Implementation of model parallelism training

Kang Peilin

Semester project presentation

Machine Learning and Optimization Laboratory(MLO)

Professor: Martin Jaggi

June 21, 2019

Goal of semester project

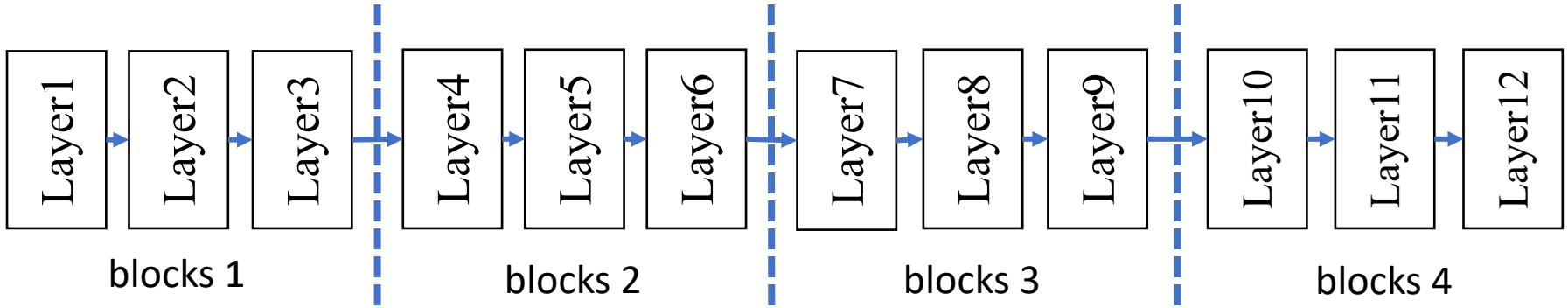
- Understand 4 different model parallelism training methods
  - Naive Sequential method(used as baseline)
  - Delayed gradient(DDG),
  - Features Reply(FR)
  - PipeDream

- Implement an unified framework(use PyTorch and mpi) to run these 4 methods

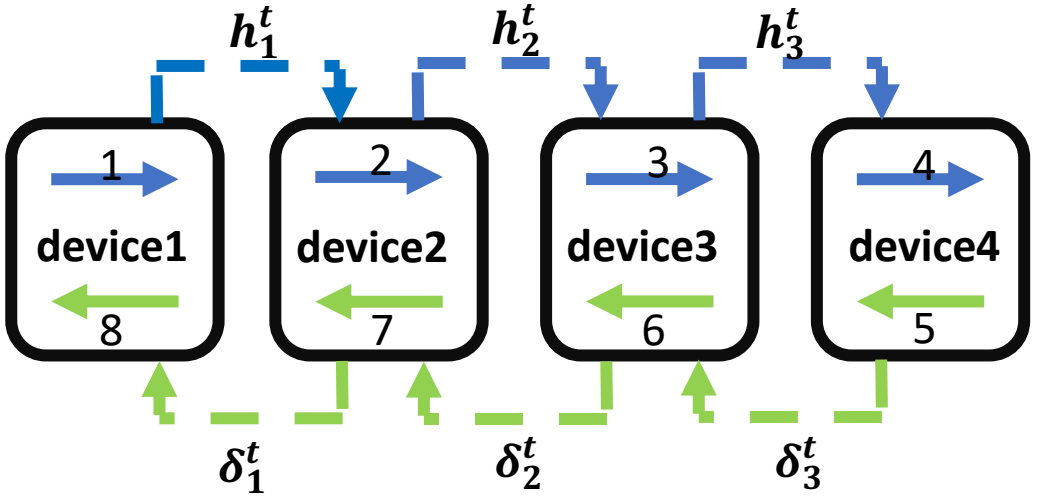
- Evaluate these methods on
  - ResNet used for image classification
  - simplified BERT used for language modeling

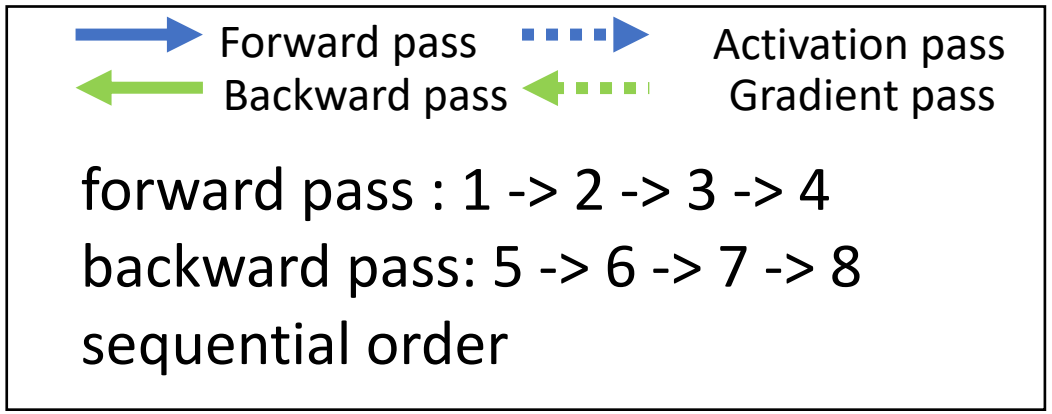
Model parallelism

Partition Partition the layers of the model into K blocks

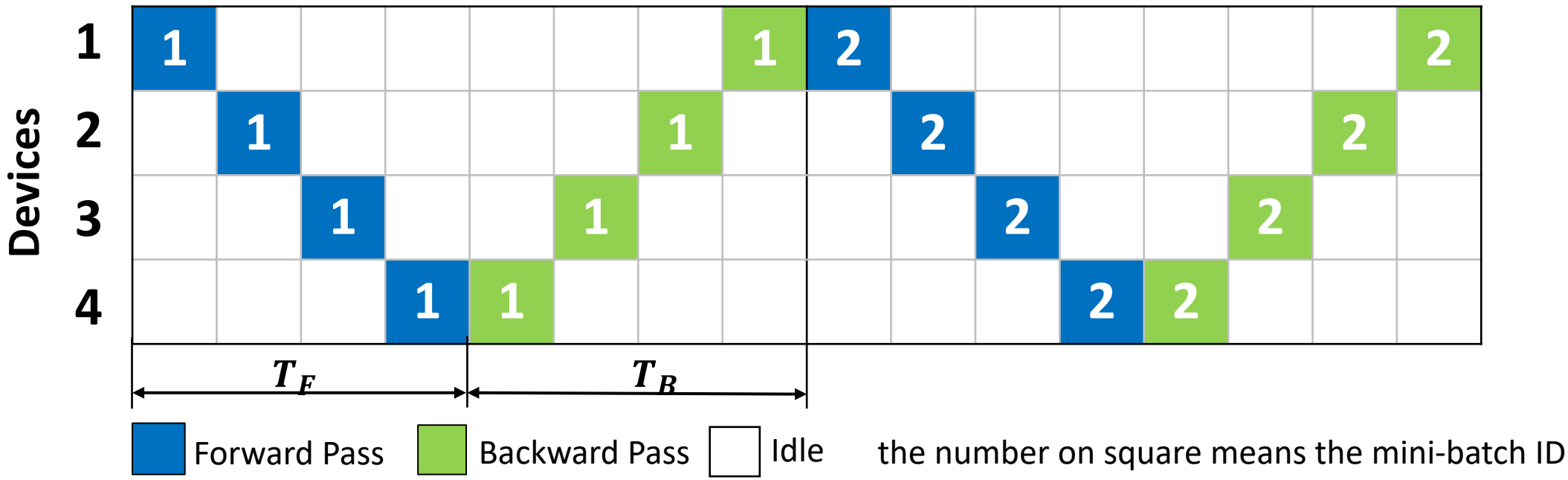


Backpropagation algorithm:





Naive Sequential method



Limitation:

the computation ability of devices are highly under utilized

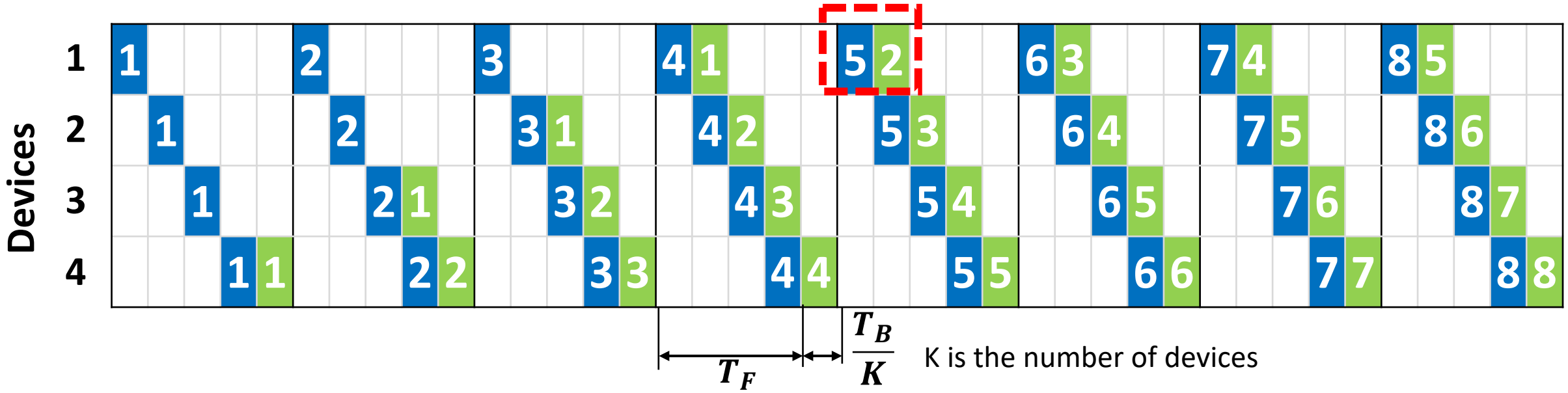
In forward pass: idle before receiving the output(activation) of the former device

In backward pass: idle before receiving error gradients from next device

Computation Time:

(T_F+T_B)**/one forward and backward pass**

Delayed Gradient (DDG)



improvements:

❖ **parallel backward pass, $(T_F + \frac{T_B}{K})$/one forward and backward pass**

