Pipelined Backpropagation at Scale:
Training Large Models without Batches

Atli Kosson, **Vitaliy Chiley**, Abhinav Venigalla, Joel Hestness, Urs Köster
Machine Learning Team

Motivation

Researchers rely on parallelism to accelerate deep learning training workloads

Data (batch) parallelism can scale training to large batch sizes but has considerable overheads which limit overall hardware utilization.

Network	System	#GPUs	Utilization
ResNet50	A100 DGX	1	16.4%
ResNet50	A100 DGX	8	15.9%
$\text{BERT}_{\text{Large}}$	A100 DGX	8	36.8%

Based on max advertised FLOPS in mixed precision

Parallel Training

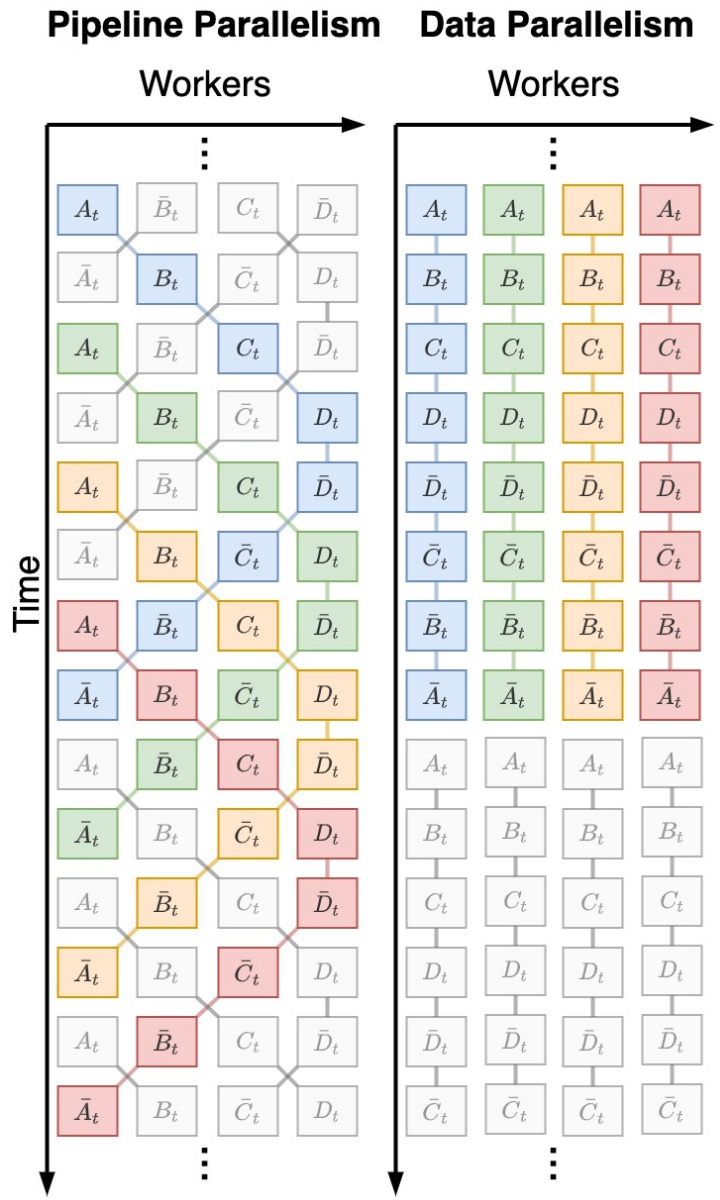
Goal: Parallelize training of a neural network with operations A, B, C, D

Data Parallelism (standard; maps well to GPU)

- Layer Sequential training
- All workers compute the same transformation on different data at a given time

Fine-Grained Pipeline Parallelism

- Layer Parallel training
- Each worker performs a single transformation on sequentially incoming data
- Enables worker specialization



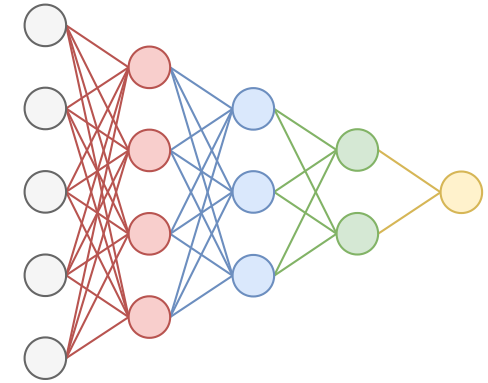
Pipelined Processing on a CGRA

Coarse-Grained Reconfigurable Array (CGRA)

- Mesh of locally connected workers (w)
- Distributed high bandwidth, low latency memory architecture

Spatially distributing network layers

- Near optimal resource allocation
- Persistent kernel execution
- Local weight and gradient storage
- Pipelined Training can result in improved throughput and energy efficiency
  - Dense training: up to a 3.5x (Zhang et al., 2019c; Li & Pedram, 2017; Chen et al., 2016)
  - Sparse training: 42.5x improvement in throughput and a 11.3x improvement in energy efficiency (Chen et al., 2019)
- Activation memory complexity: O(pipeline depth^2)
  - Use small micro batch processing to compensate



Deep Network Training

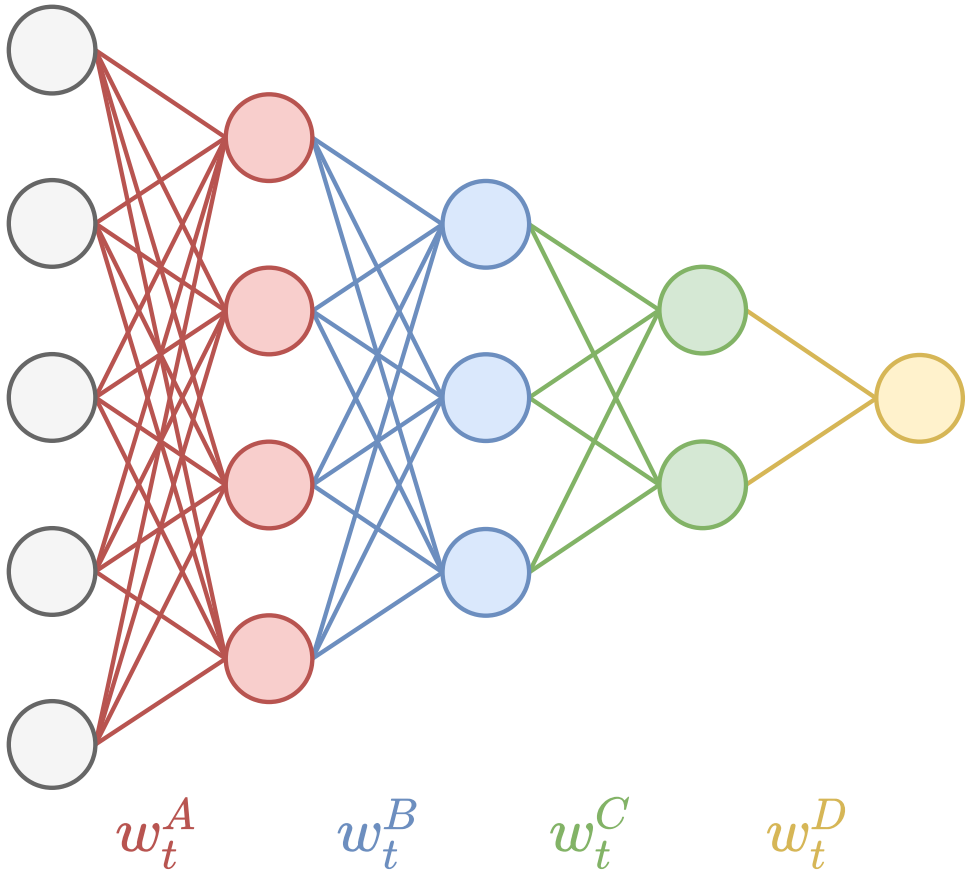
SGD:

