IEEE ICDM 2020

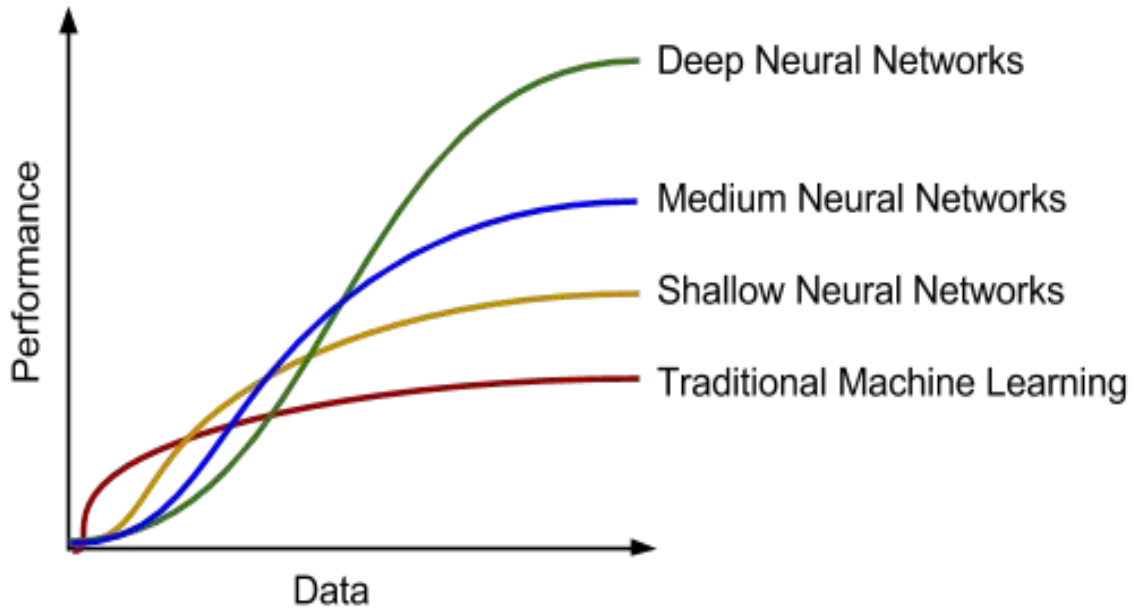
Toward Model Parallelism for Deep Neural Network based on Gradient-free ADMM Framework

Junxiang Wang, Zheng Chai, Yue Cheng and Liang Zhao

Presented by Junxiang Wang

Motivation: Why Distributed Deep Learning

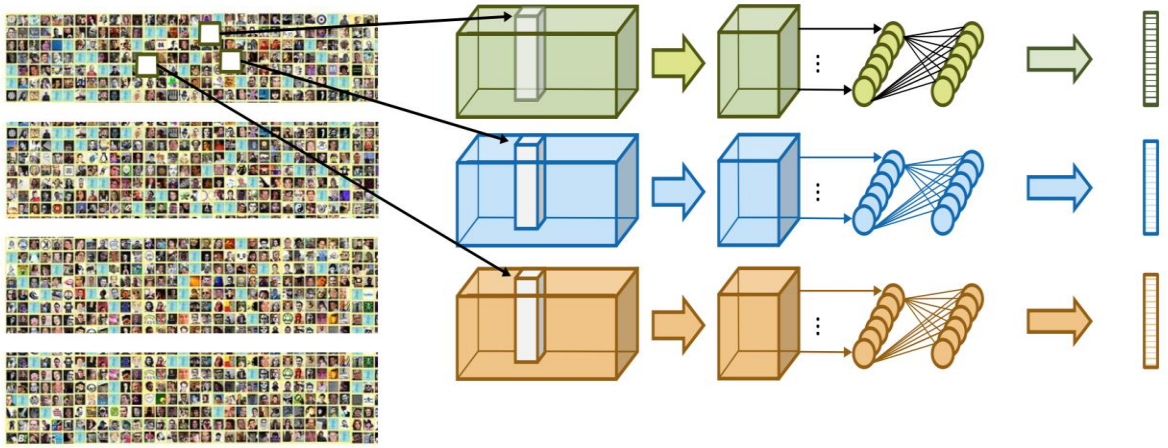
- Performance of neural networks is improved by a fast increase in the depth and size.
- It raises urgent demands for parallelize training of neural networks.



Model performance increases as neural network goes deeper.

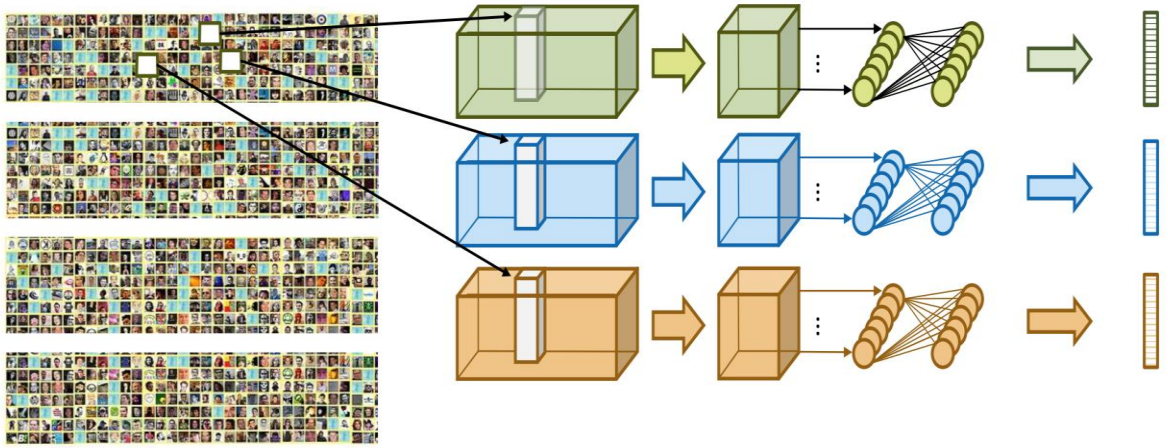
Parallelism: Data Parallelism VS Model Parallelism

