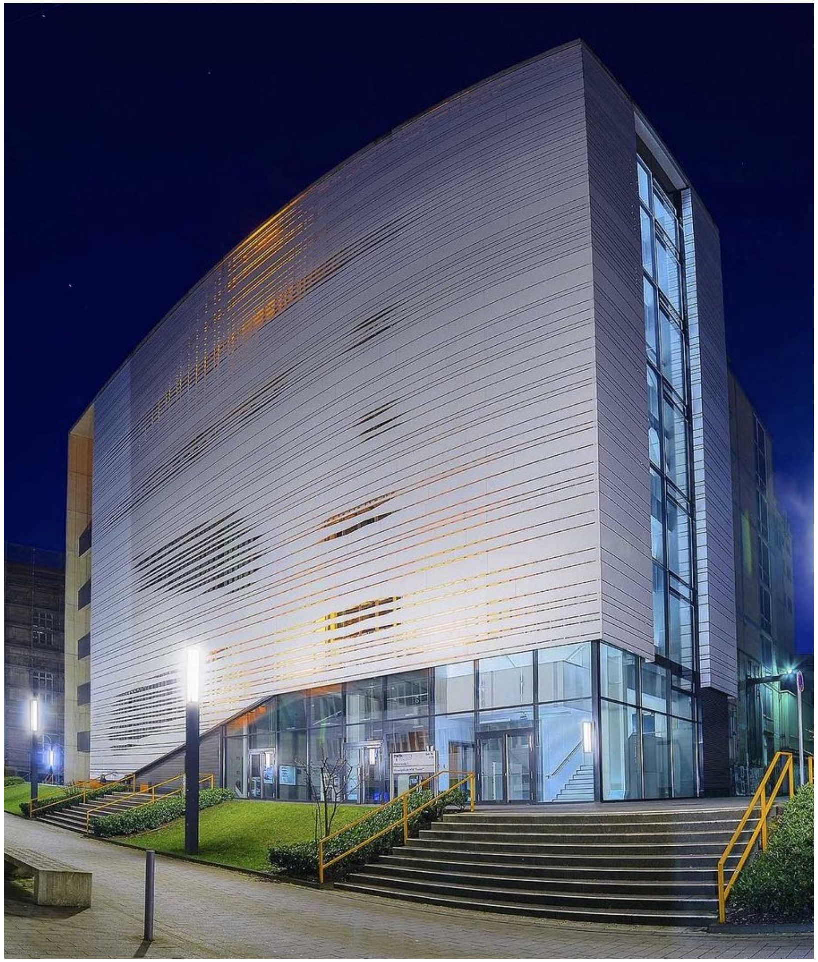
  where $\tilde{g}_k = P^T g_k$ is the projected gradient and $\tilde{s}_k = P^T(w_{k+1} - w_k)$ is the projected difference between the parameters

DLDR-based Quasi-Newton Algorithm

Algorithm 2 P-BFGS

Obtain $P = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d]$ by DLDR in Algorithm 1;
Initialize $k \leftarrow 0$ and $B_k \leftarrow I$;
while not converging **do**
    Sample the mini-batch data $\mathcal{B}_k$;
    Compute the gradient $\mathbf{g}_k$ on $\mathcal{B}_k$;
    Perform the projection $\tilde{\mathbf{g}}_k \leftarrow P^\top \mathbf{g}_k$;
    if $k > 0$ **then**
        Do Quasi-Newton update B_k with (13);
    Compute α_k using backtracking line search;
    $\tilde{\mathbf{s}}_k \leftarrow -\alpha_k B_k \tilde{\mathbf{g}}_k$;
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + P\tilde{\mathbf{s}}_k$; ▷ Update the parameters
    $k \leftarrow k + 1$;

Agenda



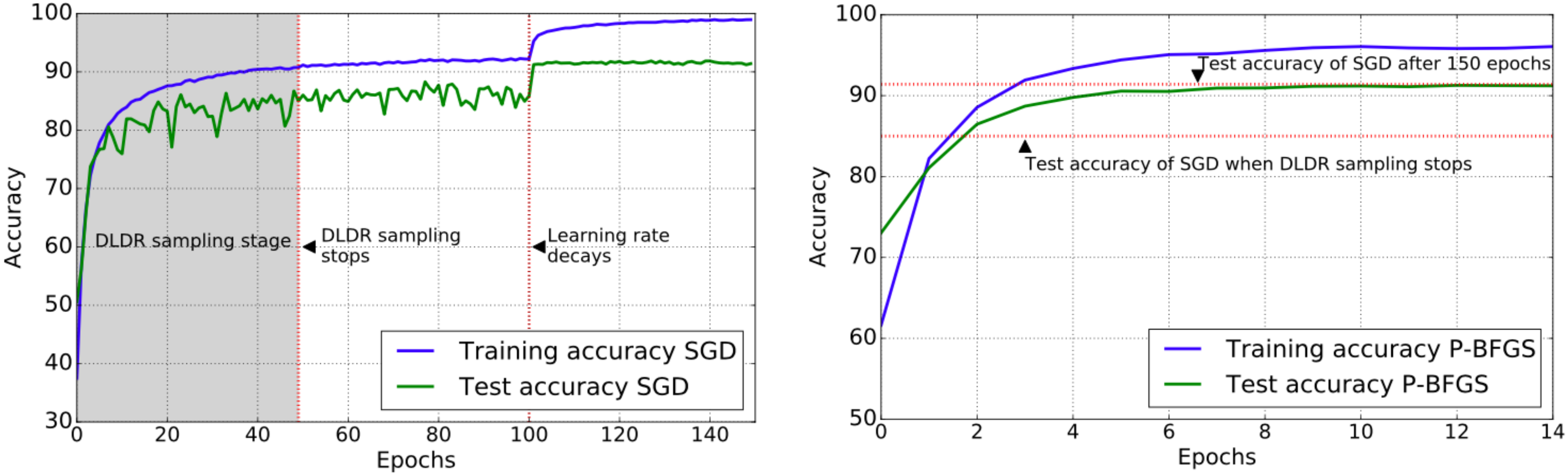
- Dynamic Linear Dimensionality Reduction (DLDR)
- DLDR-based Quasi-Newton Algorithm
- **Numerical Experiments**