$$w_{t+1} = w_t - \eta \nabla_{w_t} L = w_t - \eta g_t$$

SGDM:

$$v_{t+1} = m v_t + g_t$$

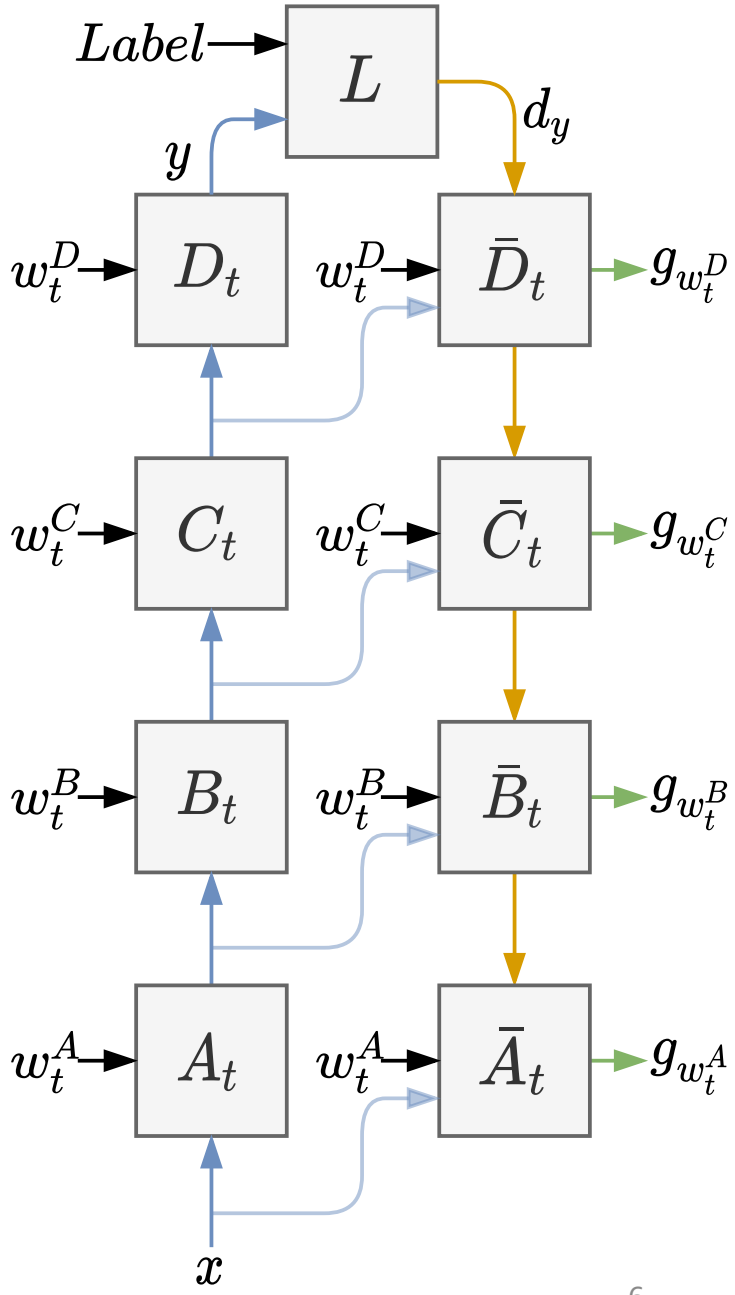
$$w_{t+1} = w_t - \eta v_{t+1}$$



Backpropagation

Efficient way to calculate gradients

- Forward Pass
  - Calculates activations through a series of operations (A,B,C,D)
- Backward Pass
  - Calculates deltas (activation gradients) and parameter gradients
  - **Need to use the same parameters**
- Update parameters based on gradient



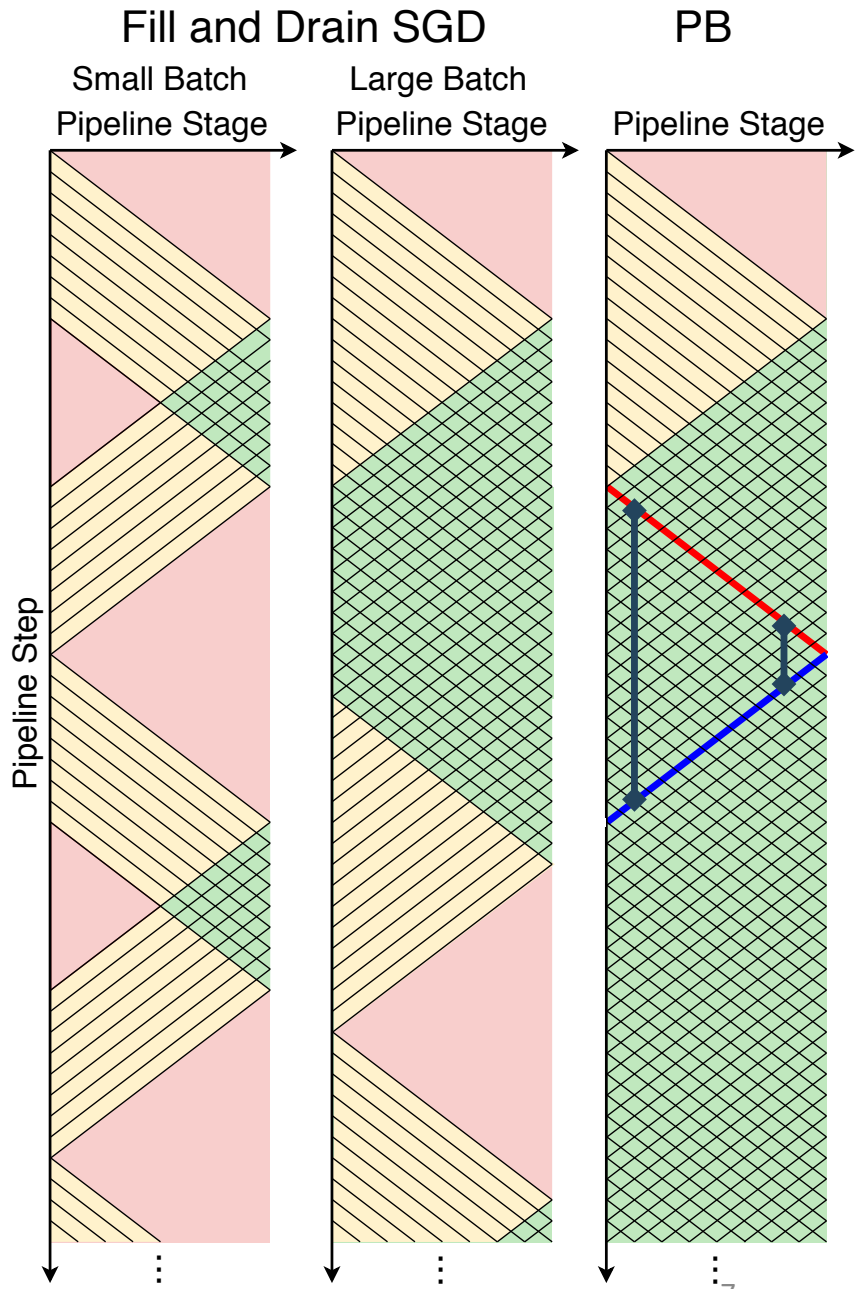
Pipeline Parallel Training

Fill and Drain SGD

- Empty the pipeline before updating the weights
- Lowers Utilization: $\approx \frac{N}{N+2S}$
  - Batch size: N; Number of pipeline stages: S