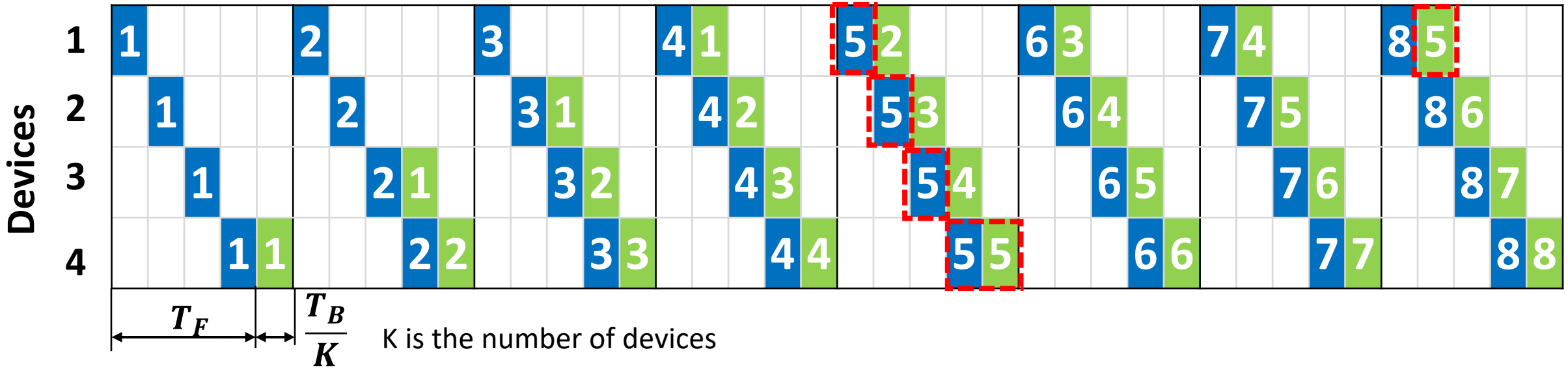
For example: Device 1

- ➢ Before forward pass of mini-batch 5
  has received error gradient of mini-batch 2 from device 2
- ➢ Do forward pass of mini-batch 5
- ➢ Do backward pass of mini-batch 2 immediately

(compared to mini-batch 5 we call the error gradient of mini-batch 2 to be delayed gradient)

Huo, Zhouyuan, et al. "Decoupled parallel backpropagation with convergence guarantee." *arXiv preprint arXiv:1804.10574*(2018).

Delayed Gradient (DDG)



Limitations:

❖ **High memory consumption**

For device k, store $K - k + 1$ intermediate states (inputs, activation and weights)

For example: Device 1 mini-batch 5
backward pass is 3 steps later after forward pass
Need to store the state of mini-batch 5 until then

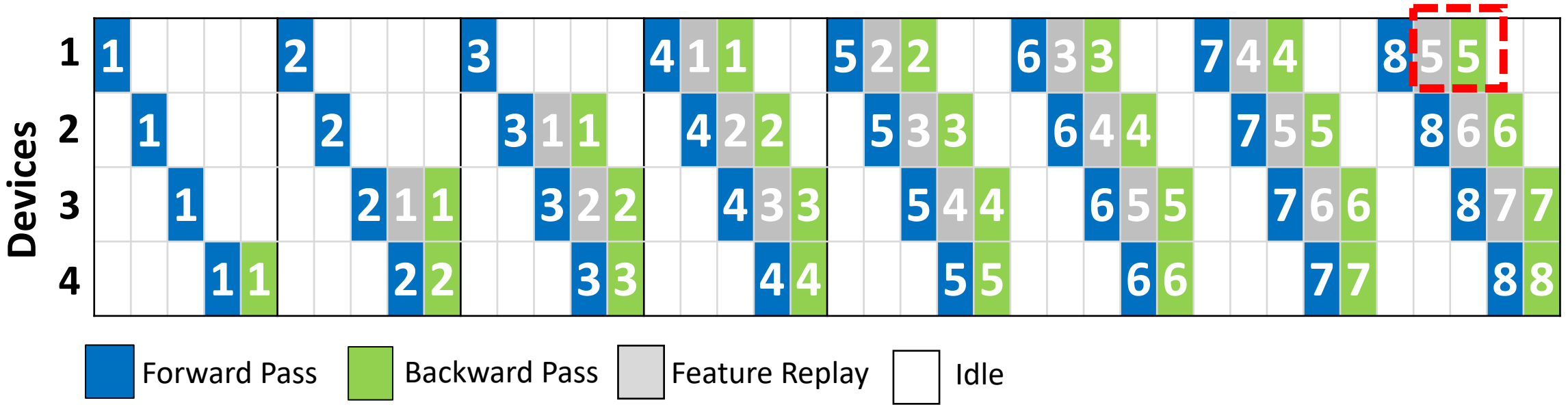
❖ **weight staleness**

Same mini-batch uses different weight versions in different devices

For example: mini-batch 5
device 1 uses weights updated from mini-batch 1
device 2 uses weights updated from mini-batch 2
device 3 uses weights updated from mini-batch 3
device 4 uses weights updated from mini-batch 4

Huo, Zhouyuan, et al. "Decoupled parallel backpropagation with convergence guarantee." *arXiv preprint arXiv:1804.10574*(2018).

Features Replay(FR)



improvements:

❖ **parallel backward pass**

For example: backward pass of Device 1 mini-batch 5

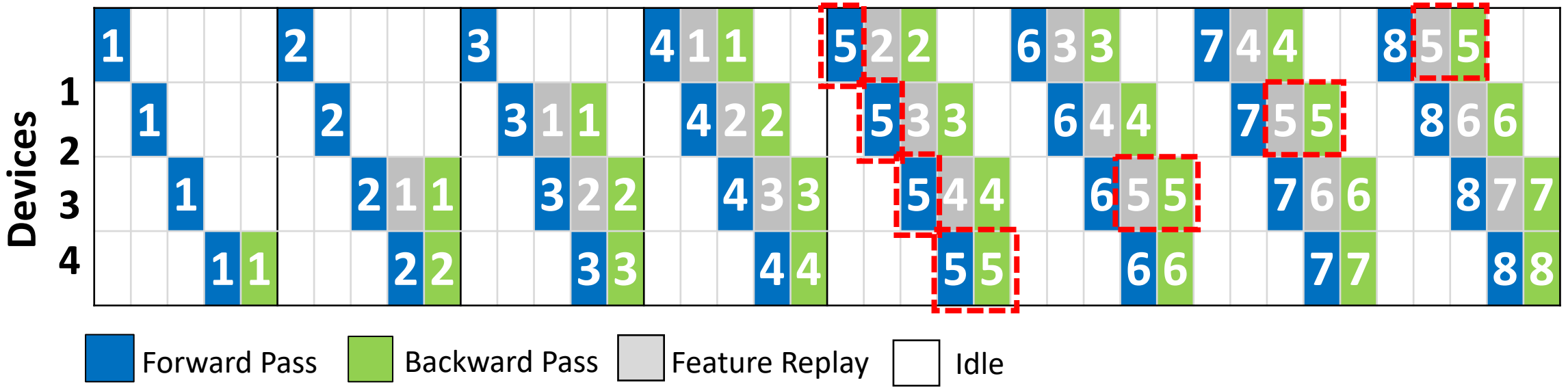
latest weight updated from mini-batch 4
stored inputs of mini-batch 5
→ recompute the activation

❖ **lower memory consumption**

For device k, store $K - k + 1$ inputs
no need to store the activations and weight

Huo, Zhouyuan, Bin Gu, and Heng Huang. "Training Neural Networks Using Features Replay." *Advances in Neural Information Processing Systems*. 2018.

Features Replay(FR)



Limitation:

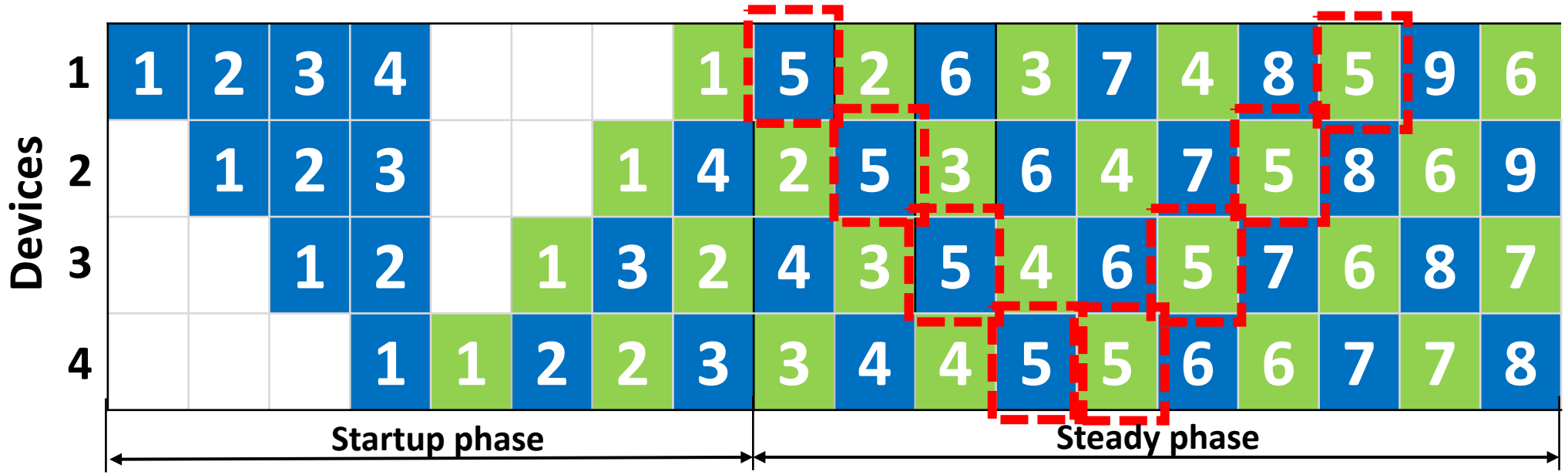
❖ **Forward and backward inconsistency**

For example: Device 1 mini-batch 5

- Forward pass: uses weights updated from mini-batch 1
- Backward pass: uses weights updated from mini-batch 4 (latest weight)
- Compared to DDG, in forward pass, mini-batch 5 also uses different weight versions in different devices but in backward ward pass, mini-batch 5 uses latest weight for all devices.

Huo, Zhouyuan, Bin Gu, and Heng Huang. "Training Neural Networks Using Features Replay." *Advances in Neural Information Processing Systems*. 2018.