Numerical Experiments

Models	# Parameter	SGD (#training epochs) 50	100	200	P-SGD (#sample epochs) 50	100
VGG11_bn [41]	28.5M	58.38	59.90	68.87	68.72	70.18
EfficientNet-B0 [47]	4.14M	62.53	63.68	72.94	71.68	72.64
MobileNet [20]	3.3M	57.15	58.67	67.94	66.86	68.00
DenseNet121 [22]	7.0M	65.39	64.57	76.76	74.25	76.34
Inceptionv3 [46]	22.3M	61.68	64.00	76.25	75.15	76.83
Xception [7]	21.0M	64.57	65.81	75.47	75.68	75.56
GoogLeNet [45]	6.2M	62.32	66.32	76.88	75.66	77.27
ShuffleNetv2 [35]	1.3M	62.90	63.15	72.06	71.34	72.29
SequeezeNet [23]	0.78M	59.52	58.56	70.29	69.89	70.60
SEResNet18 [21]	11.4M	64.74	64.68	74.95	74.33	75.09
NasNet [53]	5.2M	63.73	66.80	77.34	77.19	77.03

Test accuracy on CIFAR-100 for regular training and optimization in 40D subspaces

Numerical Experiments



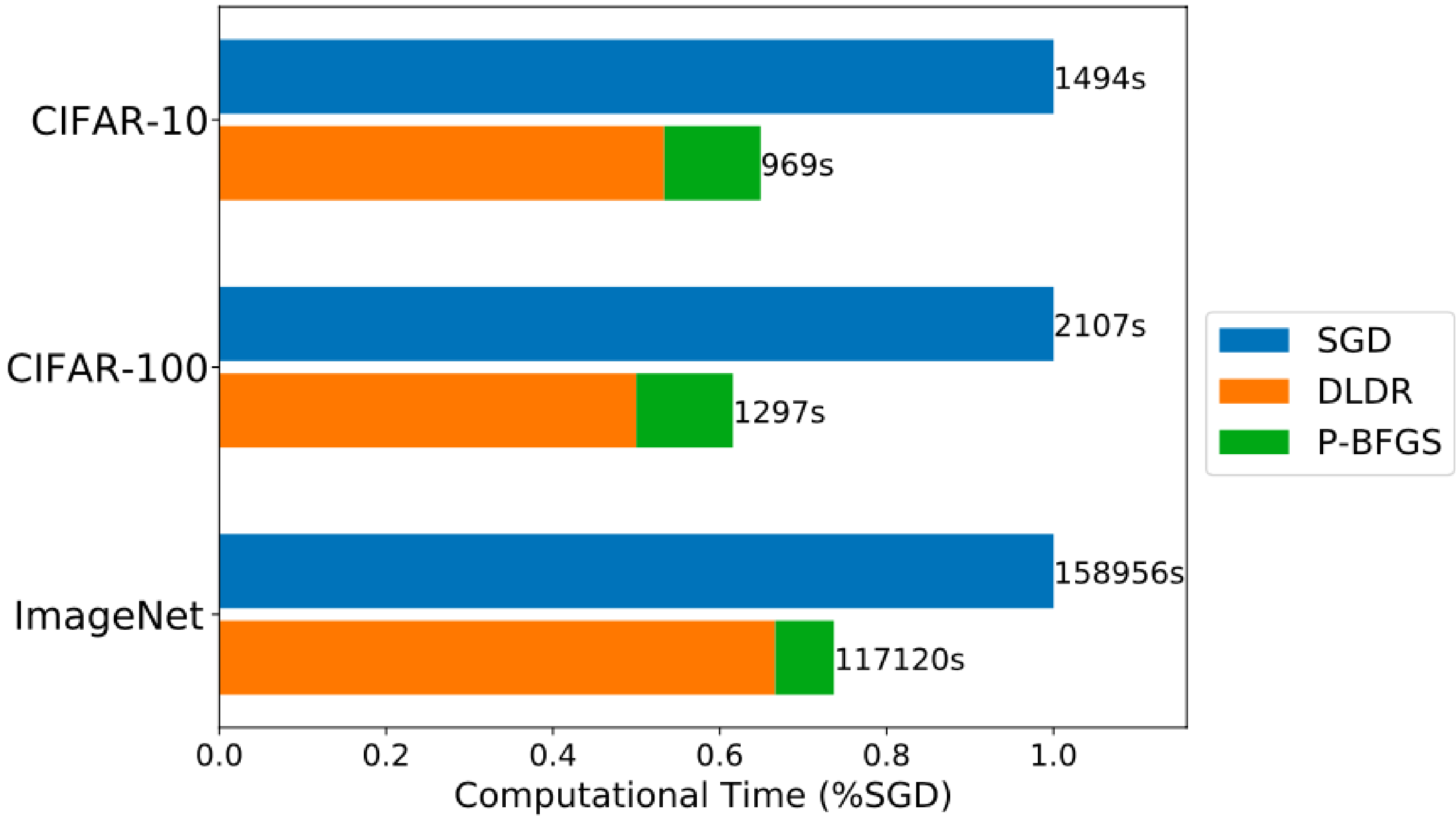
The accuracy of ResNet20 on CIFAR-10 in different epochs