Pipelined Backpropagation (PB)

- Introduced by Pétrowski et al. (1993)
- Update weights without draining pipeline
- High utilization
- Inexact gradients



Inexact Gradients in PB

Weights are updated during gradient computation

Weight Inconsistency

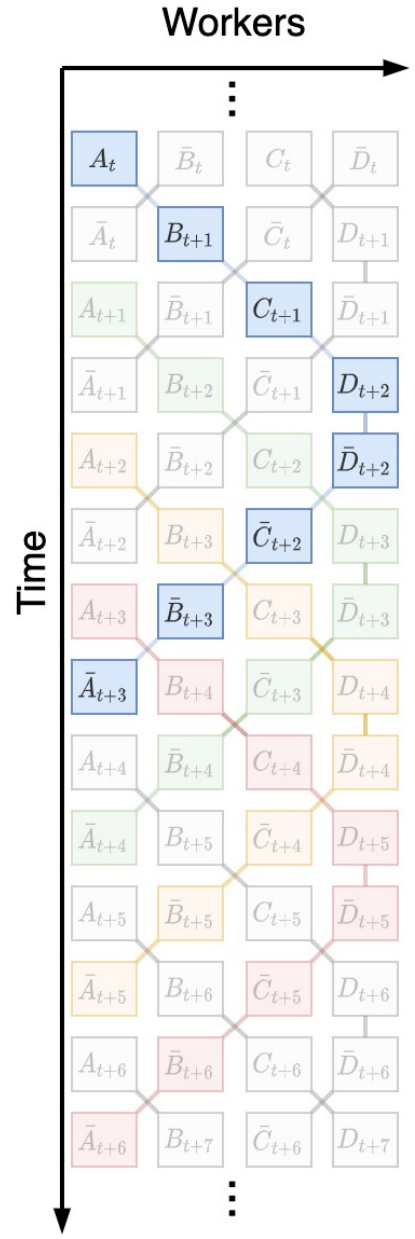
- Different weights used on the fwd and bwd passes

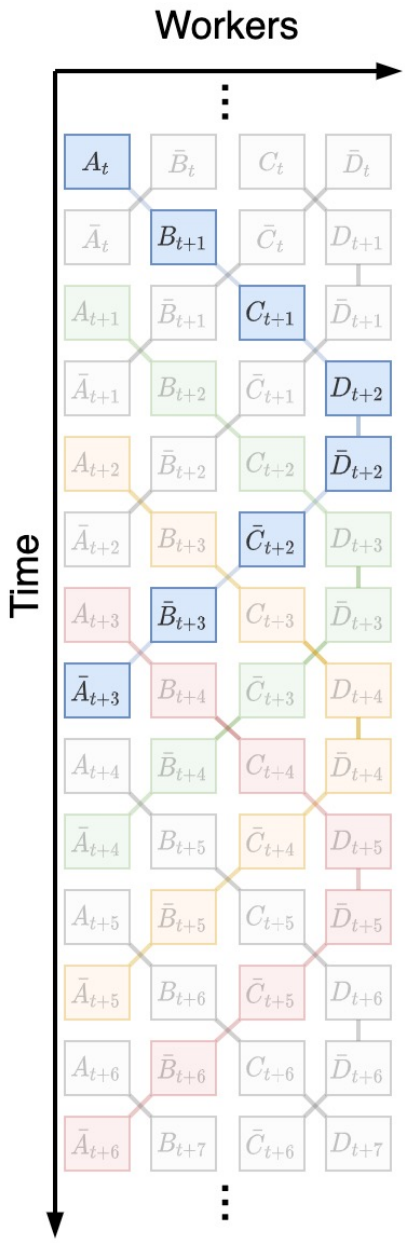
Gradient Delay

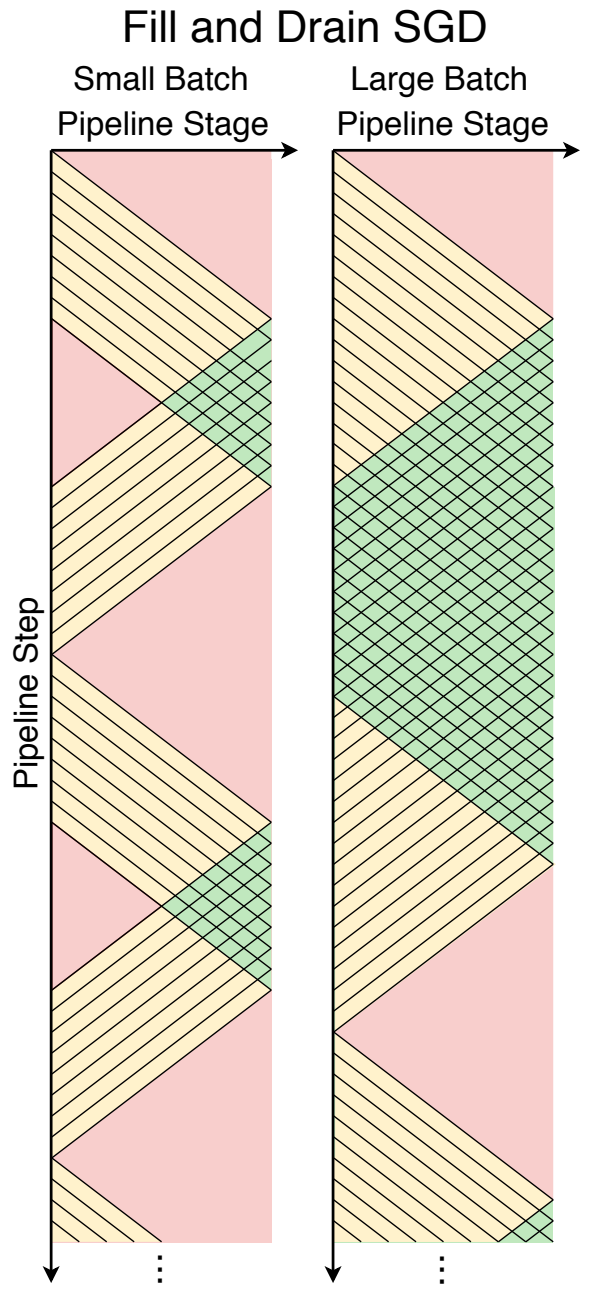
- Weights used to calculate a gradient are old when the resulting gradient is used to update the weights

