Reliable Machine Learning using Unreliable Components

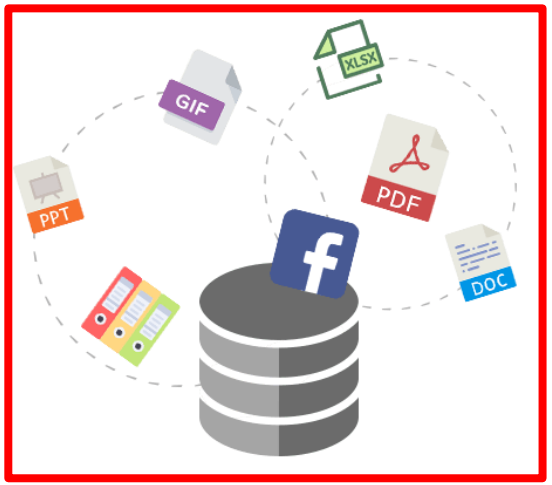
Sanghamitra Dutta

Carnegie Mellon University

Collaborators: Pulkit Grover, Gauri Joshi, Tze-Meng Low, Viveck Cadambe, Ziqian Bai, Haewon Jeong, Soumyadip Ghosh, Parijat Dube, Priya Nagpurkar

Motivation

Ever-increasing data and computing requirements



Neural Network with **60 million** parameters

[Krizhevsky et al. 2012]

Supercomputers with **10000+** nodes

Parallel and Distributed Processing

Reliability at Scale ?

"Supercomputing's monster in the closet" [Geist et al. 2016]

Key Issues affecting Reliability

- **Soft-Errors**: Random bit-flips and garbage outputs
- **Straggling Processors**: Few slow/faulty processors delay the entire computation

Question: How to compute reliably using ***unreliable*** components?

Solution: Use ***redundant*** computations in an efficient way

Brief History of the Field

Started with the seminal works of [Von Neumann, 1956]

Follow Up works:
[Pippenger, 1977] [Spielman, 1995] [Yang et al. 2015]

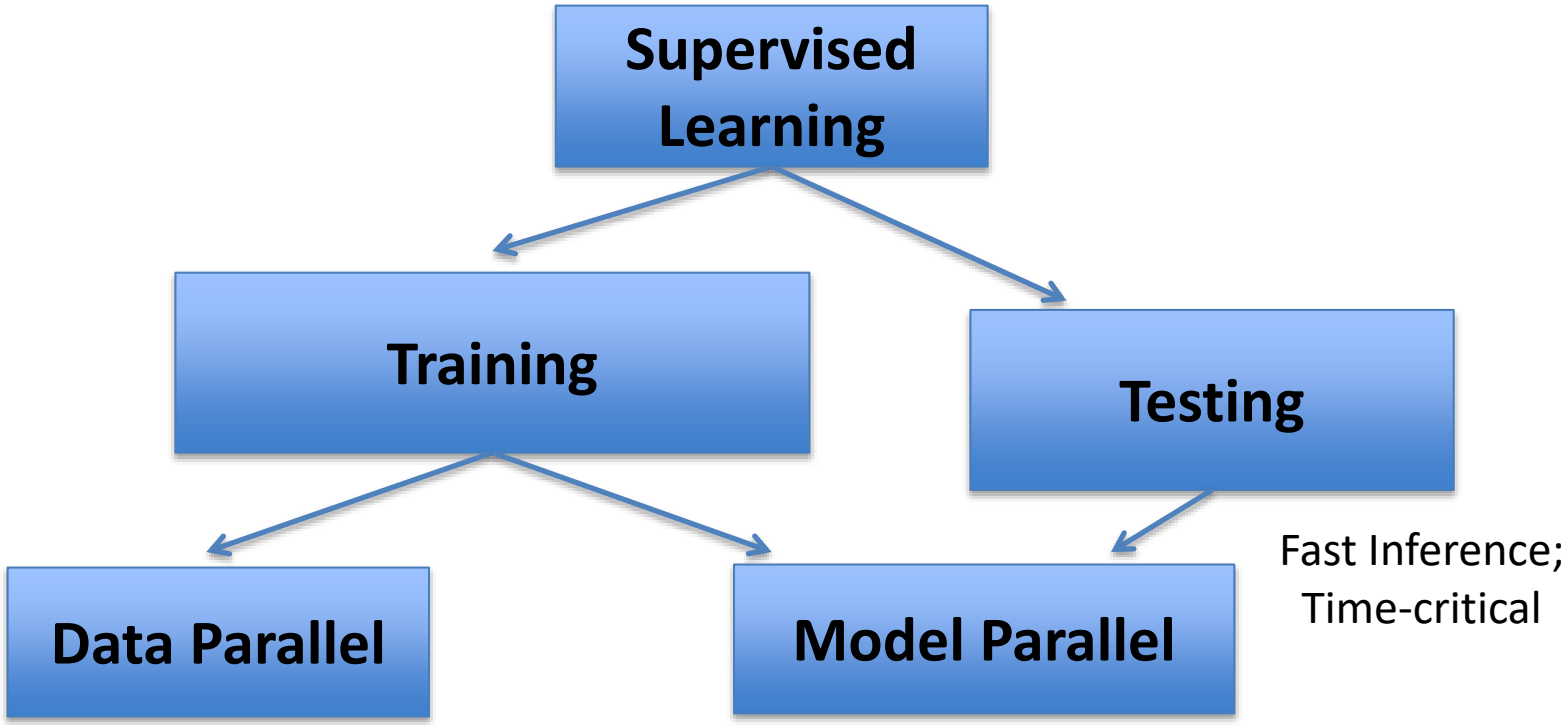
Algorithm Based Fault Tolerance [Huang & Abraham, 1984]