- Data Parallelism
  - It focuses on distributing the data across different computing devices, which operate on the data in parallel.
  - Simple and efficient solution, easy to implement.
  - Duplicate parameters at all computing devices.

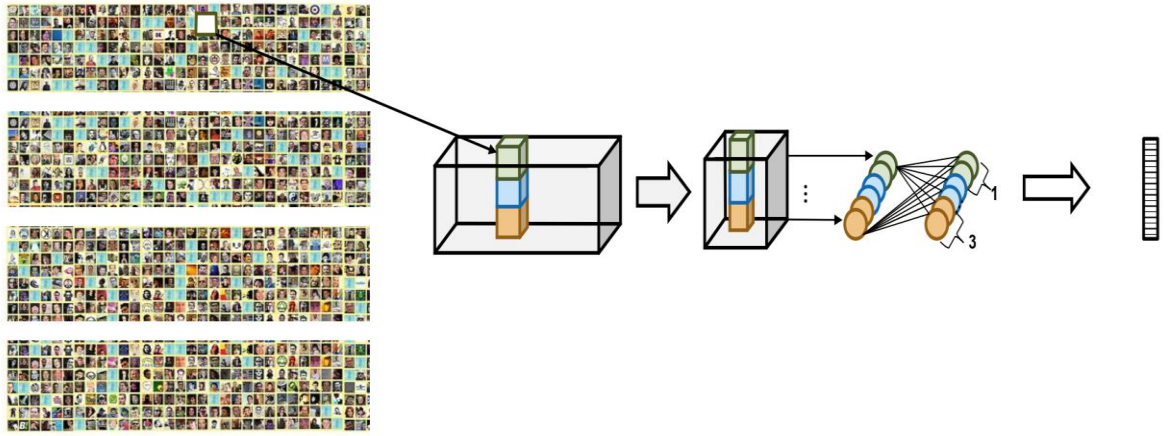


Parallelism: Data Parallelism VS Model Parallelism

- Data Parallelism
  - ➢ It focuses on distributing the data across different nodes, which operate on the data in parallel.
  - ➢ Simple and efficient solution, easy to implement.
  - ➢ Duplicate parameters at all processors.