Impacts training

- Loss of final accuracy
- Potential training instability
- Effect depends on the number of stages




Workers
Time


Fill and Drain SGD
Small Batch
Pipeline Stage
Large Batch
Pipeline Stage
Pipeline Step

Mitigation Methods

Our work focuses on mitigating the effects of Pipelined Backpropogation

- **Linear Weight Prediction**
  - Changes how the gradient is computed
- **Spike Compensation**
  - Changes how the gradient is applied
- Delayed SGDM:

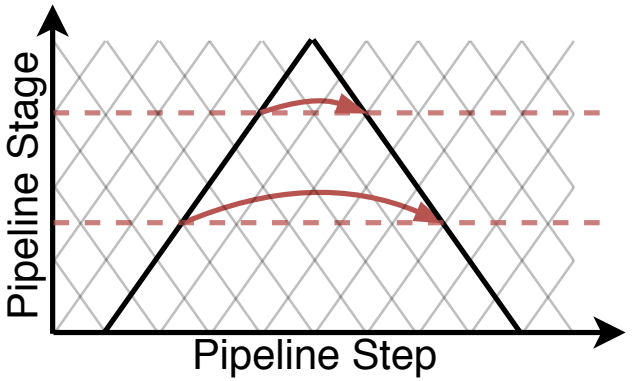
$$g_t = G\left(w_{t-D}\right)$$
$$v_{t+1} = mv_t + g_t$$
$$w_{t+1} = w_t - \eta v_{t+1}$$

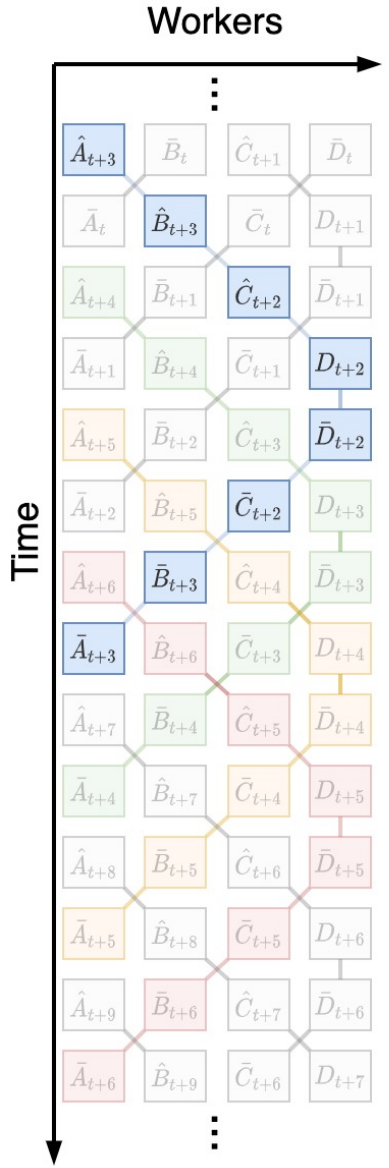
Linear Weight Prediction (LWP)

Approximates the future backwards pass weights for use in the forward pass

- Assumes weight updates are roughly constant over the delay
  - Holds well with large momentum or small delays

$$g_t = G\left(w_{t-D} - \eta T v_{t-D}\right)$$
$$v_{t+1} = m v_t + g_t$$
$$w_{t+1} = w_t - \eta v_{t+1}$$

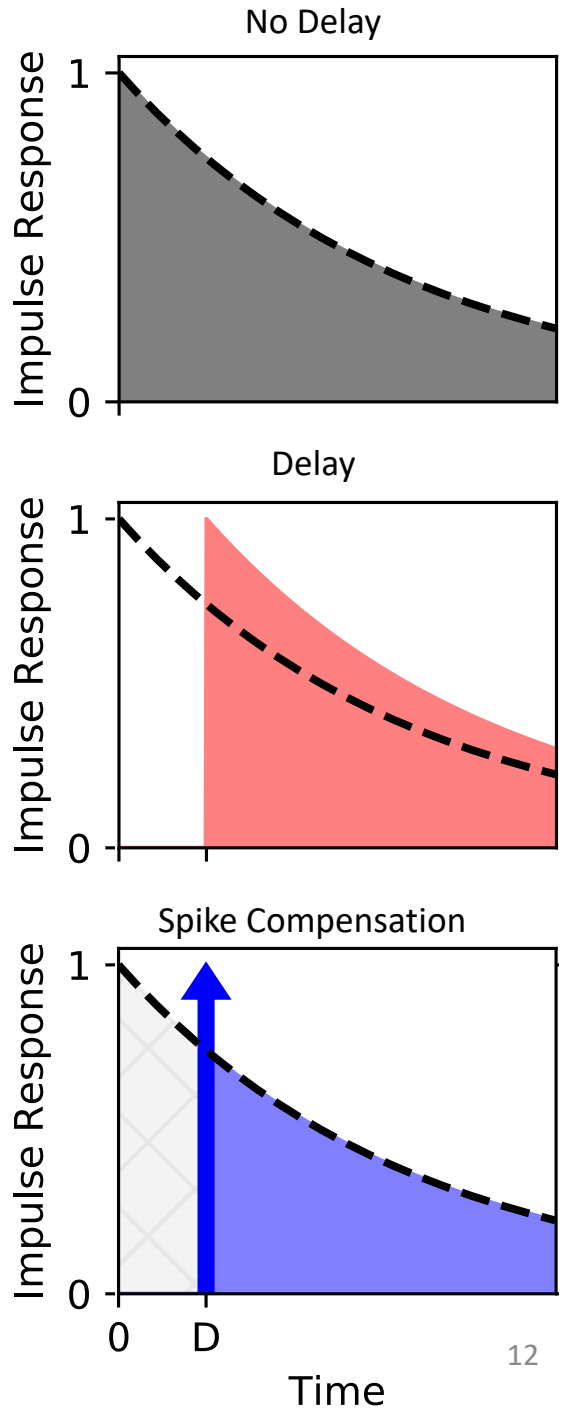




Spike Compensation (SC)

- Momentum in SGDM
  - Exponentially smoothed gradients used for weight updates
  - “Impulse response” is exponential
- Spike Compensation
  - Apply “missing” weight updates immediately (spike)
  - Match no-delay impulse response afterwards

$$g_t = G\left(w_{t-D}\right)$$
$$v_{t+1} = mv_t + g_t$$
$$w_{t+1} = w_t - \eta \cdot \left(m^D v_{t+1} + \frac{1-m^D}{1-m} g_t\right)$$