The Tail at Scale [Dean & Barroso, 2013]

Theoretical Analysis of Replication [Wang et al. 2015]

Asynchronous Methods
DistBelief [Dean et al. 2012]

Coded Computing
[Lee et al. 2015] [Dutta et al. 2016]



Straggler Problem

- Gradient Coding
  [Tandon et al. ICML 2015]
- Encoded Distributed Optimization
  [Karakus et al. NIPS 2017]
- Parallel Iterative Solver
  [Yang et al. NIPS 2017]
- **This talk: Redundancy Techniques**
  [Dutta et al. AISTATS 2018]

Critical Computation: matrix operations

- ABFT [Huang & Abraham, 1984]
- Matrix-vector [Lee et al. ISIT 2016]
  [Dutta et al. NIPS 2016]
- Matrix-matrix [Lee et al. ISIT 2017]
  [Yu et al. NIPS 2017]
  [Fahim et al. Allerton 2017]
- **This talk: CodeNet**
  [Dutta et al. ISIT 2018]

Outline

- Resilience in Model-parallel DNN Training
  - DNN Key Operations
  - CodeNet strategy
  - Theoretical Expected Time gains
  - Simulations on Amazon clusters
- Redundancy Techniques in Data-parallel Training
  - Synchronous and Asynchronous SGD variants
  - SGD Variants and their runtimes
  - Error-Runtime Trade-offs under straggling
  - Staleness Compensation using Asynchronous SGD

Outline

- **Resilience in Model-parallel DNN Training**
  - **DNN Key Operations**
  - CodeNet strategy
  - Theoretical Expected Time gains
  - Simulations on Amazon clusters
- Redundancy Techniques in Data-parallel Training
  - Synchronous and Asynchronous SGD variants
  - SGD Variants and their runtimes
  - Error-Runtime Trade-offs under straggling
  - Staleness Compensation using Asynchronous SGD