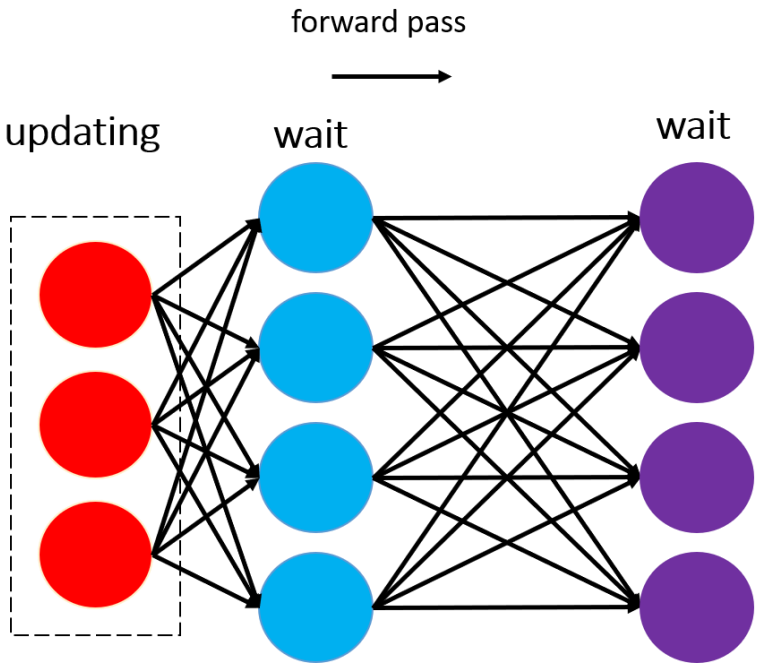


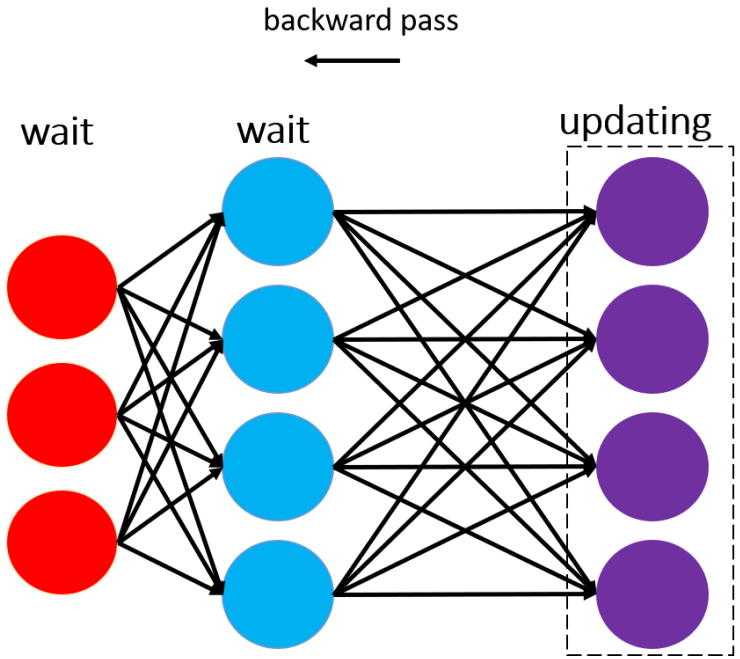
- Model Parallelism
  - ➢ It splits the model among computing devices and use the same data for each model.
  - ➢ Parameters can be distributed across processors.



Challenge of Data Parallelism

- Parallel training neural networks is mainly achieved by backprogration with data parallelism. However …
- The inherent bottleneck from backpropagation prevents the gradients of different layers being calculated in parallel.
- The gradient calculation of one layer depends on its previous layers.





Preliminary: ADMM as an Alternative

Recently, the Alternating Direction Method of Multipliers (ADMM) has been widely used in many deep learning applications.

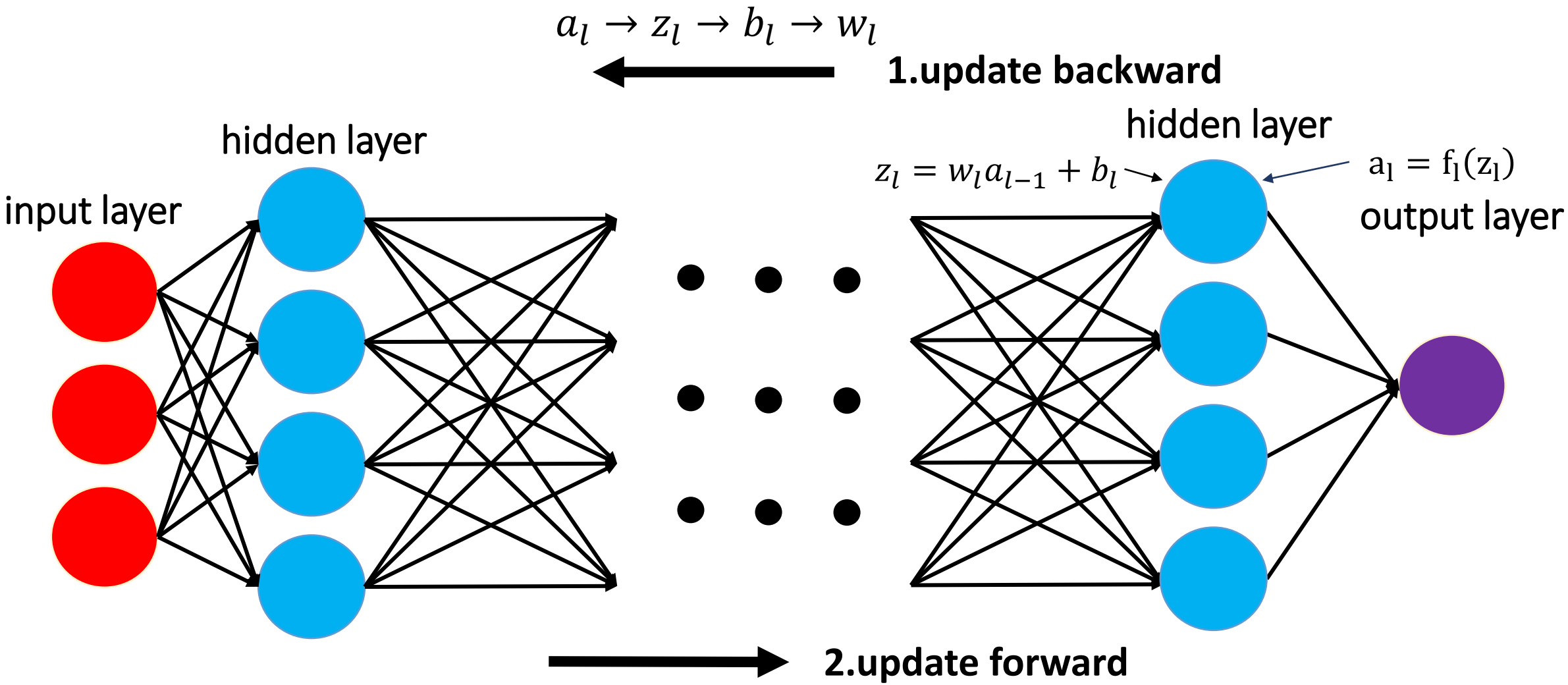
Original Problem: $\min_{\{x\,z\}} F(x) + G(z)$
$s.t.\, Ax + Bz = c$