PipeDream

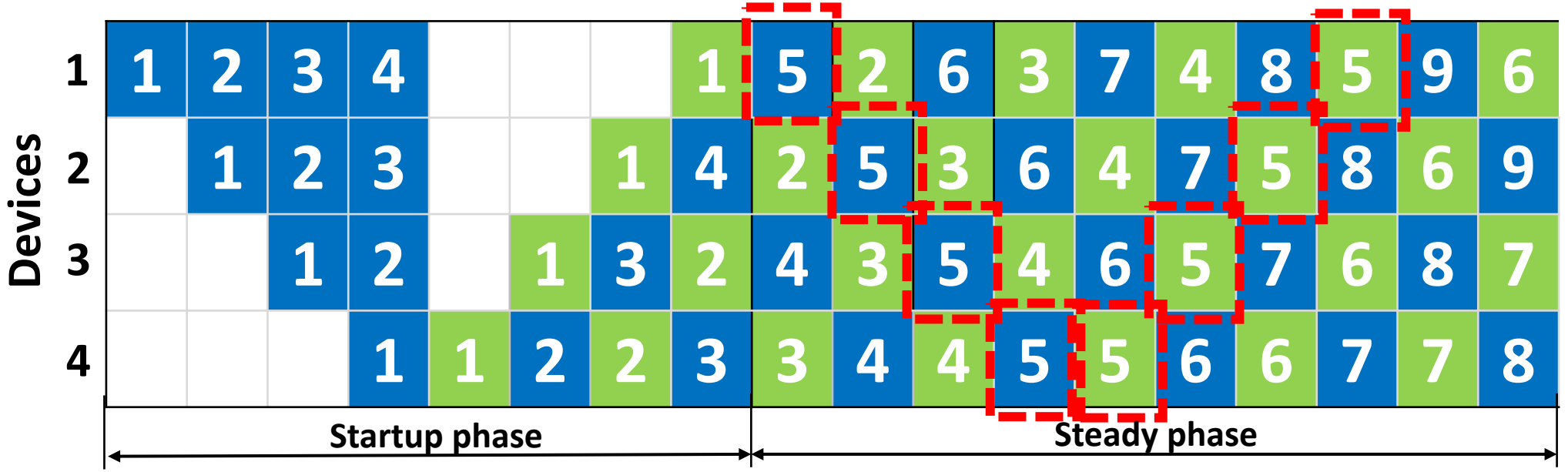


improvements:

❖ **Parallel the forward and backward pass** $\frac{T_F+T_B}{K}$**/per forward and backward pass**

- startup phase
  the first device loads K(K is the number of devices) mini-batches
- steady state
  alternates between performing the forward and backward pass for a mini-batch.
- Same as DDG, use delayed gradients to parallel the backward pass

PipeDream



limitations:

- **High memory consumption**

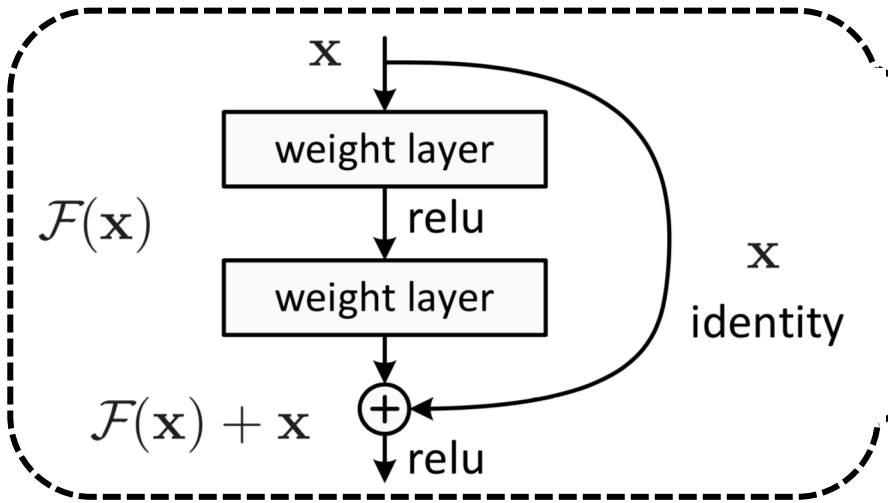
  For device k, store $K - k + 1$ intermediate states (inputs, activation and weights)

- **Weight staleness**

  Same mini-batch different weight version

Method	Computation Time	Memory	Weight staleness	Forward and backward inconsistency
Naive sequential	$T_F + T_B$	1 mini-batch states	No	No
Delayed Gradient(DDG)	$T_F + \frac{T_B}{K}$	$K - k + 1$ mini-batches states	Yes	No
Features Replay(FR)	$T_F + \frac{T_B}{K}$	Only store $K - k + 1$ mini-batches inputs	No	Yes
PipeDream	$\frac{T_F + T_B}{K}$	$K - k + 1$ mini-batches states	Yes	No

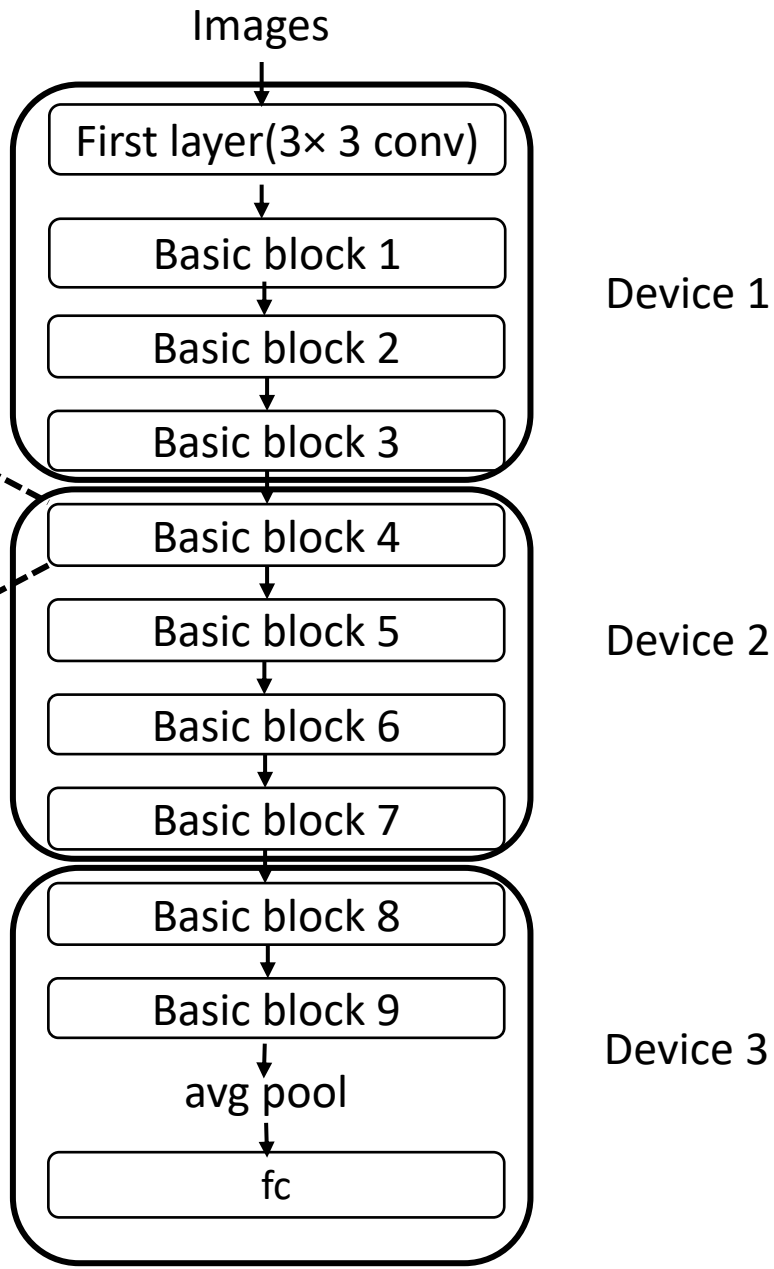
Implementation: Model partition



Basic block of Resnet

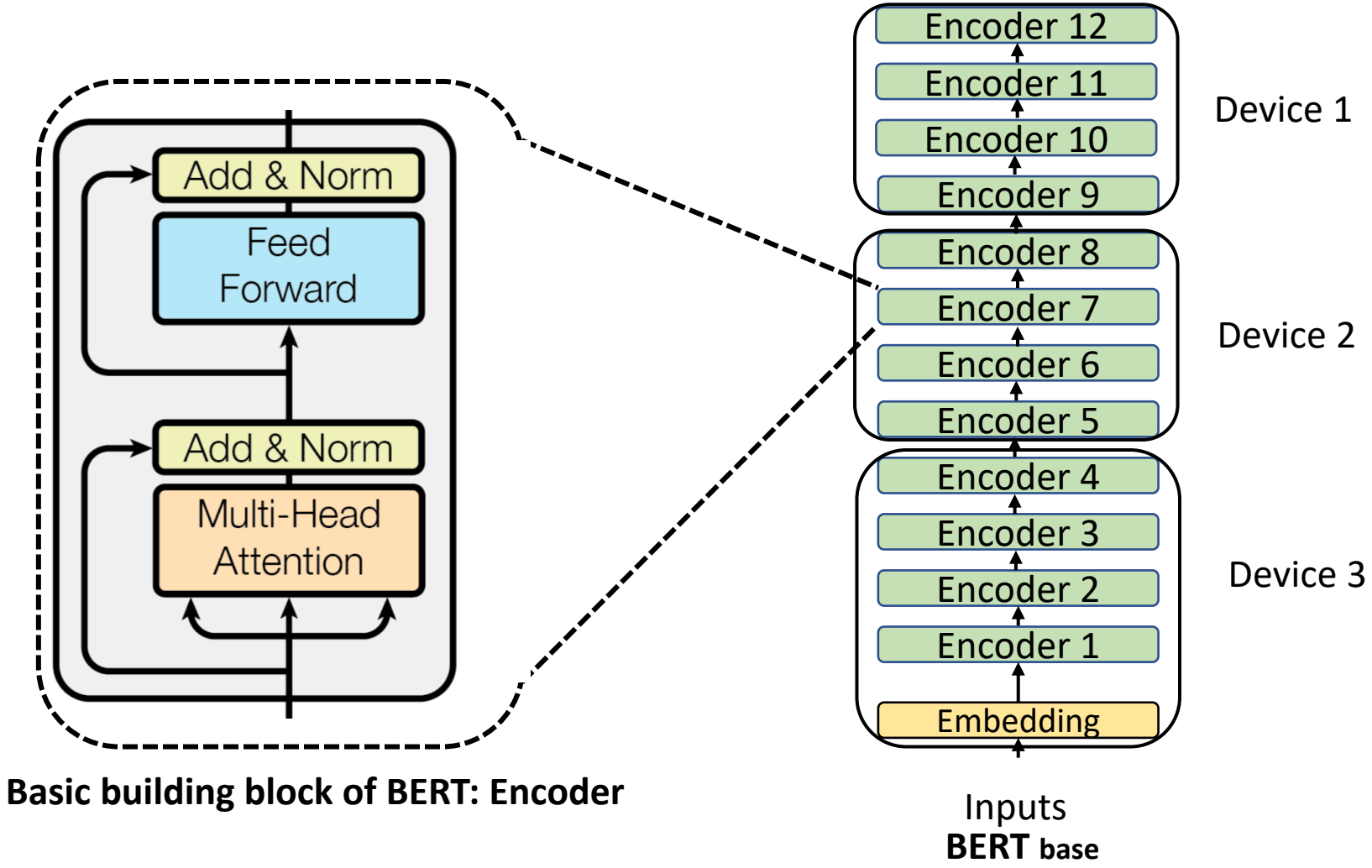
Take Two sub-layers with a residual connection as Basic building block of ResNet.