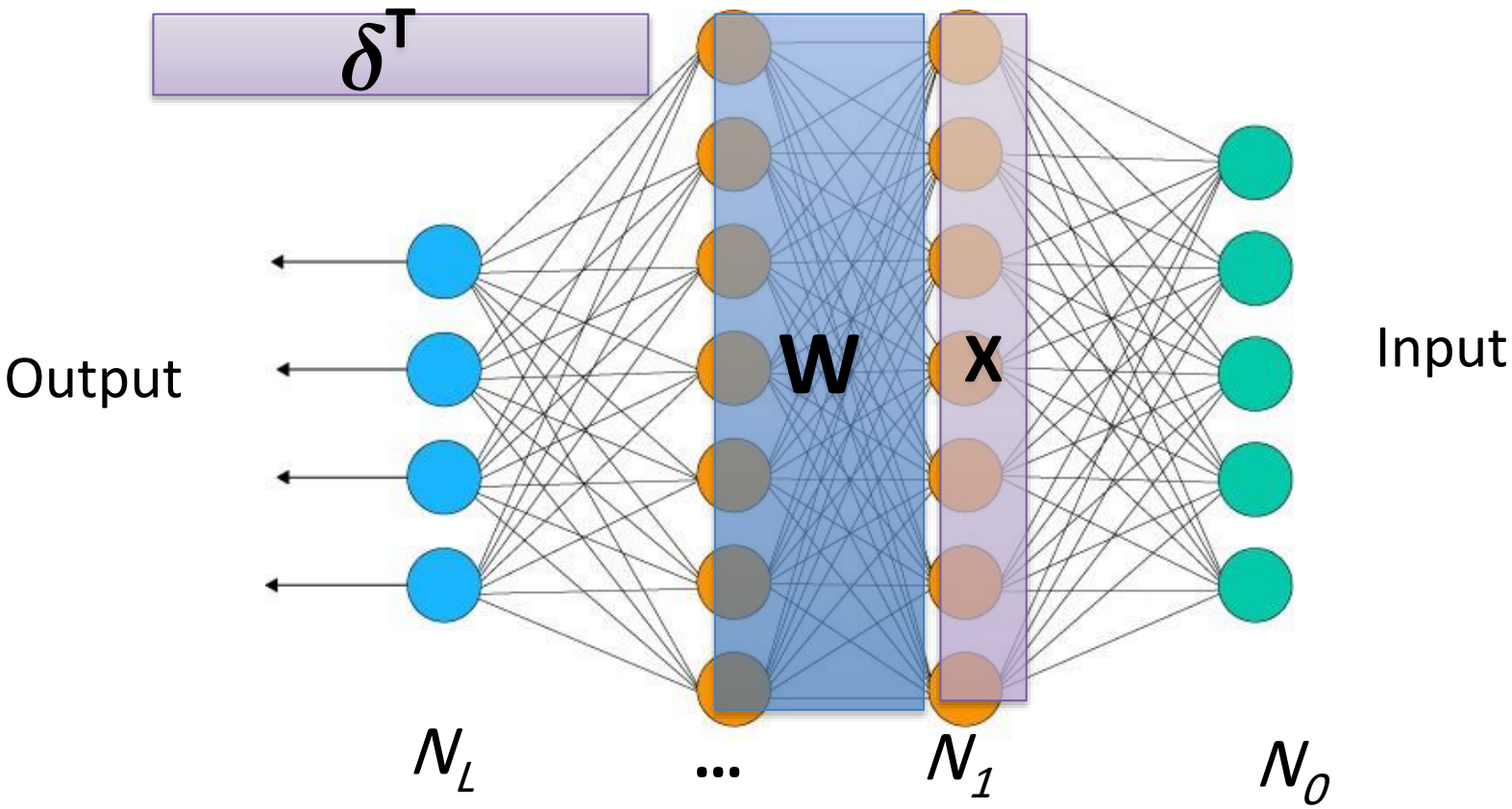
DNN Key Operations

Three Stages:

- Feedforward Stage
- Backpropagation Stage
- Update Stage

Matrix-vector products and Rank-1 updates have highest complexity.