Subroblem: $x^{k+1} \rightarrow \text{argmin}_x\, F(x) + (y^k)^T\left(Ax + Bz^k - c\right) + \frac{\rho}{2}\left\|Ax + Bz^k - c\right\|_2^2$

Subproblem: $z^{k+1} \rightarrow \text{argmin}_z\, G(z) + (y^k)^T\left(Ax^{k+1} + Bz - c\right) + \frac{\rho}{2}\left\|Ax^{k+1} + Bz - c\right\|_2^2$

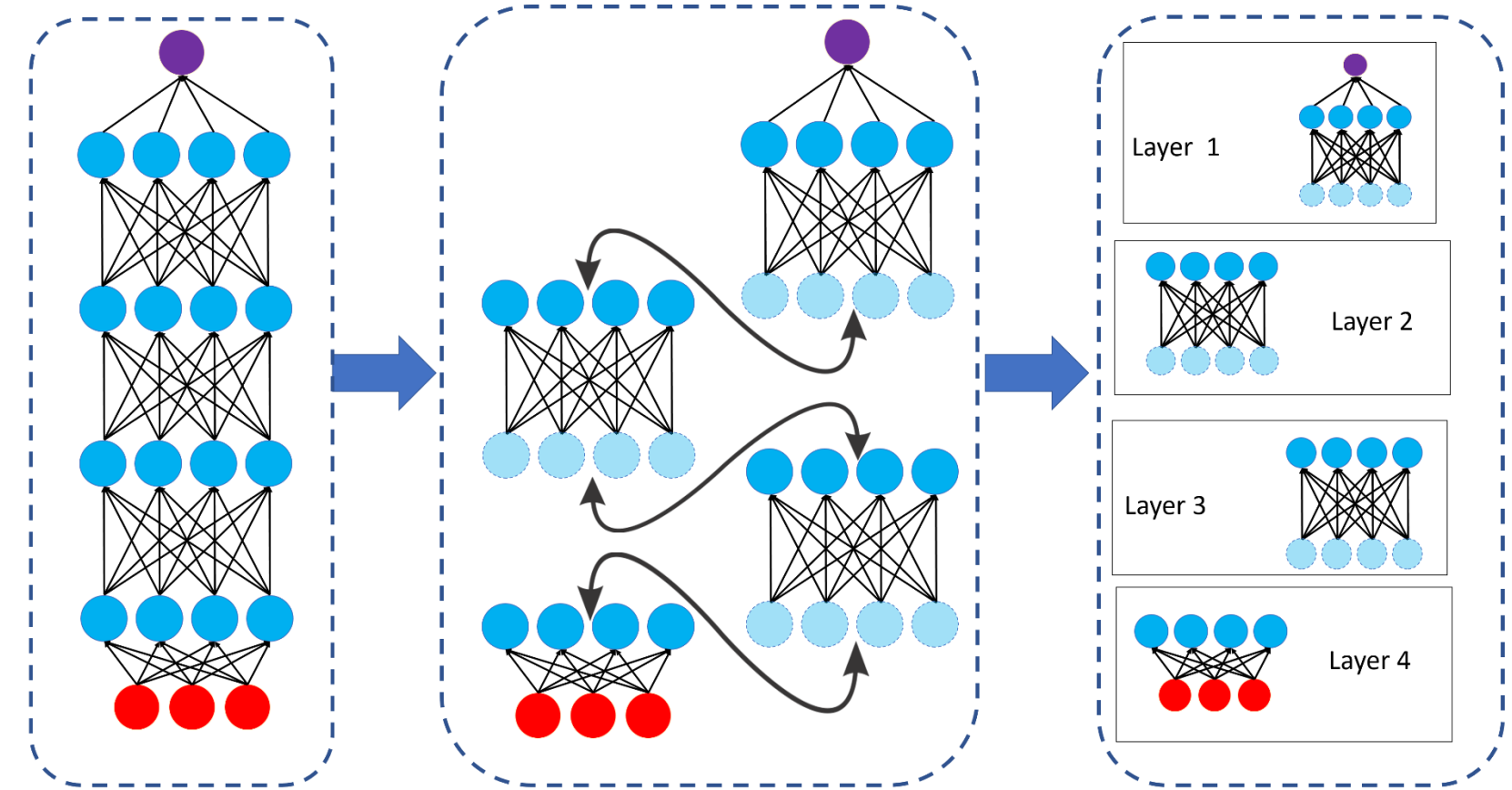
$y^{k+1} \rightarrow y^k + \rho(\text{A}x^{k+1} + Bz^{k+1} - c)$

Preliminary: dlADMM



Model Parallelism by ADMM

- Challenge of dlADMM: sequential update can not lead to parallel training.
- Idea: partition deep neural networks into layerwise components.



(a) Deep Neural Nets (DNNs) (b) DNN reformulation (c) Model Parallelism

Parallel Deep Learning Alternating Direction Method of Multipliers (pdADMM)

Problem 1.

Fully-connected neural network training problem

$$\min_{W_l, b_l, z_l, p_l} \overbrace{R(z_L; y)}^{\text{loss function}}$$

$$s.t. z_l = W_l \mathrm{p}_l + b_l (l = 1, \cdots, L)$$

$$p_{l+1} = \underbrace{f_l(z_l)}_{\text{activation function}} (l = 1, \cdots, L-1)$$

W_l: weight matrix for the l-th layer.
b_l: intercept vector for the l-th layer.
p_l: input for the l-th layer.
z_l: output for the linear mapping of the l-th layer.
L: number of layers.
y: predefined label vector.