We will not partition between these 2 sub-layers



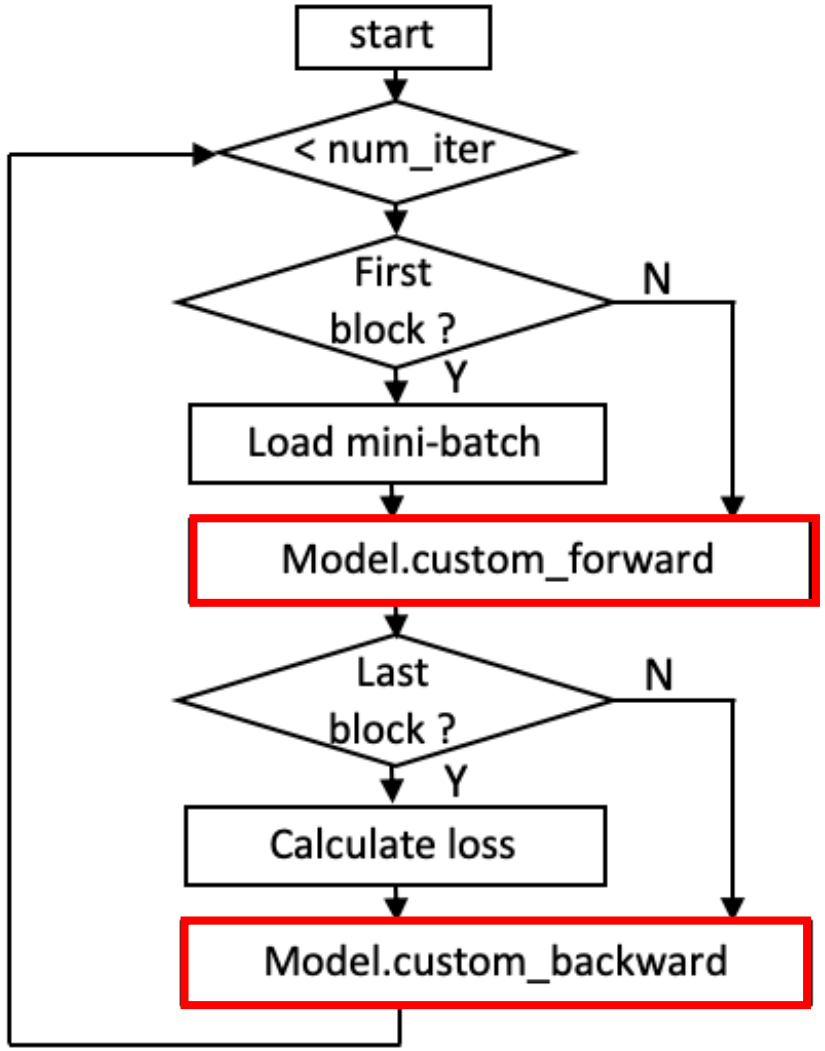
Resnet 20

Implementation: Model partition



Basic building block of BERT: Encoder

Implementation: Training schedule



General training schedule
Same for different models and methods

Partial Model class(nn.Module)

Functions:

- Forward
- Backward

Inherited from model

- Custom_forward:
  rev/send data
  invoke Forward
  Store input/output
- Custom_backward:
  invoke Backward
  rev/send gradient

Defined by ourselves
Different for each parallelism methods

Evaluation: ResNet20

- ResNet20 on image classification benchmark datasets: CIFAR-10
- batch size: 128 epochs:300
- The number of blocks [2, 3, 4, 6, 11]
- Learning rate [0.01, 0.04, 0.08, 0.16, 0.20, 0.25, 0.32]
- each run 3 times

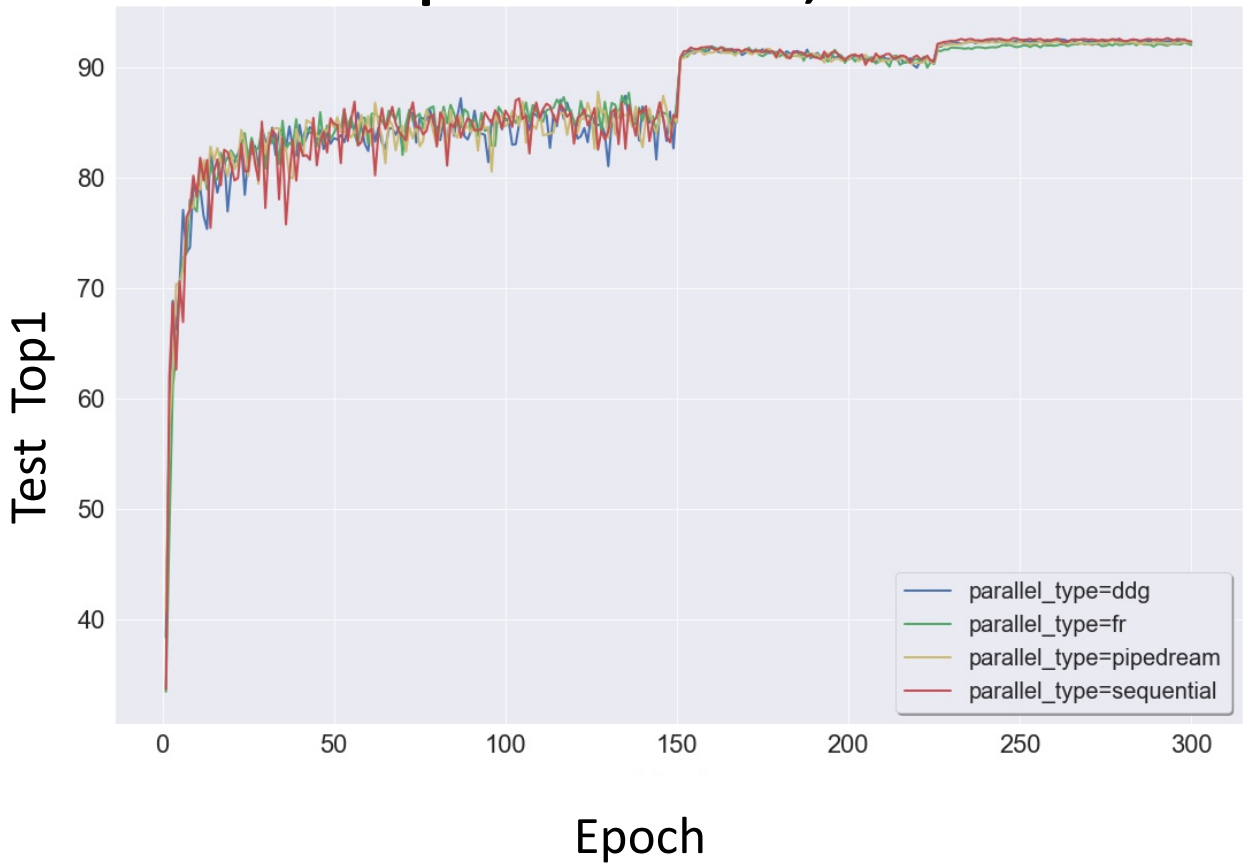
The best top1 accuracy of different methods of different number of blocks.

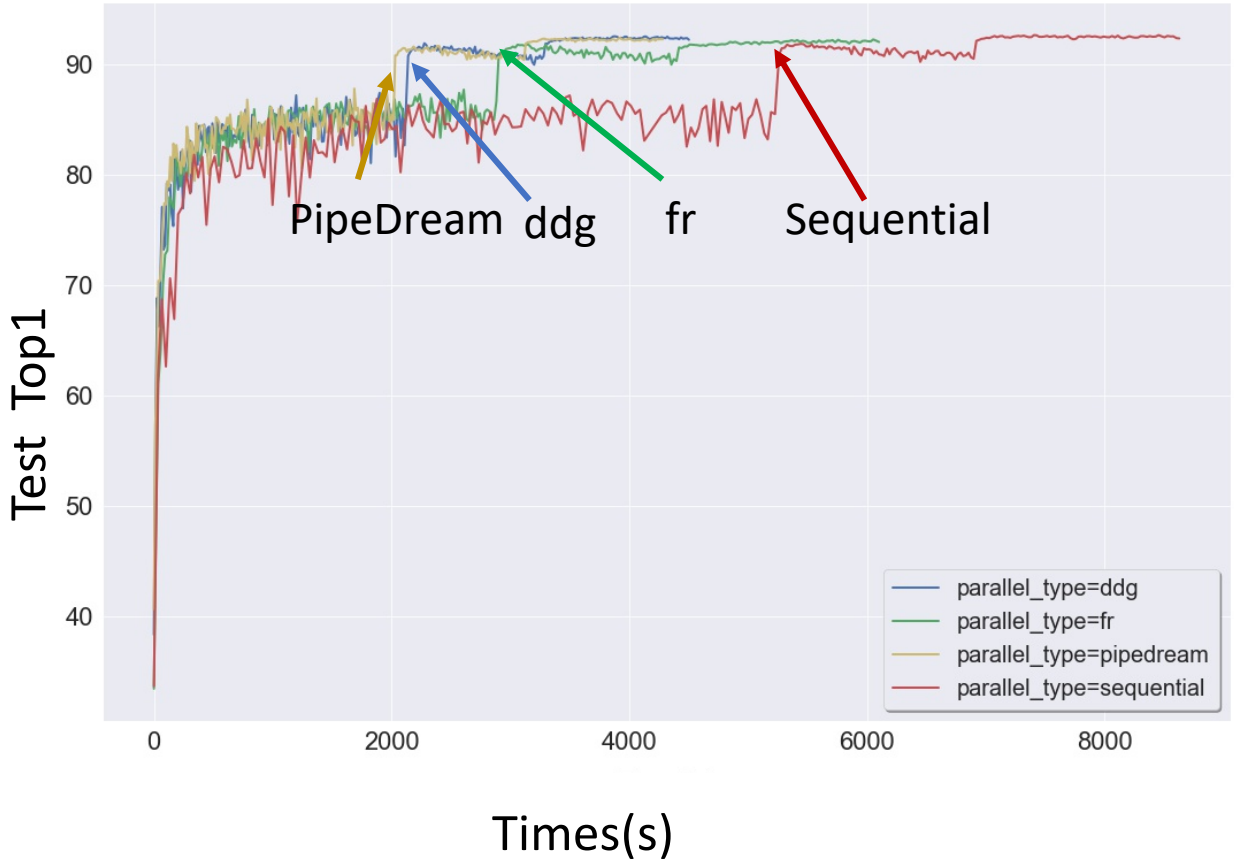
Num of blocks	Sequential		DDG		FR		PipeDream	
	top1	lr	top1	lr	top1	lr	top1	lr
2 blocks	92.622 ± 0.176	0.25	92.695 ± 0.113	0.32	92.555 ± 0.181	0.25	92.541 ± 0.012	0.20
3 blocks	92.471 ± 0.187	0.32	92.515 ± 0.031	0.20	**92.364** ± 0.029	0.20	92.458 ± 0.041	0.32
4 blocks	92.551 ± 0.144	0.25	**92.258** ± 0.128	0.25	**92.388** ± 0.037	0.32	92.521 ± 0.233	0.32
6 blocks	92.762 ± 0.151	0.32	**92.258** ± 0.121	0.32	**92.157** ± 0.087	0.25	**92.288** ± 0.225	0.25
11 blocks	92.598 ± 0.164	0.32	**90.728** ± 1.511	0.16	**92.241** ± 0.144	0.32	**91.727** ± 0.393	0.25