DNN Key Operations



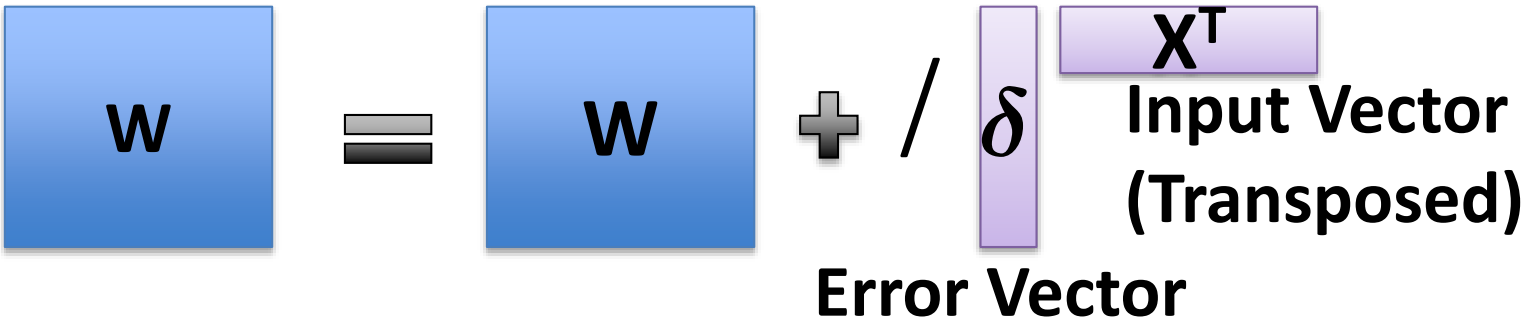
N_l : Number of Neurons in layer l

Input Layer Hidden Layer Output Layer

DNN Key Operations

Update Stage

Update each weight matrix as:



Matrix-vector products and Rank-1 updates are the primary steps.

Assumptions

- Storage Constraint: Matrix **W** is too large to be stored in one node.

  >> **Need to parallelize across nodes**

- Error Model: Any node can be affected by soft-errors and produce garbage outputs.

It is important to make the primary steps error-resilient!

In Addition:

- **Error resilience in the other steps**
- **Negligible communication & encoding/decoding overhead**

Goal

Given P base processors,

design an error-resilient parallelization strategy

using minimum redundant processors such that:

- Every node stores a fraction 1/P of the weight matrix $\boldsymbol{W}$ for each layer.
- Any node can be affected by soft-errors and produce garbage outputs.
- Negligible communication and encoding/decoding cost.
- Fully Decentralized (No master).

Outline

- Resilience in Model-parallel DNN Training
  - DNN Key Operations
  - **CodeNet strategy**
  - Theoretical Expected Time gains
  - Simulations on Amazon clusters
- Redundancy Techniques in Data-parallel Training
  - Synchronous and Asynchronous SGD variants
  - SGD Variants and their runtimes
  - Error-Runtime Trade-offs under straggling
  - Staleness Compensation using Asynchronous SGD

Basic Idea of Coded Computing

$$s = Wx$$

$$\begin{bmatrix} W_{0,:} \\ W_{1,:} \end{bmatrix}$$

Break W into Row Blocks

Cannot encode matrix W at every iteration!!

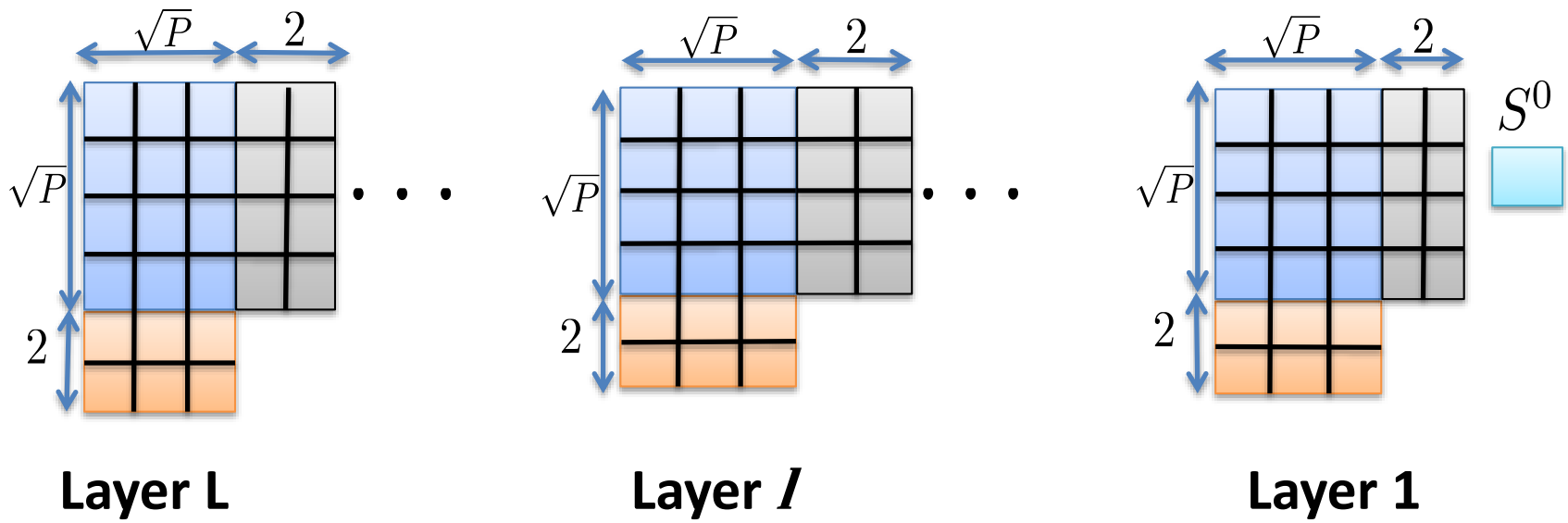
$$\begin{bmatrix} W_{0,:} \\ W_{1,:} \\ \widetilde{W}_{2,:} \\ \widetilde{W}_{3,:} \end{bmatrix} x = \begin{bmatrix} W_{0,:} \\ W_{1,:} \\ W_{2,:} + W_{3,:} \\ W_{0,:} - W_{1,:} \end{bmatrix} x$$