Parallel Deep Learning Alternating Direction Method of Multipliers (pdADMM)

Problem 1.

Fully-connected neural network training problem

loss function

$$\min_{W_l,b_l,z_l,p_l} R(z_L;y)$$

$$s.t.\ z_l = W_l \mathrm{p}_l + b_l (l = 1,\cdots,L)$$

$$p_{l+1} = f_l(z_l)(l = 1,\cdots,L-1)$$

activation function

W_l: weight matrix for the l-th layer.
b_l: intercept vector for the l-th layer.
p_l: input for the l-th layer.
q_l: output for the l-th layer.
z_l: output for the linear mapping of the l-th layer.
L: number of layers.
y: predefined label vector.
ν: tuning parameter.

Problem 2.

loss function

$$\min_{W_l,b_l,z_l,p_l,q_l} F(\boldsymbol{p},\boldsymbol{W},\boldsymbol{b},\boldsymbol{z},\boldsymbol{q}) = R(z_L;y)$$

$$+\frac{\nu}{2}\sum_{l=1}^{L}(\|z_l - W_l p_l - b_l\|_2^2 + \|q_l - f_l(z_l)\|_2^2)$$

relaxation

splitting networks

$$s.t.\ p_{l+1} = q_l (l = 1,\cdots,L-1)$$

activation function

Parallel Deep Learning Alternating Direction Method of Multipliers (pdADMM)

Augmented Lagrangian Function:

$$L_\rho(\mathbf{p}, \mathbf{W}, \mathbf{b}, \mathbf{z}, \mathbf{q}, \mathbf{u}) = F(\mathbf{p}, \mathbf{W}, \mathbf{b}, \mathbf{z}, \mathbf{q}) + \sum_{l=1}^{L-1} (u_l^T(p_{l+1} - q_l) + (\rho/2)\|p_{l+1} - q_l\|_2^2)$$

Algorithm 1 the pdADMM Algorithm

Require: y, $p_1 = x$, ρ, ν.
Ensure: $\mathbf{p}, \mathbf{W}, \mathbf{b}, \mathbf{z}, \mathbf{q}$.
Initialize $k = 0$.
while $\mathbf{p}^k, \mathbf{W}^k, \mathbf{b}^k, \mathbf{z}^k, \mathbf{q}^k$ not converged **do**
Update p_l^{k+1} of different l in parallel.
Update W_l^{k+1} of different l in parallel.
Update b_l^{k+1} of different l in parallel.
Update z_l^{k+1} of different l in parallel.
Update q_l^{k+1} of different l in parallel.
$r_l^k \leftarrow p_{l+1}^{k+1} - q_l^{k+1} (l = 1, \cdots, L)$ in parallel # Compute residuals.
Update u_l^{k+1} of different l in parallel.
$k \leftarrow k + 1$.
end while
Output $\mathbf{p}, \mathbf{W}, \mathbf{b}, \mathbf{z}, \mathbf{q}$.

W_l: weight matrix for the l-th layer.
b_l: intercept vector for the l-th layer.
p_l: input for the l-th layer.
q_l: output for the l-th layer.
z_l: output for the linear mapping of the l-th layer.
L: number of layers.
y: predefined label vector.
ν: tuning parameter.
u_l: dual variable for the l-th layer.
ρ: penalty parameter.
r_l: residual for the l-th layer.

- Strategy to training deep neural network
  - Training a swallow neural network with the first few layers of the deep neural network.
  - More layers are added for training step by step until finally all layers are involved in the training process.

pdADMM Convergence

- Assumptions
  - $f_l(z_l)$ is Lipchitz continuous=> the augmented Lagrangian is decreasing.
  - $\partial f_l(z_l)$ is bounded=> the subrgadient of the augmented Lagrangian is bounded.
  - $F(\boldsymbol{p}, \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{z}, \boldsymbol{q})$ is coercive=> all variables are bounded.

- Convergence Properties (ρ is sufficiently large):
  - Objective Decrease (Lemma 1): $L_\rho(\boldsymbol{p^k}, \boldsymbol{W^k}, \boldsymbol{b^k}, \boldsymbol{z^k}, \boldsymbol{q^k}, \boldsymbol{u}^k)$ is decreasing.
  - Objective Bound (Lemma 2): $L_\rho(\boldsymbol{p^k}, \boldsymbol{W^k}, \boldsymbol{b^k}, \boldsymbol{z^k}, \boldsymbol{q^k}, \boldsymbol{u}^k)$ is bounded.
  - Subgradient Bound (Lemma 3): the subgradient $\partial L_\rho\left(\boldsymbol{p^k}, \boldsymbol{W^k}, \boldsymbol{b^k}, \boldsymbol{z^k}, \boldsymbol{q^k}, \boldsymbol{u}^k\right)$ is upper bounded by variables.

- Convergence Results:
  - pdADMM converges to a critical point of Problem 2(Theorem 2).
  - Its convergence rate is is $o(\frac{1}{k})$(Theorem 3).

Please refer to Appendix for proof details.