*The bold one indicates that the top1 accuracy is lower than others.

Evaluation: ResNet20

ResNet20 split to 3 blocks, lr is 0.32

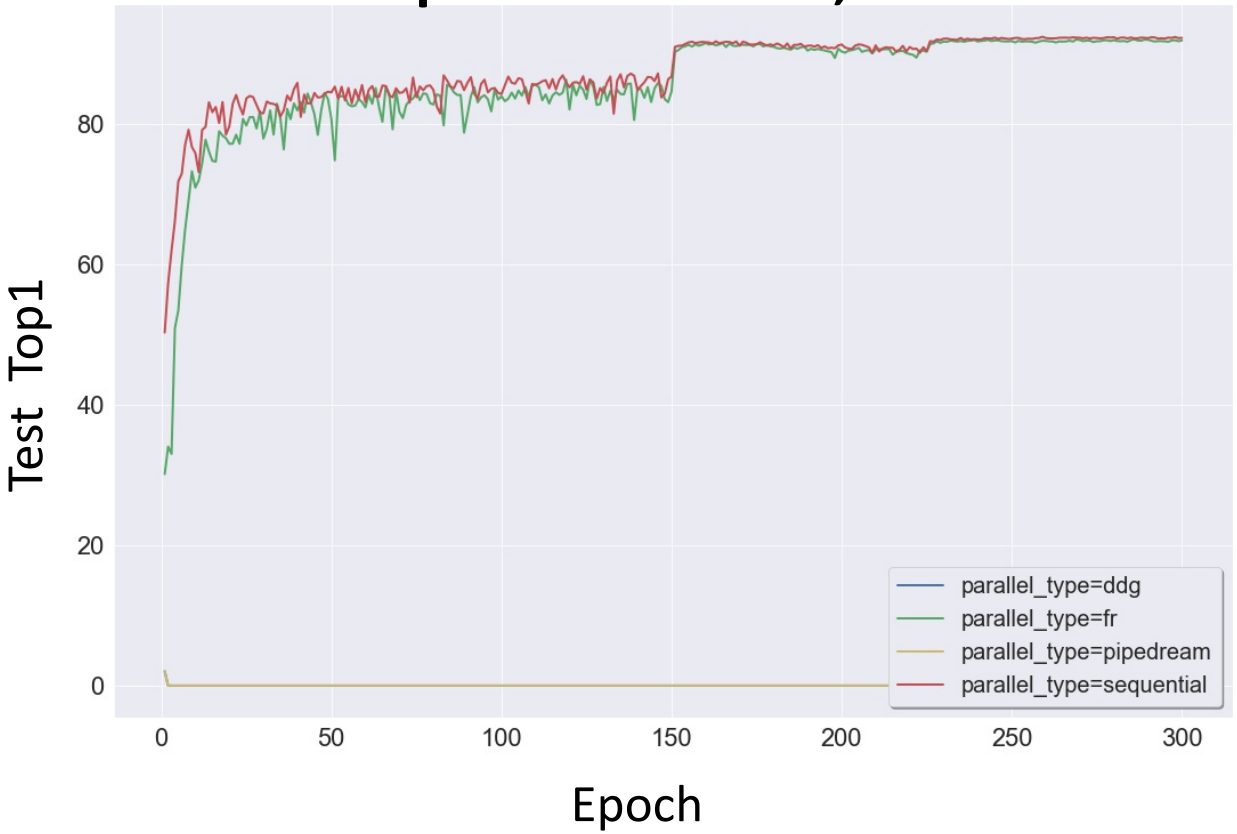


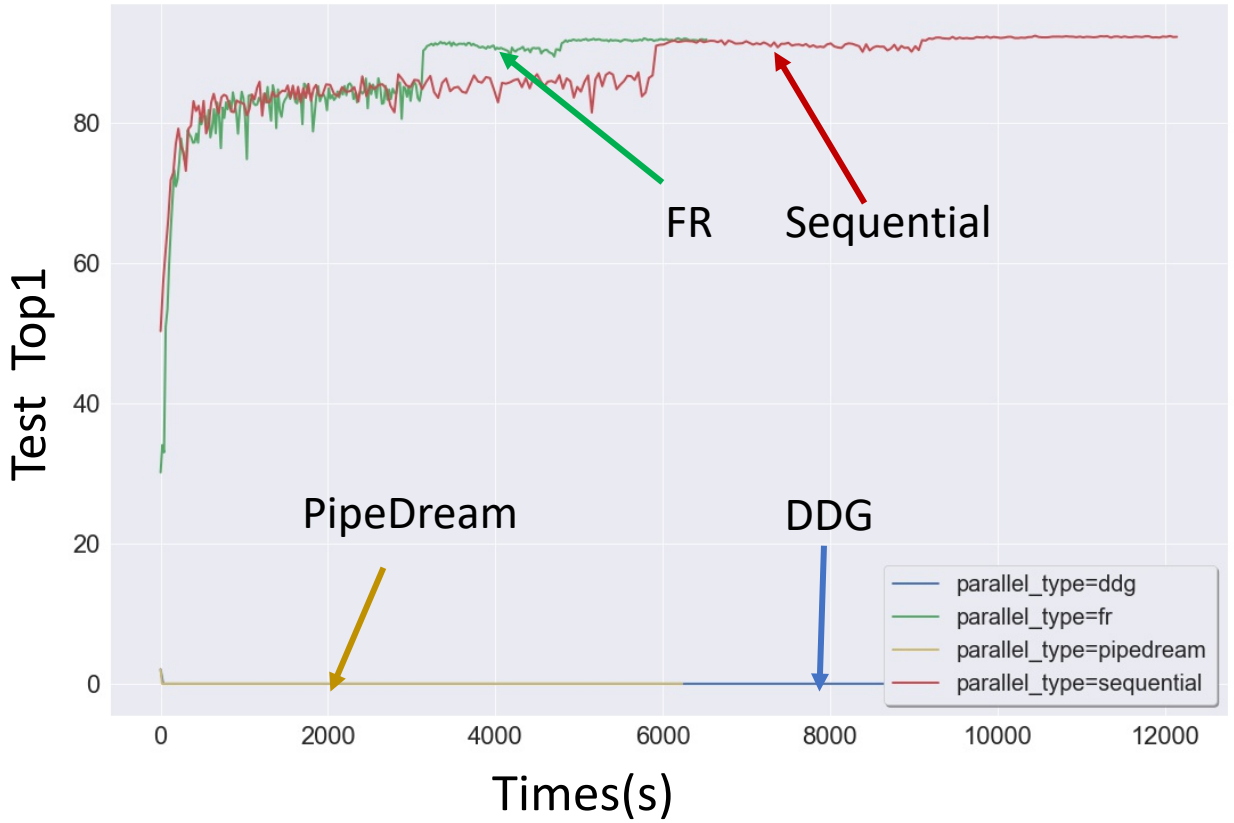


- From Test Top1 vs. Epoch => Number of epochs to reach a given accuracy is almost the same.
- From Test Top1 vs. Times(s) => the time needed to train an epoch PipeDream < DDG < FR < Sequential

Evaluation: ResNet20

ResNet20 split to 11 blocks, lr is 0.32





- DDG and PipeDream divergence, while FR still convergence
  needed exploring further (for example: use resnet110 to test more case)

Evaluation: BERT

simplified BERT

Encoder layers L=3
hidden size H=768
self-attention heads A=12
feed-forward size 3072

(in original paper[1],
Encoder layer is 12)

Two unsupervised prediction tasks

- **Masked Language Modeling**

  mask 15% of the input tokens at random
  predicting only those masked tokens

- **Next Sentence Prediction**

  When choosing the sentences A and B
  50% of the time B is the truly next sentence,
  while 50% of the time not.