Very high complexity !!

Can recover output if any one is **erroneous** and you don't know which one

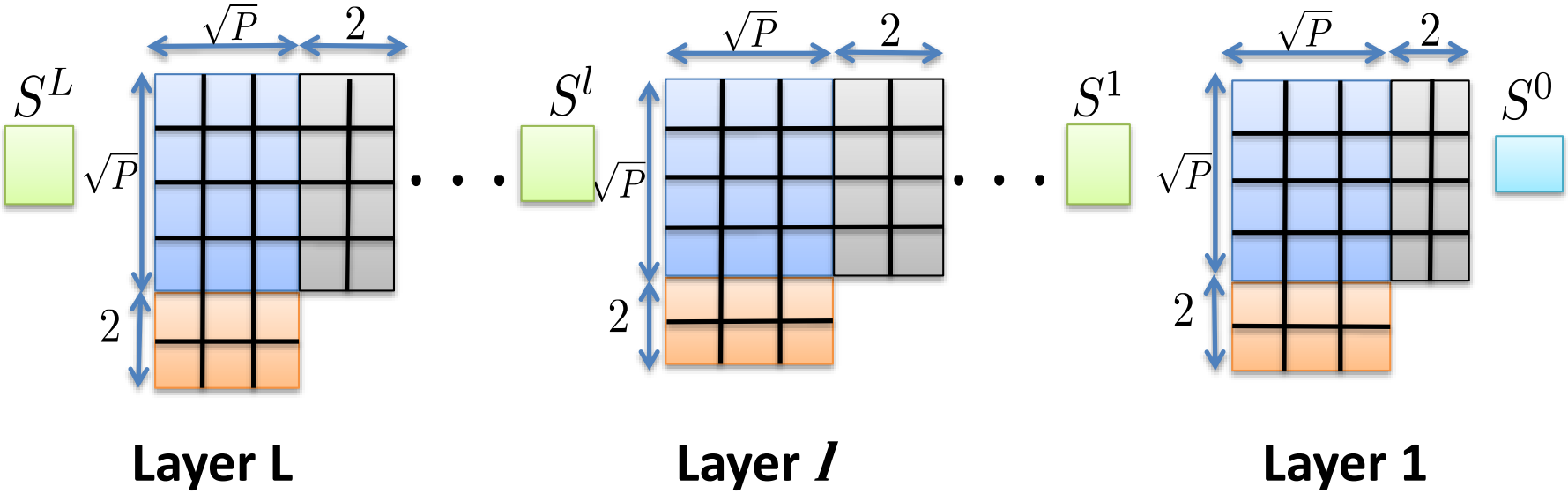
CodeNet Strategy

- Processor Layout



CodeNet Strategy

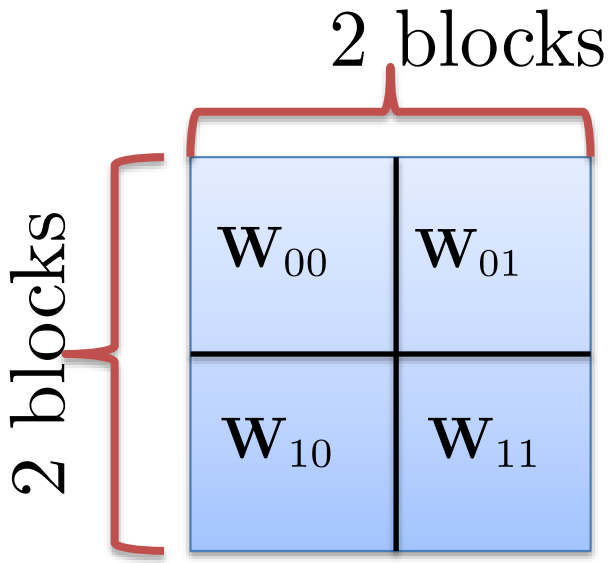
- Processor Layout



Error-free virtual nodes for ease of understanding