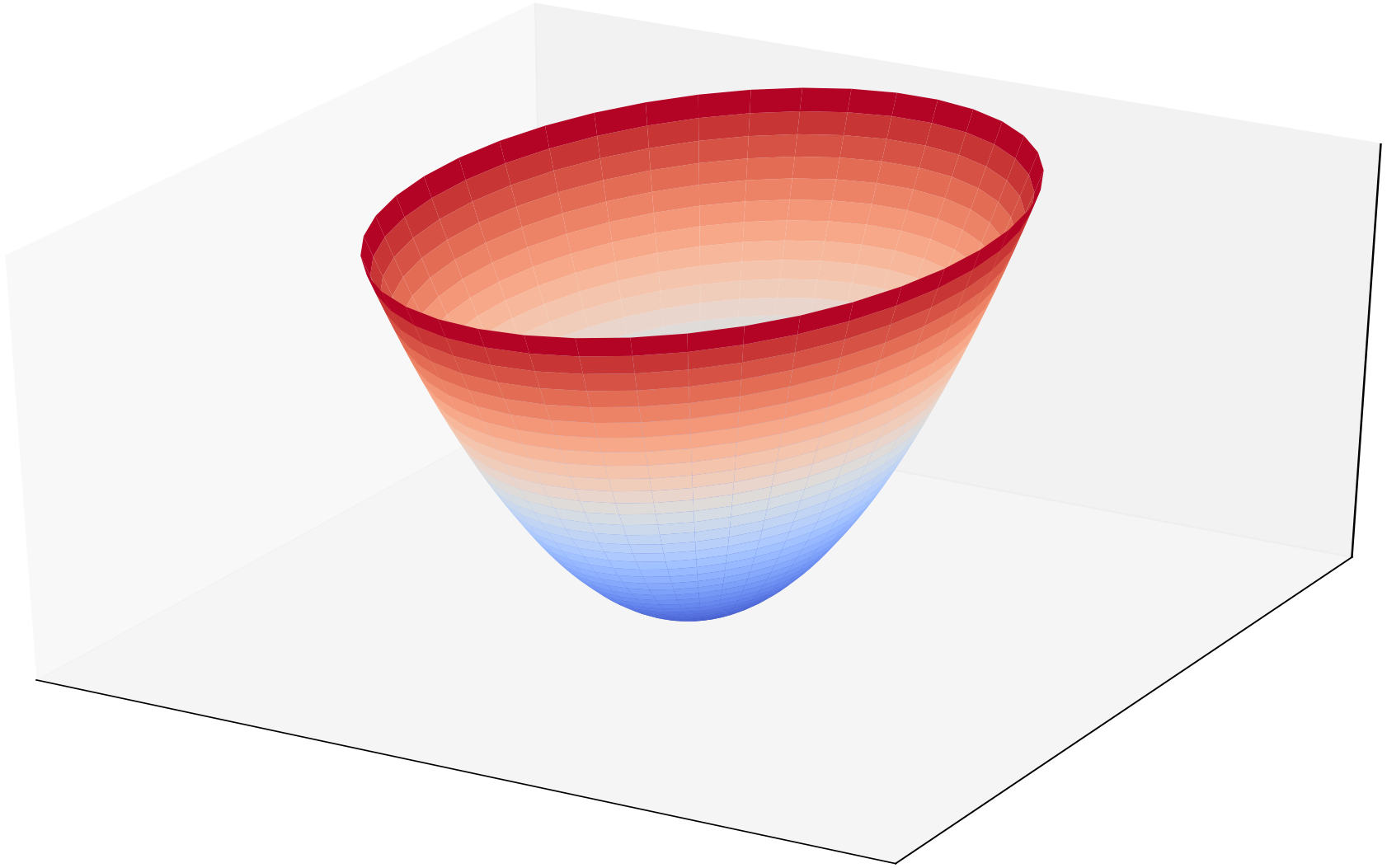


LWP + SC

Combining our methods overcompensates for delay

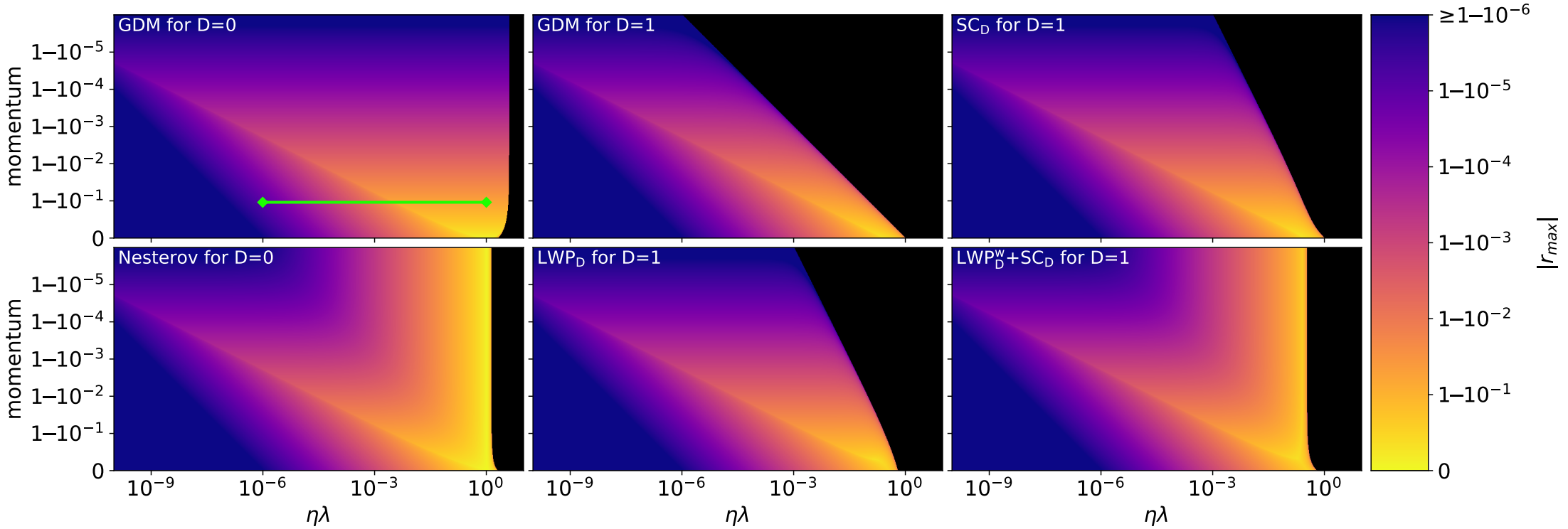
$$
\begin{aligned}
g_t &= G\left(w_{t-D} - \eta T v_{t-D}\right) \\
v_{t+1} &= m v_t + g_t \\
w_{t+1} &= w_t - \eta \cdot \left(m^D v_{t+1} + \frac{1-m^D}{1-m} g_t\right)
\end{aligned}
$$