Experiment: Datasets and Comparison Methods

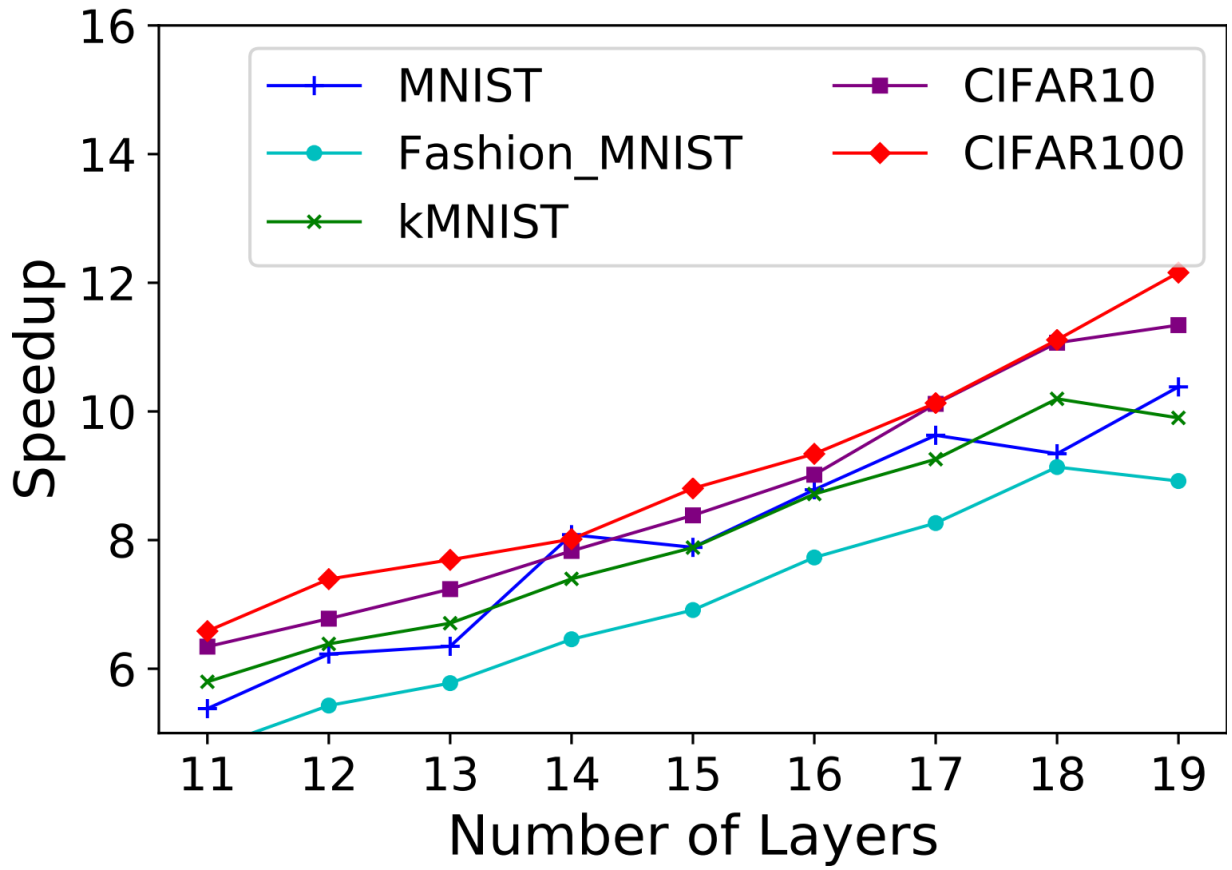
Dataset	Number of Features	Number of Training Samples	Number of Test Samples
MNIST	196	55000	10000
Fashion MNIST	196	60000	10000
kMNIST	196	60000	10000
SVHN	768	24446	9248
CIFAR10	768	12000	2000
CIFAR100	768	5000	1000

Comparison Methods: gradient descent, Adadelta, Adagrad, Adam and dlADMM.