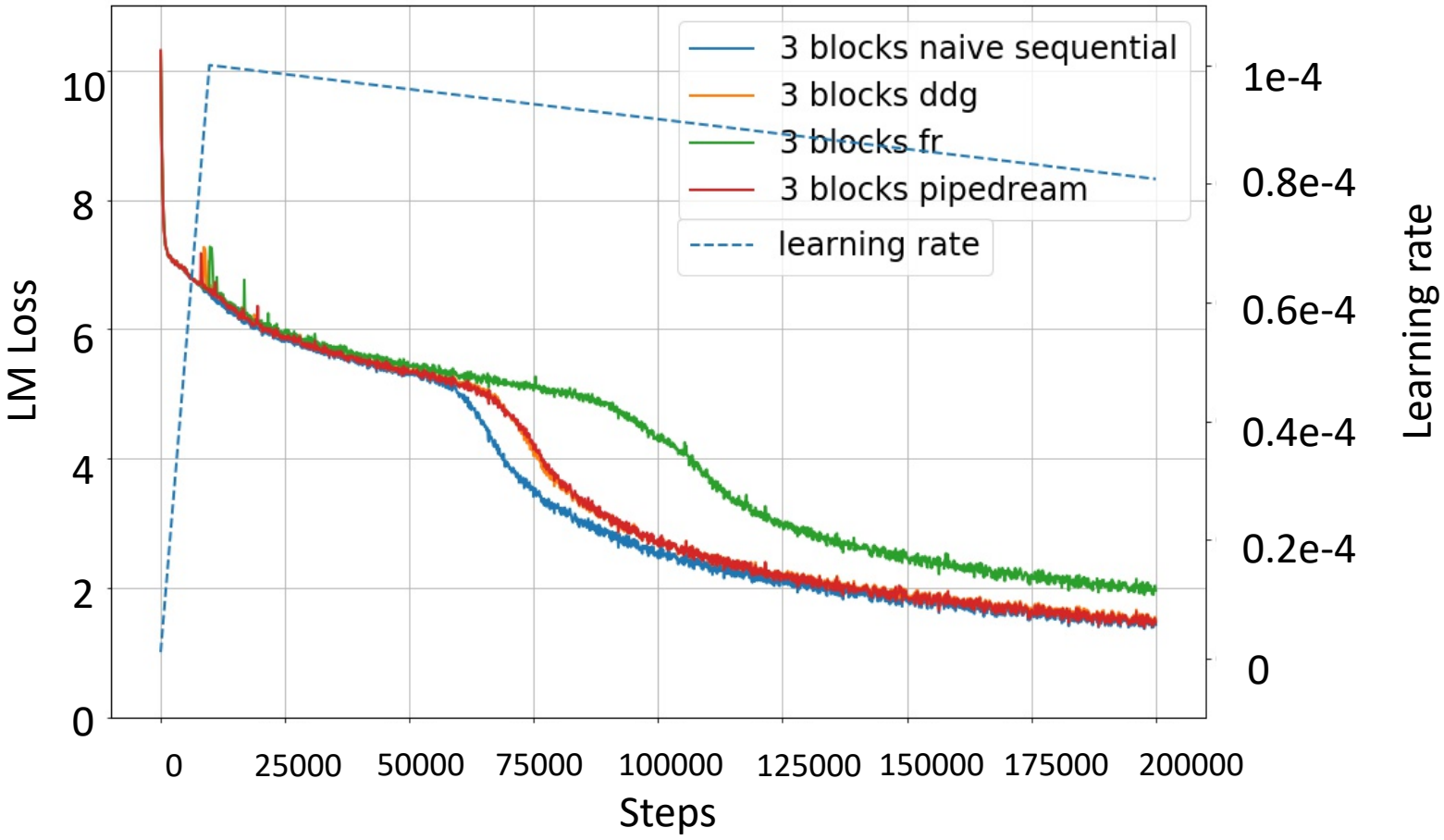
- **Dataset**

  English Wikipedia, Extract according[2]
  During experiment, we only use 11M data out of about 16G data

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre- training of Deep Bidirectional Transformers for Language Understanding. arXiv e-prints, page arXiv:1810.04805, Oct 2018.

[2] Giuseppe Attardi. A tool for extracting plain text from wikipedia dumps. https: //github.com/attardi/wikiextractor, 2019.

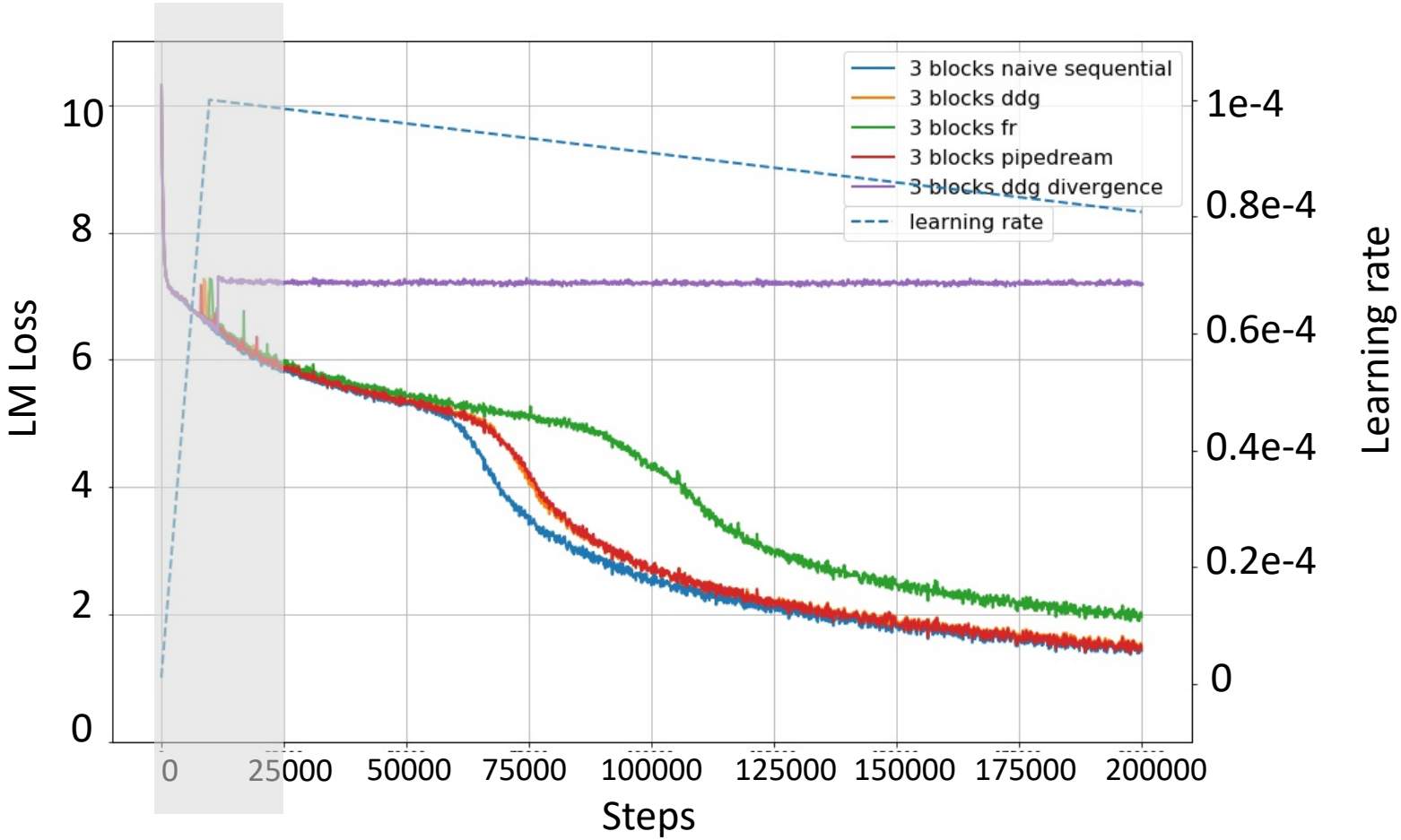
Evaluation: BERT



sentence length is 512
batch size: 4 sentences
Steps: 200000(about 2 epochs)

- DDG and PipeDream can reach almost the same LM train loss at the end as Naive Sequential
- FR can't reach the same LM train loss as the Naive Sequential
- DDG and PipeDream need more steps to reach a given LM loss than Naive Sequential
  FR is worse than DDG and PipeDream

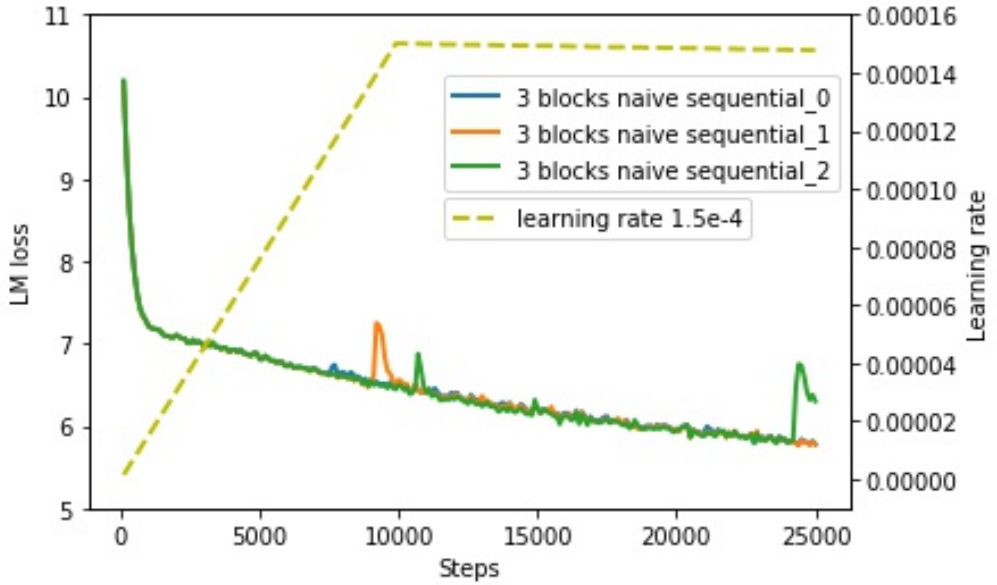
Evaluation: BERT



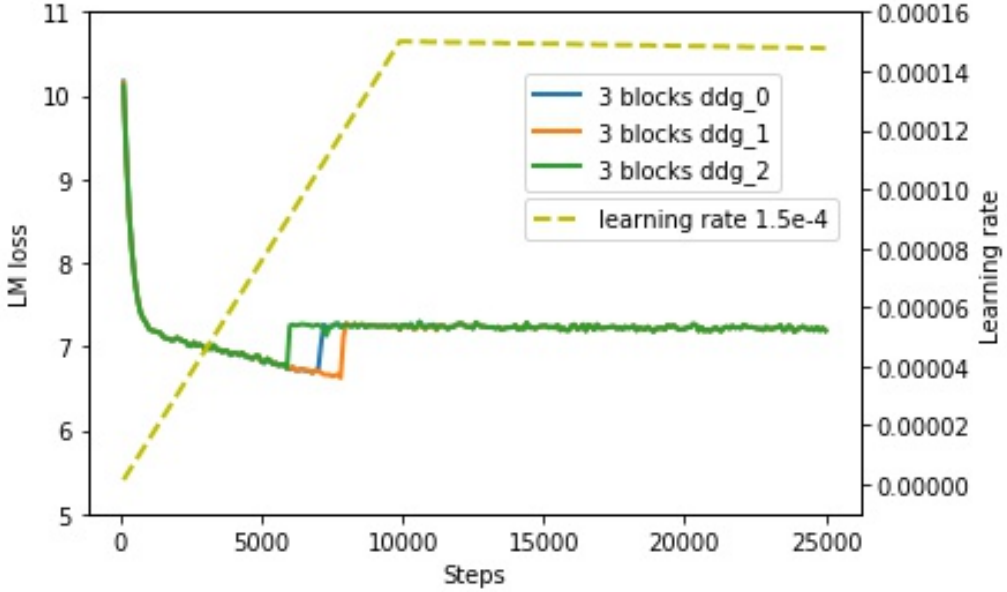
Encounter divergence: When learning rate is close to the set learning rate.