Delayed Quadratic Optimization

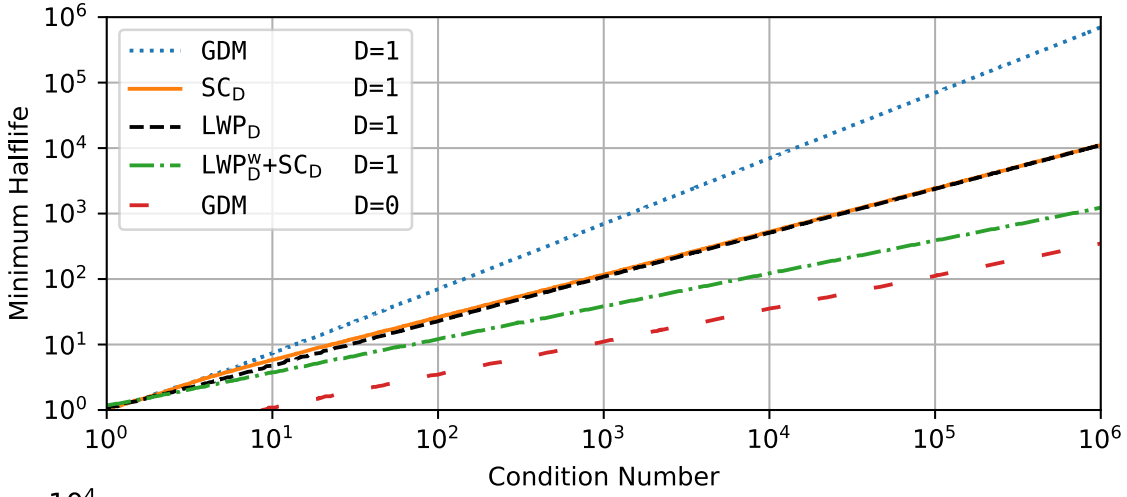


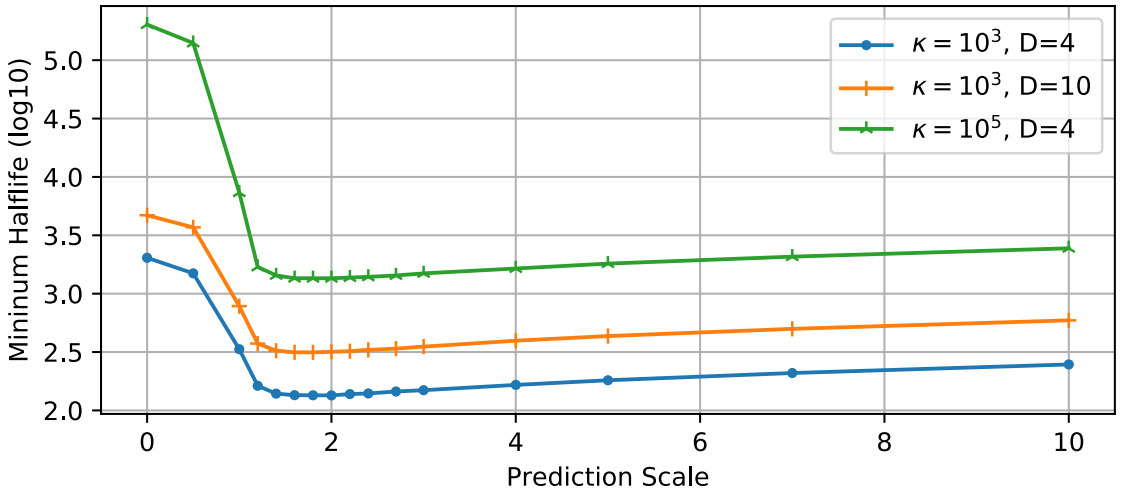
Delayed Quadratic Optimization

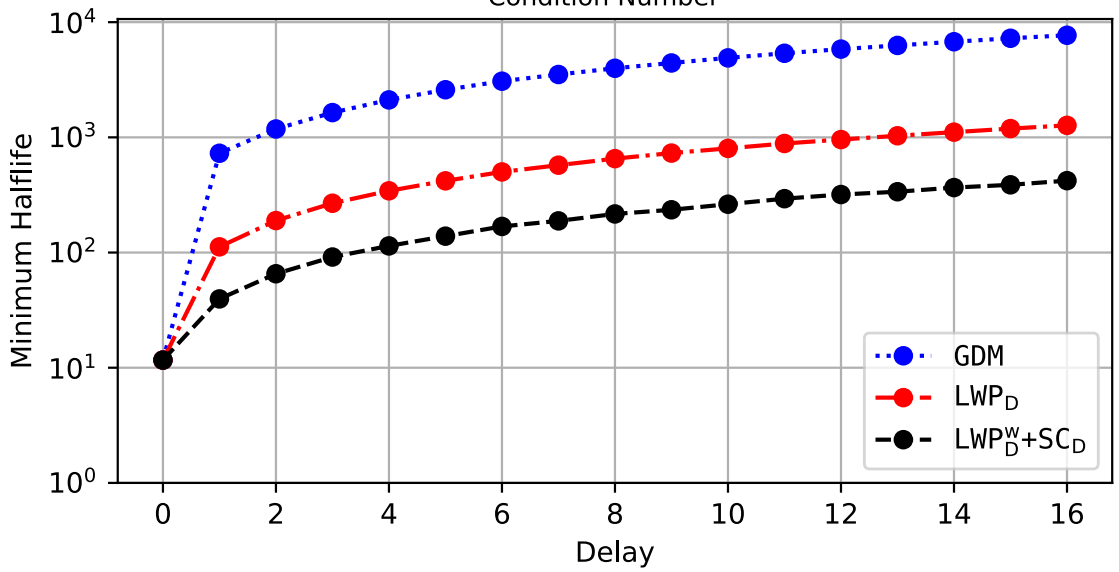
Analyze the characteristic polynomial of the linear recurrence relation of the state transition equations.