Experiment: Speedup

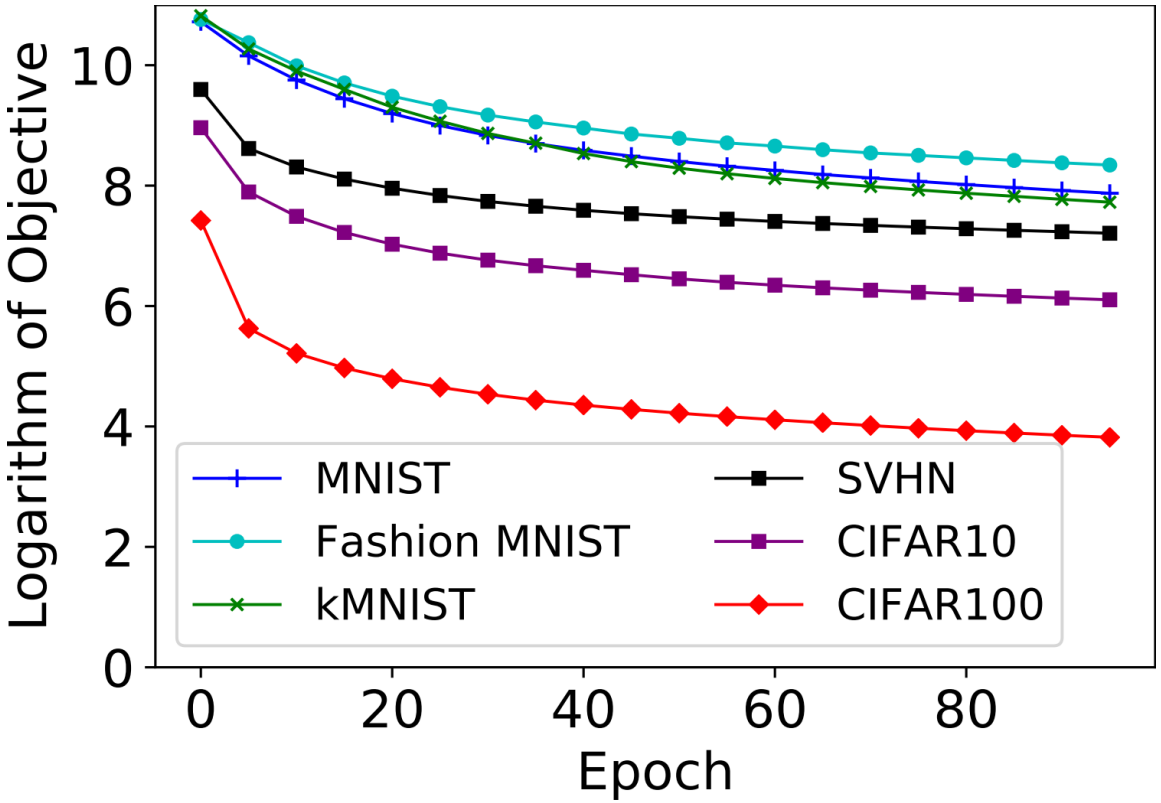
Number of Neurons	Serial pdADMM	pdADMM	Speedup
1500	237.66	26.78	8.87
1600	348.70	31.78	10.97
1700	390.51	35.79	10.91
1800	475.60	41.37	11.50
1900	465.57	45.87	10.15
2000	570.90	50.70	11.26
2100	570	54.91	10.38
2200	678.83	63.59	10.68
2300	710.3	70.36	10.10
2400	766.82	62.5	12.27

The speedup of pdADMM is more than 10 times on the MNIST dataset.

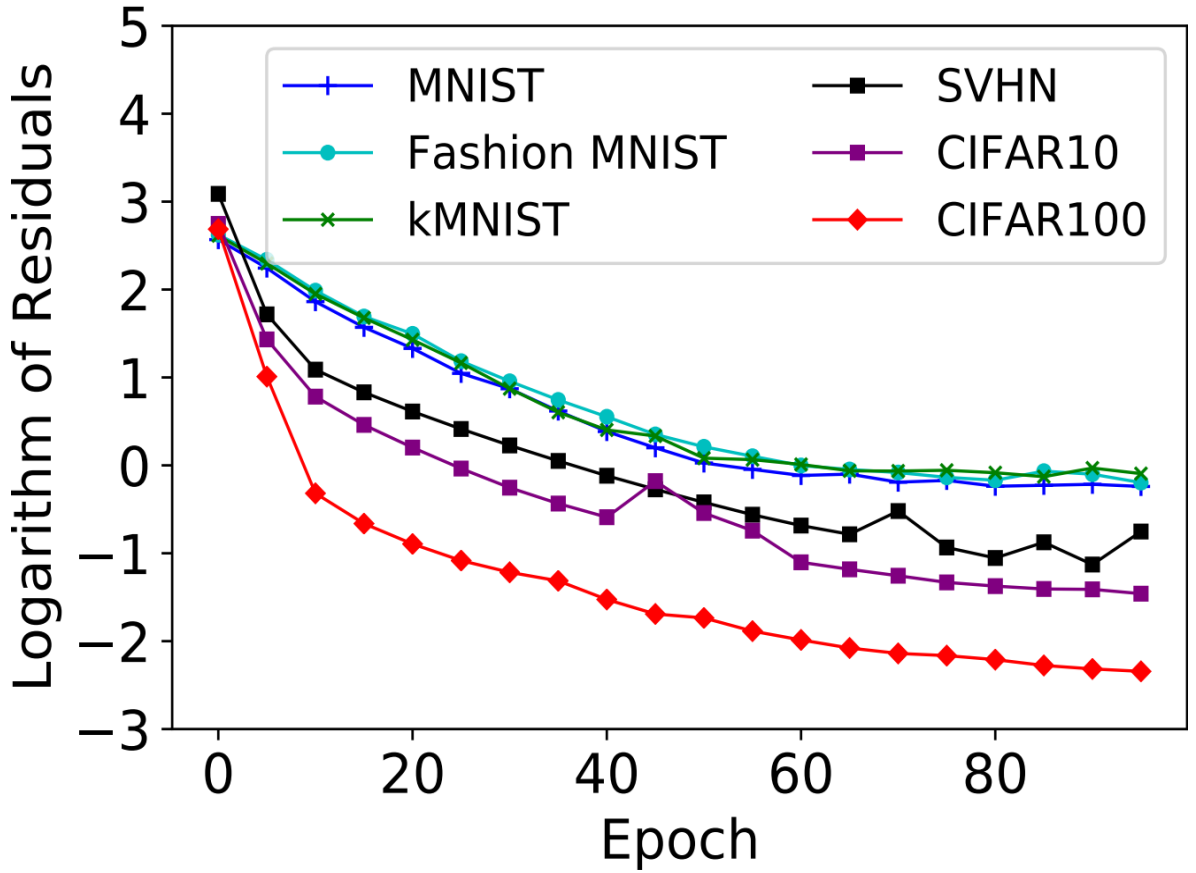


The speedup of pdADMM increases linearly with the number of layers.