Focus on steps 0- 25000, Set learning rate to be [1.5e-4, 1e-4, 0.5e-4], and run three times for each learning rate

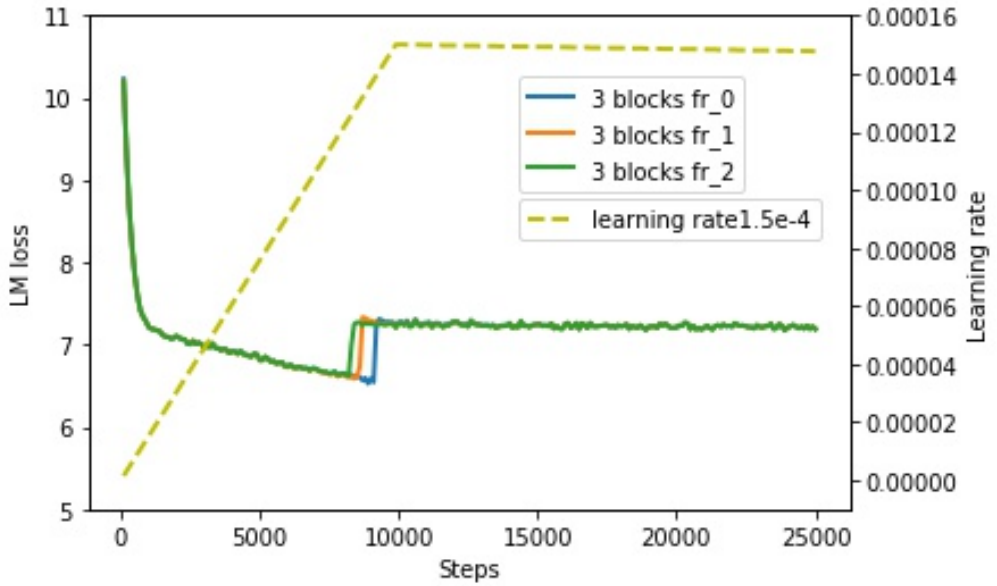
Learning rate: 1.5e-4



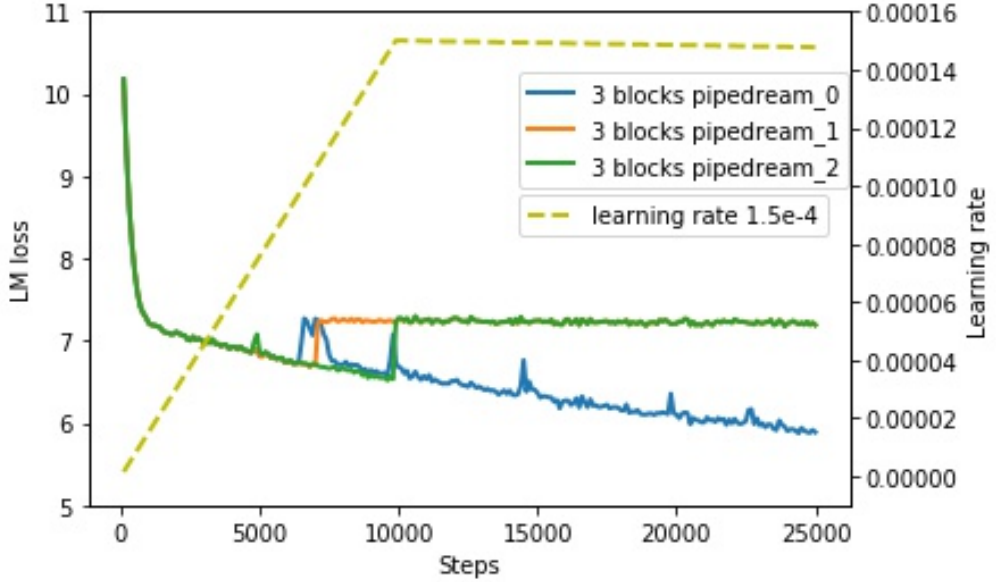
Sequential



DDG

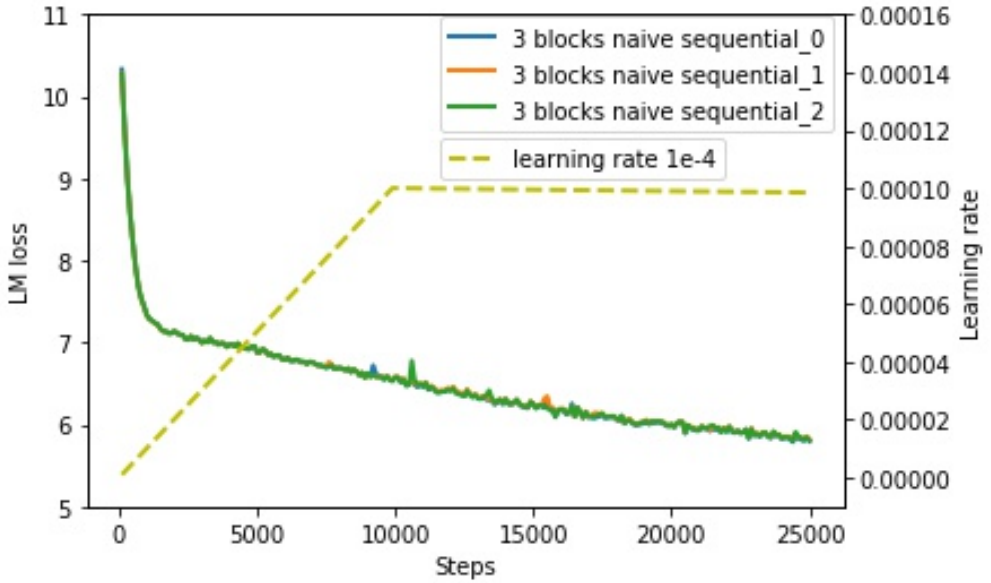


FR

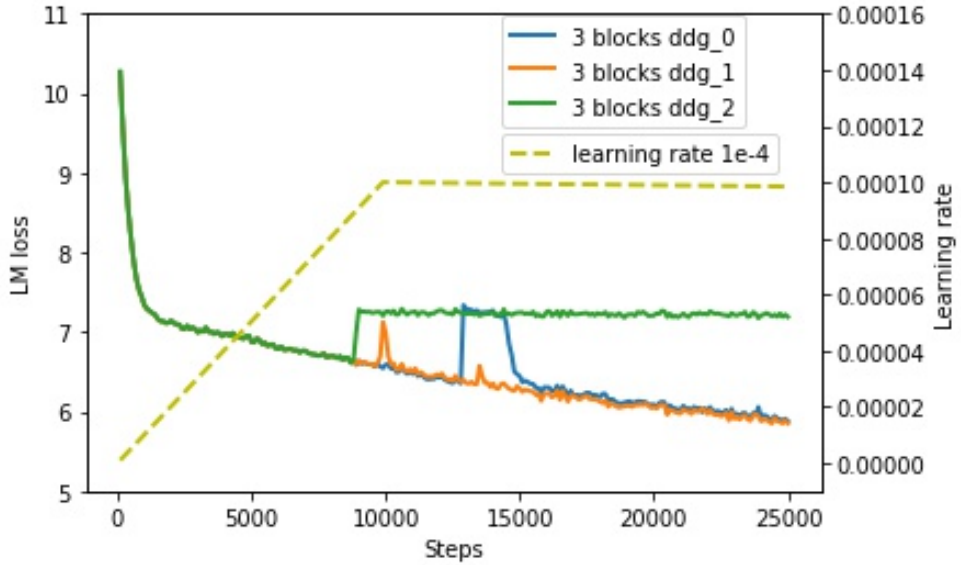


PipeDream

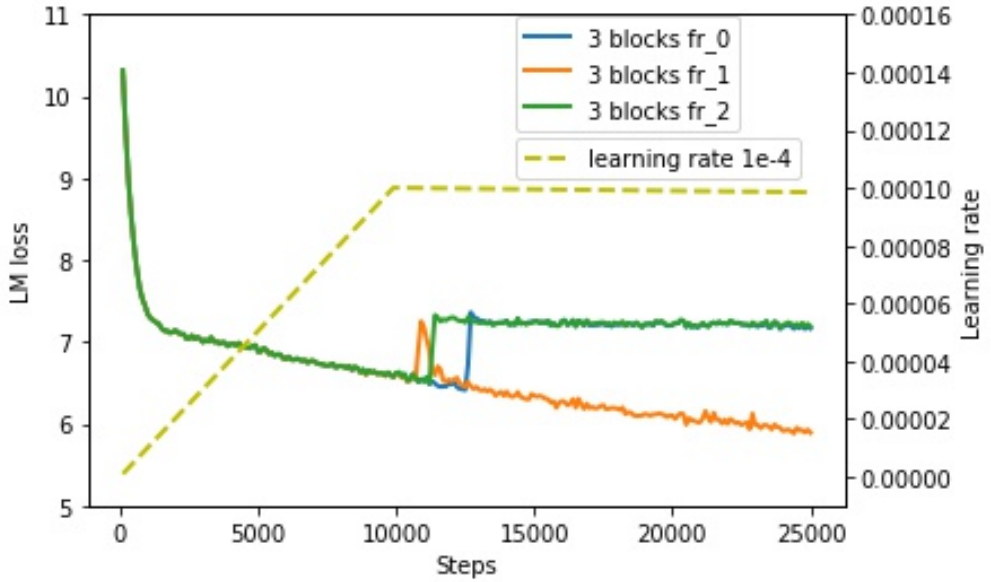
Learning rate: 1e-4



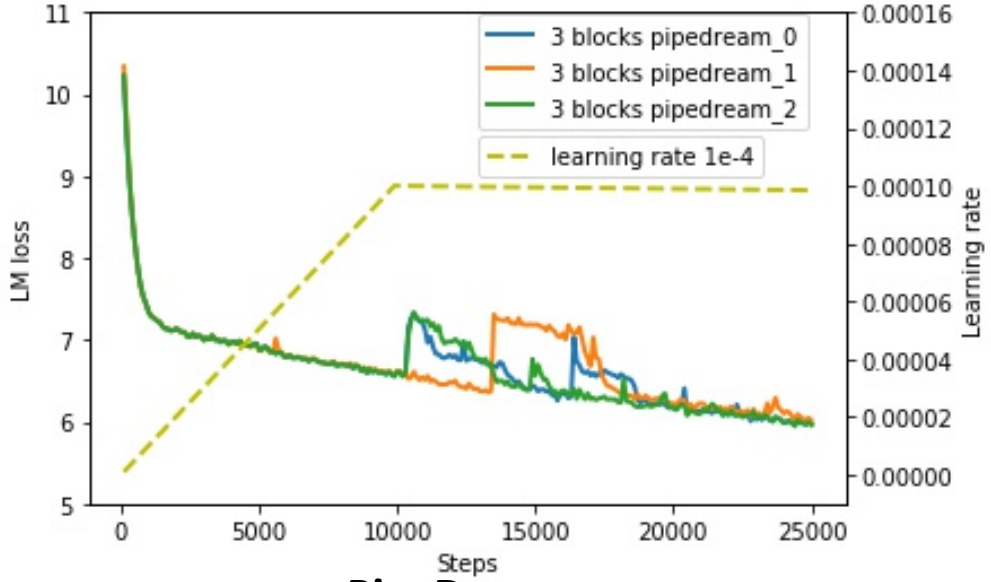
Sequential



DDG

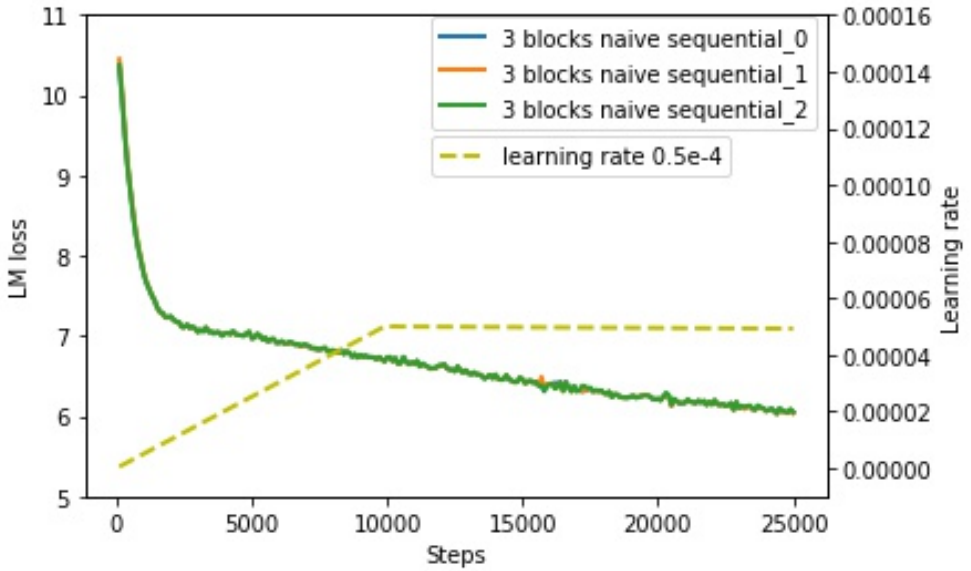


FR

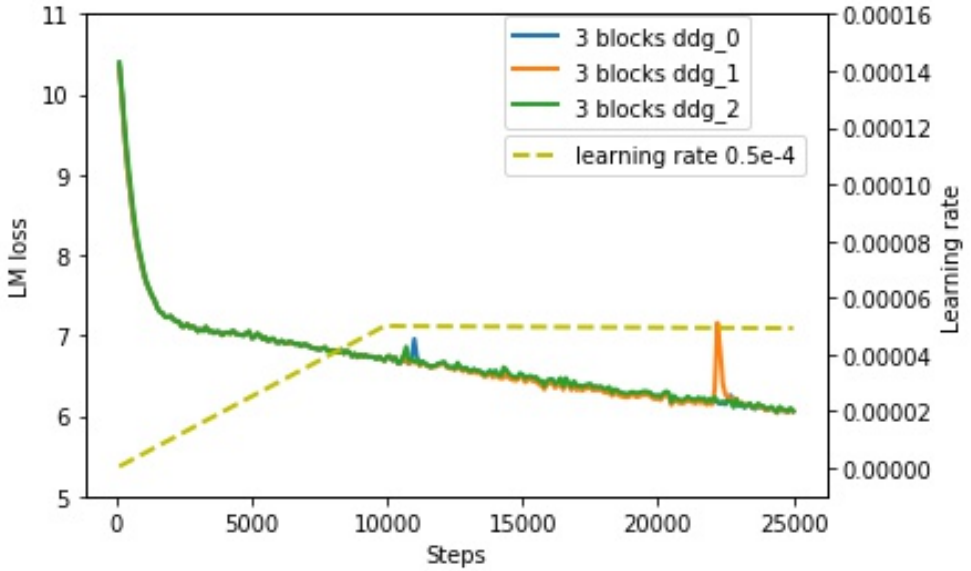


PipeDream

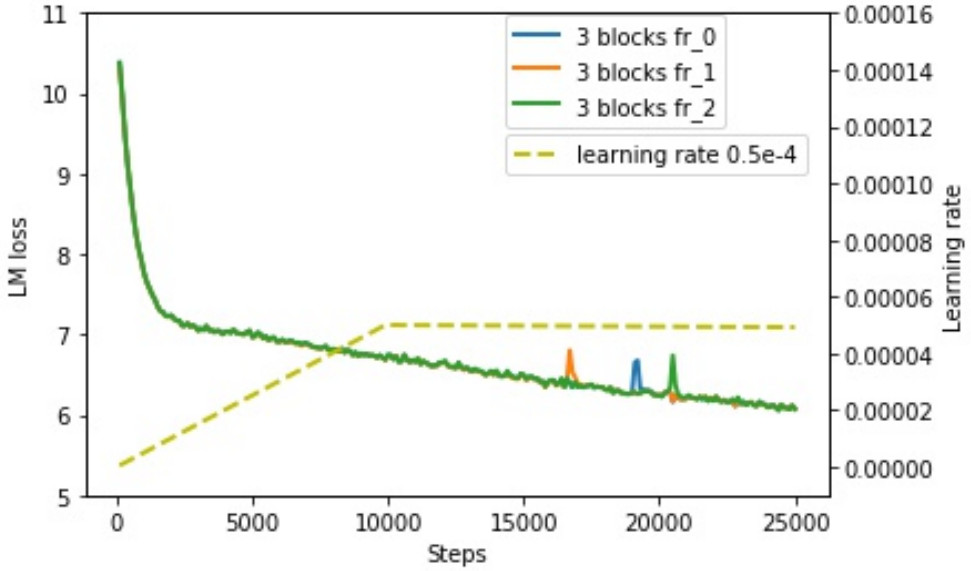
Learning rate: 1e-4



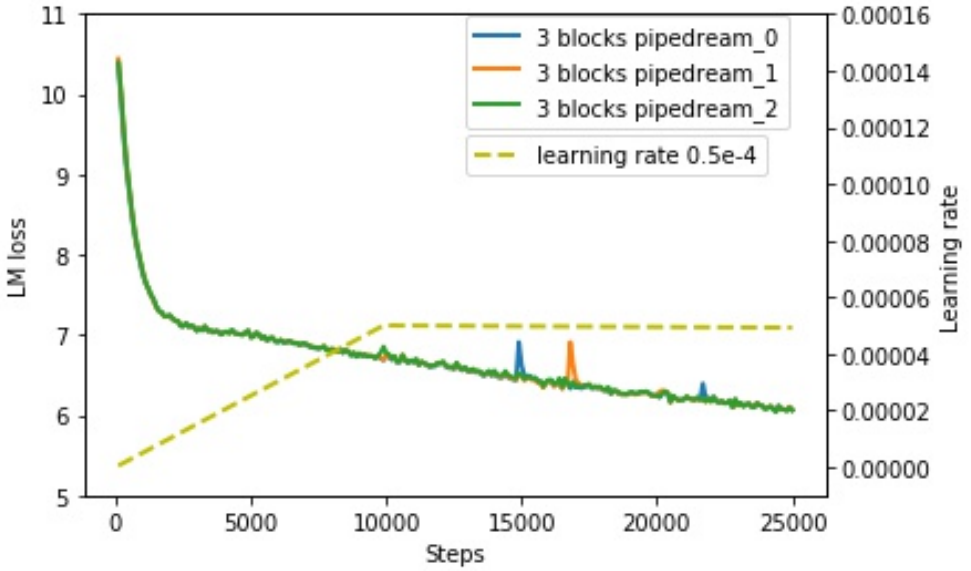
Sequential



DDG



FR



PipeDream

Evaluation: BERT

Learning Rate	Sequential	DDG	FR	PipeDream
	Divergence	Divergence	Divergence	Divergence
1.5e-4	0/3	**3/3**	**3/3**	**2/3**
1e-4	0/3	**1/3**	**2/3**	0/3
0.5e-4	0/3	0/3	0/3	0/3

Conclusion

For resnet20

- the best accuracy of top1 is reduced slightly
- When the model is divided into more blocks, the loss of top1 accuracy is more serious.
- When the number of blocks is [2,3,4,6], FR is slightly worse than DDG and PipeDream
- When the number of blocks is [11], FR is better than than DDG and PipeDream

When the number of block is small, forward and backward inconsistency(FR) is crucial
When the number of block is large, the weight staleness(DDG and PipeDream) becomes crucial and can't be ignored

For simplified BERT model

- The number of steps needed to reach a given LM Loss is increased
  FR is much worse than DDG and PipeDream.
- when the learning rate is big(≥ 1e-4)), DDG, FR and PipeDream may cause divergence
  FR is worse than DDG and PipeDream.

Future work

- Currently partition the model uniformly, Better partition the model

  the total computation time across the forward and backward pass for the layer
  the size of the output activations of the layer
  the size of parameters for layer
- ResNet110 on datasets: CIFAR-100

  To further understand the effect of weight staleness and forward backward inconsistency
- Bert train with more Encoder layers and different number of blocks
- Combine model parallel with data parallel
- Maybe can try multithread to parallel communication and training