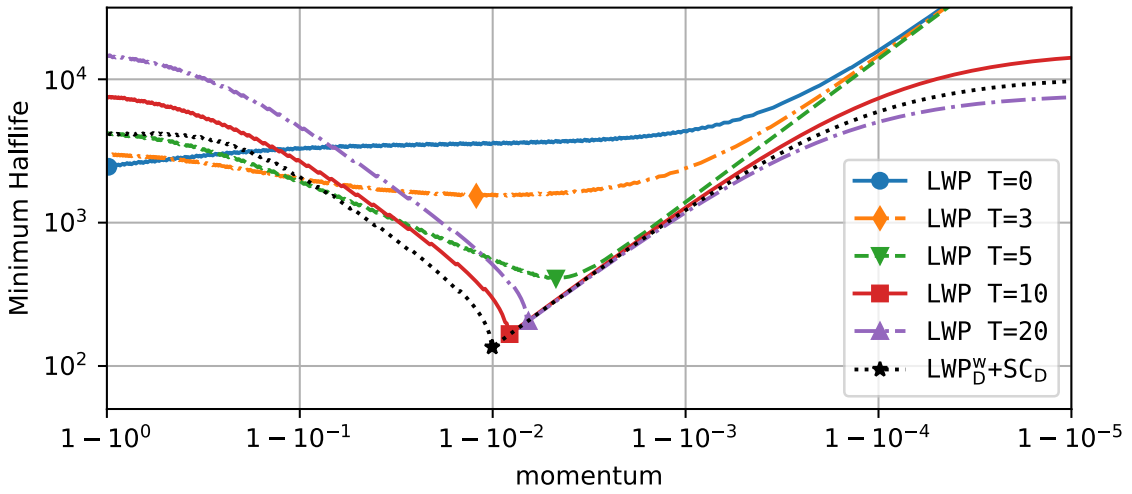


Delayed Quadratic Optimization

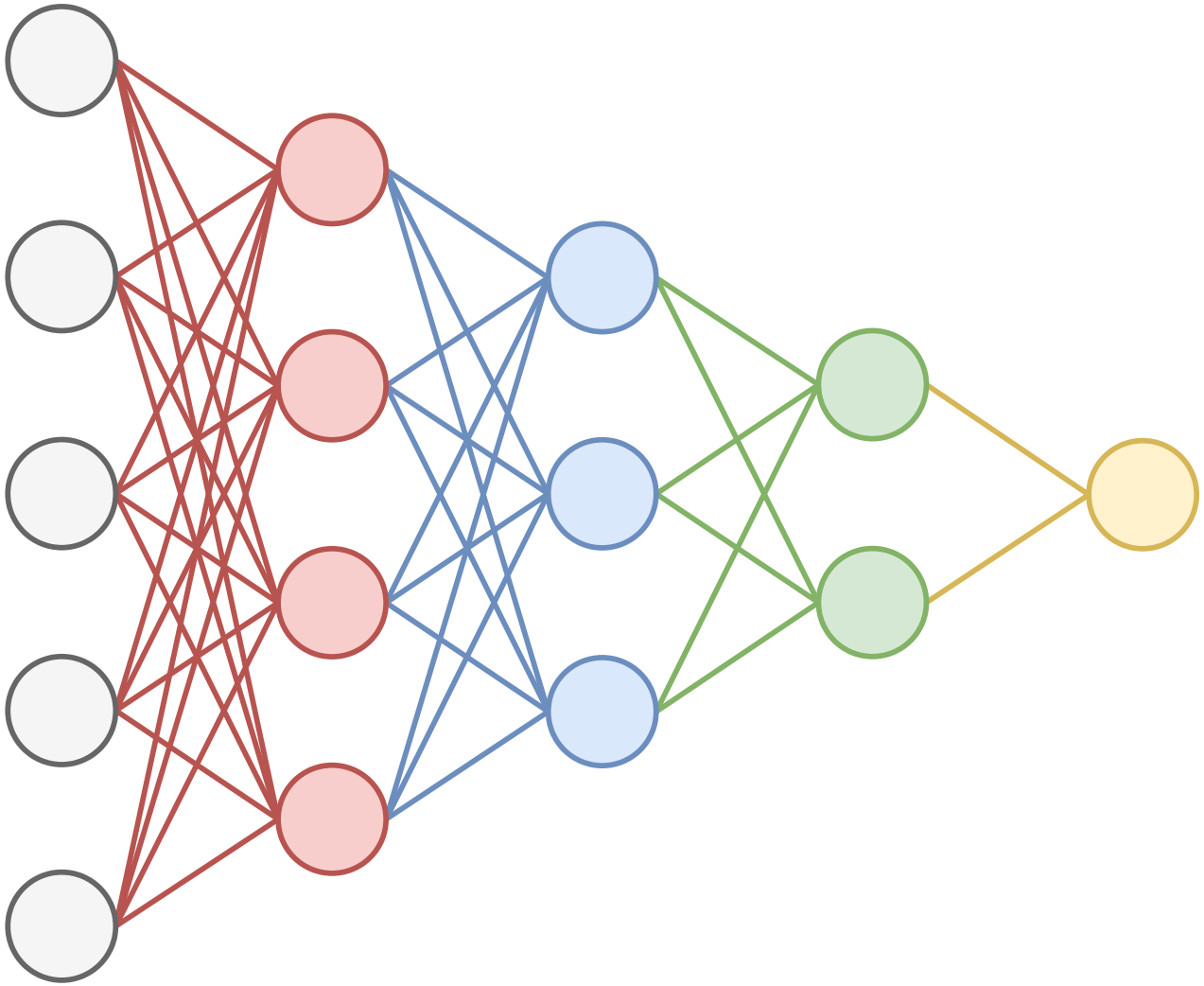








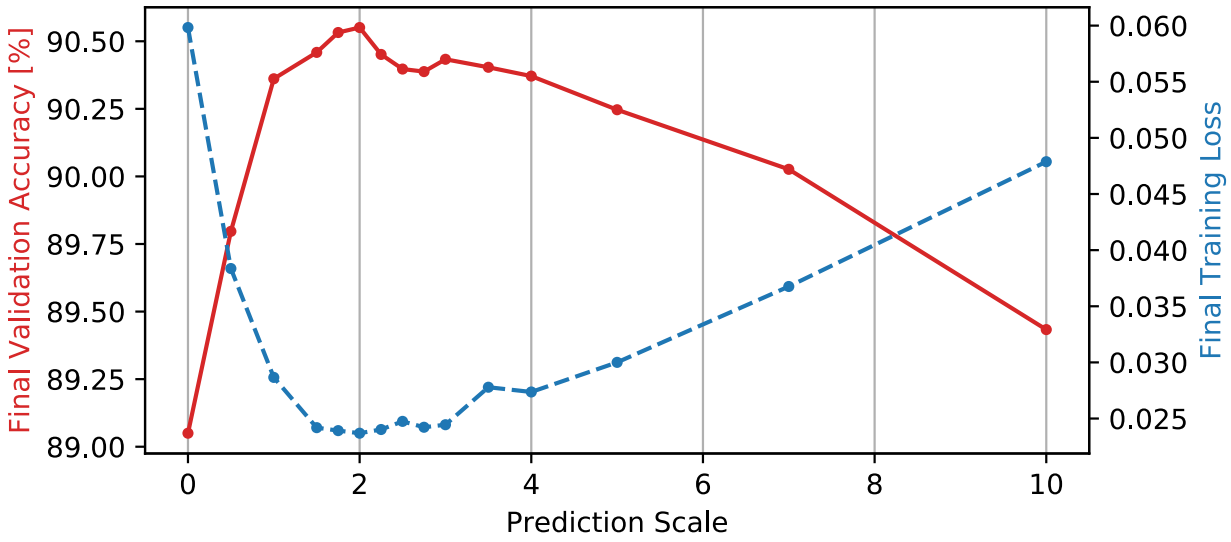
PB Training of Neural Networks



PB Training of Neural Networks

Table 3. CIFAR-10 validation accuracy (mean±std.dev of 5 runs) when tuning the learning rate (LR) for ResNet-20 with GN training. The learning rate shown is used for batch size 128 SGDM training and is adjusted for batch size one PB training.