Experiment: Convergence

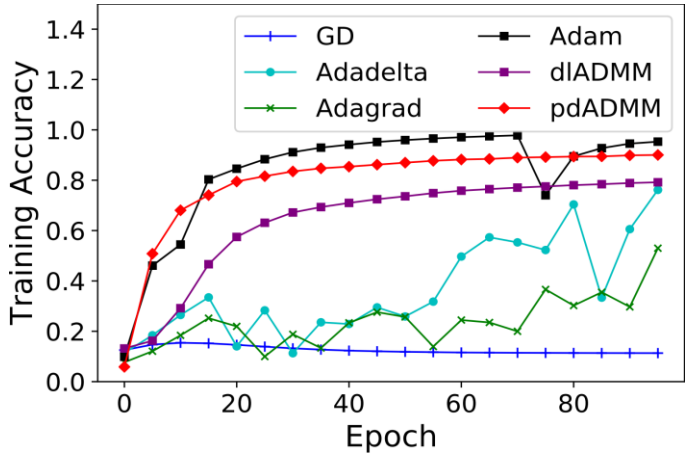


Objective is decreasing and convergent.

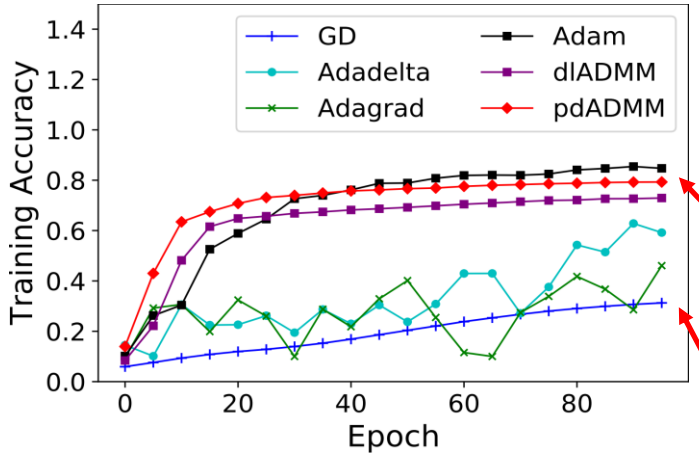


Residuals are convergent to 0.

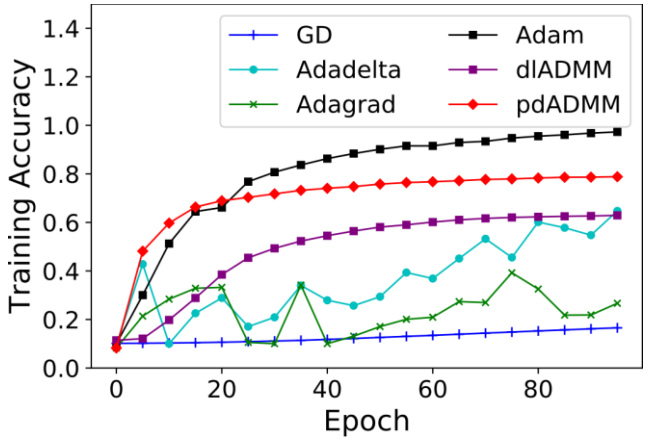
Experiment: Performance



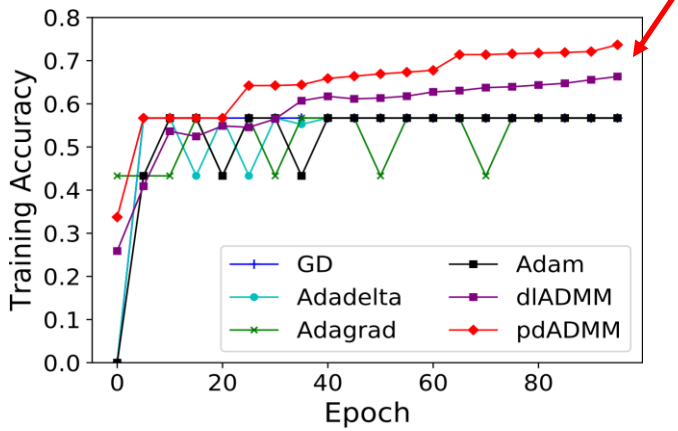
MNIST dataset



Fashion MNIST dataset



kMNIST dataset



SVHN dataset

pdADMM performs well among comparison methods.

Gradient descent suffers from gradient vanishing problems.

Summary

- A novel ADMM algorithm to train fully-connected neural network problem, in order to achieve model parallelism.
- Theoretical analysis of the proposed pdADMM to converge to a critical point of the problem with a sublinear convergence rate $o\left(\frac{1}{k}\right)$.
- Comprehensive experiments to demonstrate huge speedup, expected convergence and excellent performance of the proposed pdADMM.

We have released our code at: https://github.com/xianggebenben/pdADMM.
Please contact jwan936@emory.edu if you have any questions.

Thank you!