Numerical Experiments

Dataset		CIFAR-10	CIFAR-100	ImageNet
Model		ResNet20	ResNet32	ResNet18
SGD	epochs	150	200	90
	acc	91.55 ± 0.23	68.40 ± 0.45	69.794
P-BFGS	sampling	80	100	60
	epochs	20	20	4
	acc	91.72 ± 0.10	69.90 ± 0.48	69.720

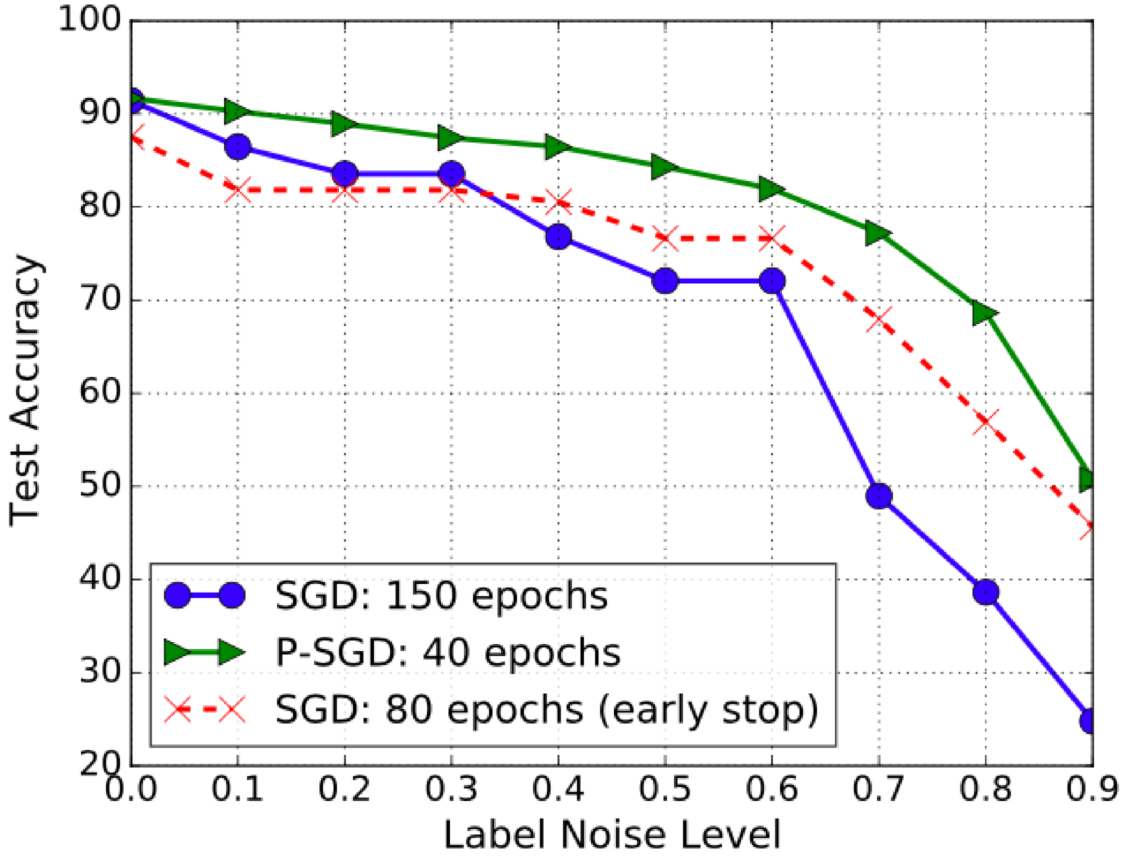
Test accuracy obtained by SGD and P-BFGS

Numerical Experiments



Wall-clock time consumption comparisons

Numerical Experiments



The performance under different level of label noise

Are there any questions?

Ideas for further exploration

- Test different second order methods
- Improve the subspace extraction (different sampling strategies, different dimensions, etc.)

Further ideas?