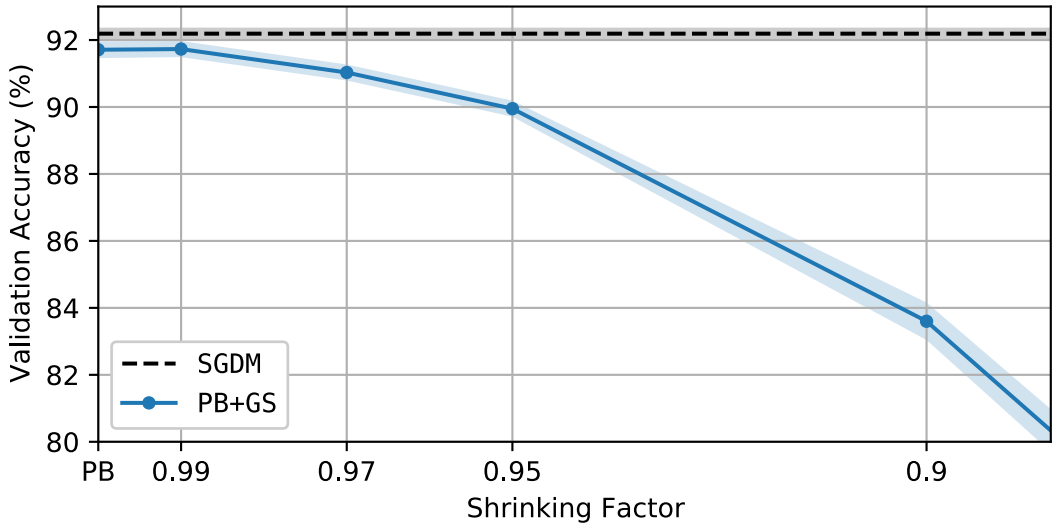
LR	SGDM	PB	PB+LWP$^{v}_{D}$+SC$_{D}$
0.0125	88.76±0.45	88.77±0.22	**89.32**±0.26
0.025	89.88±0.32	89.55±0.35	**90.06**±0.23
0.05	90.47±0.22	90.10±0.40	**90.80**±0.37
0.1	90.63±0.31	90.44±0.24	**90.92**±0.25
0.2	90.69±0.25	90.22±0.11	**90.89**±0.28
0.4	89.54±0.32	88.82±0.32	**89.93**±0.20
0.8	69.16±33.08 [2]	83.53±1.39	**88.01**±0.56



Mitigation Strategies

Table 2. CIFAR-10 final validation accuracy (mean±std.dev of 5 runs) for ResNet (RN) with group normalization and VGG training.

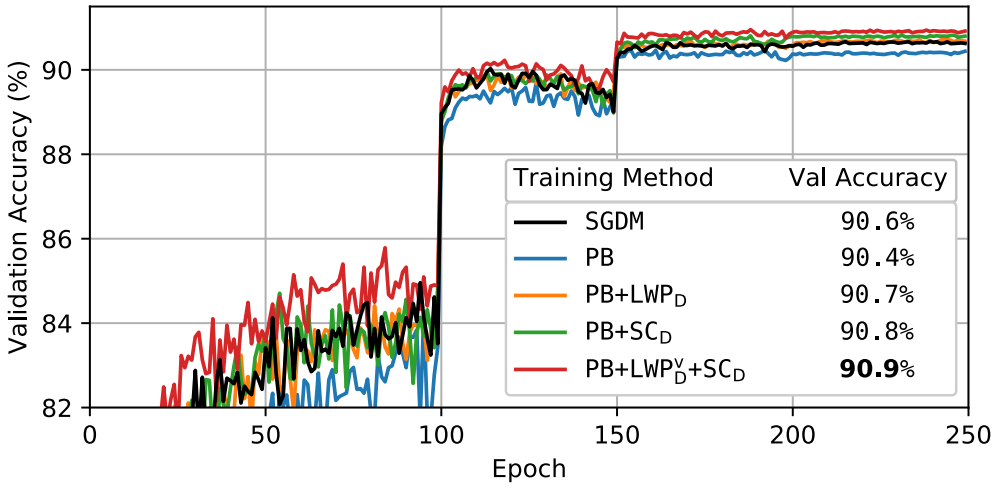
NETWORK	STAGES	SGDM	PB	PIPEDREAM	PB+LWP$_D$	PB+SC$_D$	PB+LWP^v_D+SC$_D$
VGG11	29	**91.16**±0.19	90.83±0.20	90.93±0.12	91.05±0.11	**91.08**±0.19	**91.12**±0.18
VGG13	33	**92.57**±0.15	**92.59**±0.15	92.30±0.24	92.51±0.11	92.38±0.27	**92.56**±0.14
VGG16	39	92.24±0.19	92.06±0.21	59.31±45.01[2]	92.22±0.24	**92.45**±0.30	**92.38**±0.27
RN20	34	90.63±0.31	90.44±0.24	90.36±0.06	90.68±0.30	**90.80**±0.29	**90.92**±0.25
RN32	52	91.68±0.23	91.46±0.09	91.40±0.28	91.66±0.10	91.55±0.14	**92.04**±0.13
RN44	70	**92.19**±0.14	91.71±0.25	91.72±0.14	92.00±0.14	**92.13**±0.16	**92.16**±0.26
RN56	88	92.39±0.20	91.89±0.40	91.82±0.19	92.31±0.14	92.33±0.16	**92.48**±0.11
RN110	169	**92.77**±0.22	91.81±0.15	91.92±0.33	**92.76**±0.05	92.28±0.29	92.41±0.16

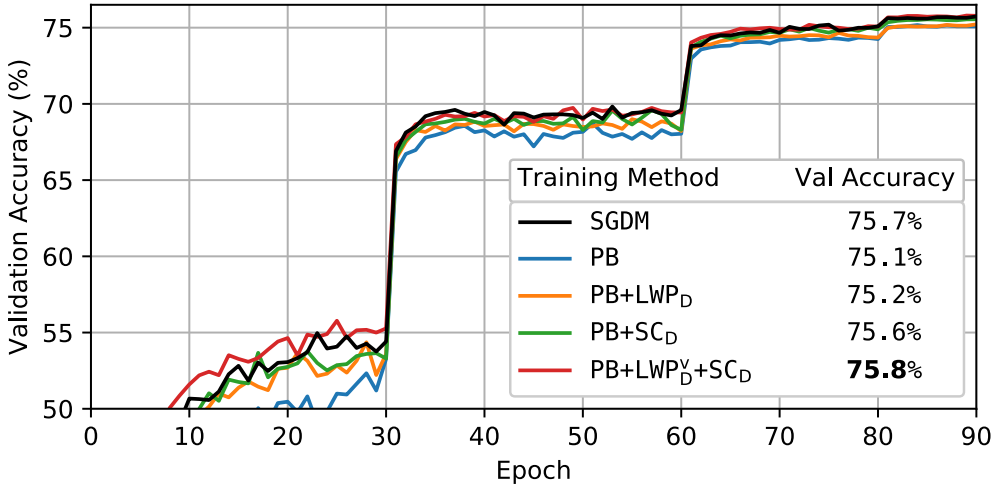


Mitigation Strategies

Table 4. CIFAR-10 (C10) validation accuracy (mean±std.dev of five runs) and ImageNet (I1k) validation accuracy (single run) comparing SpecTrain and our methods for ResNet (RN) and VGG training.

NETWORKS(DATASET)	SGDM	PB	PB+LWP$^{v}_{D}$+SC$_{D}$	SPECTRAIN
VGG13 (C10)	**92.57**±0.15	**92.59**±0.15	**92.56**±0.14	**92.49**±0.12
RN20 (C10)	90.63±0.31	90.44±0.24	**90.92**±0.25	**90.93**±0.09
RN56 (C10)	92.39±0.20	91.89±0.40	92.48±0.11	**92.72**±0.10
RN50 (I1K)	75.7	75.1	**75.8**	75.3





Summary

Pipelined Backpropagation

- Avoid the Fill & Drain overhead of pipeline parallel training
- Causes inconsistent weights and delayed gradients
  - Loss of final accuracy, potential instability
- LWP+SC mitigates for these issues and outperforms existing methods in our setting

Pipelined Backpropagation at Scale:
Training Large Models without Batches

Atli Kosson, Vitaliy Chiley, Abhinav Venigalla, Joel Hestness, Urs Köster
Machine Learning Team