CodeNet Strategy

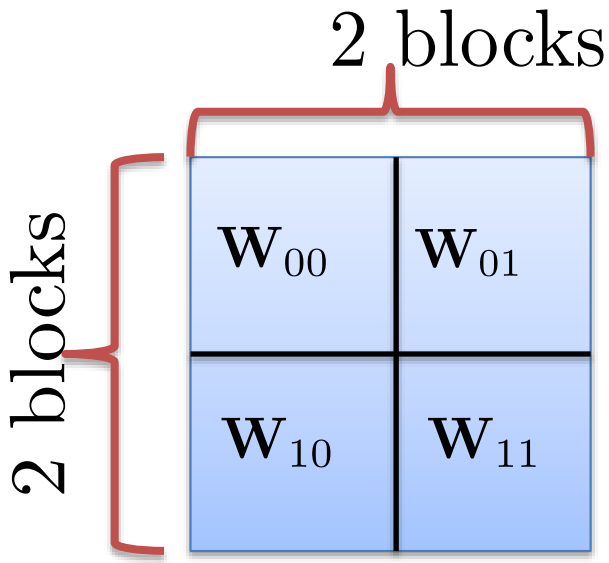
- Pre-processing – Initial encoding of matrix **W** only once prior to training



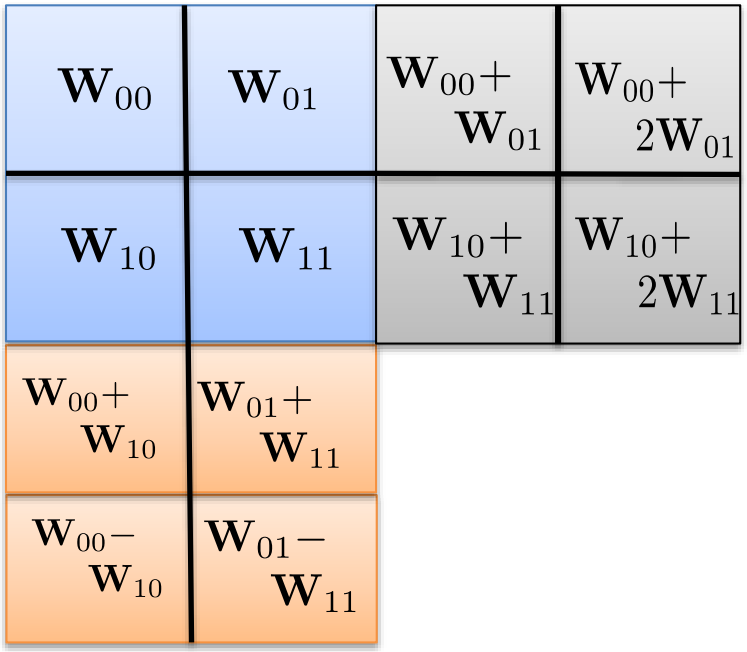
P=4

CodeNet Strategy

- Pre-processing – Initial encoding of matrix **W** only once prior to training

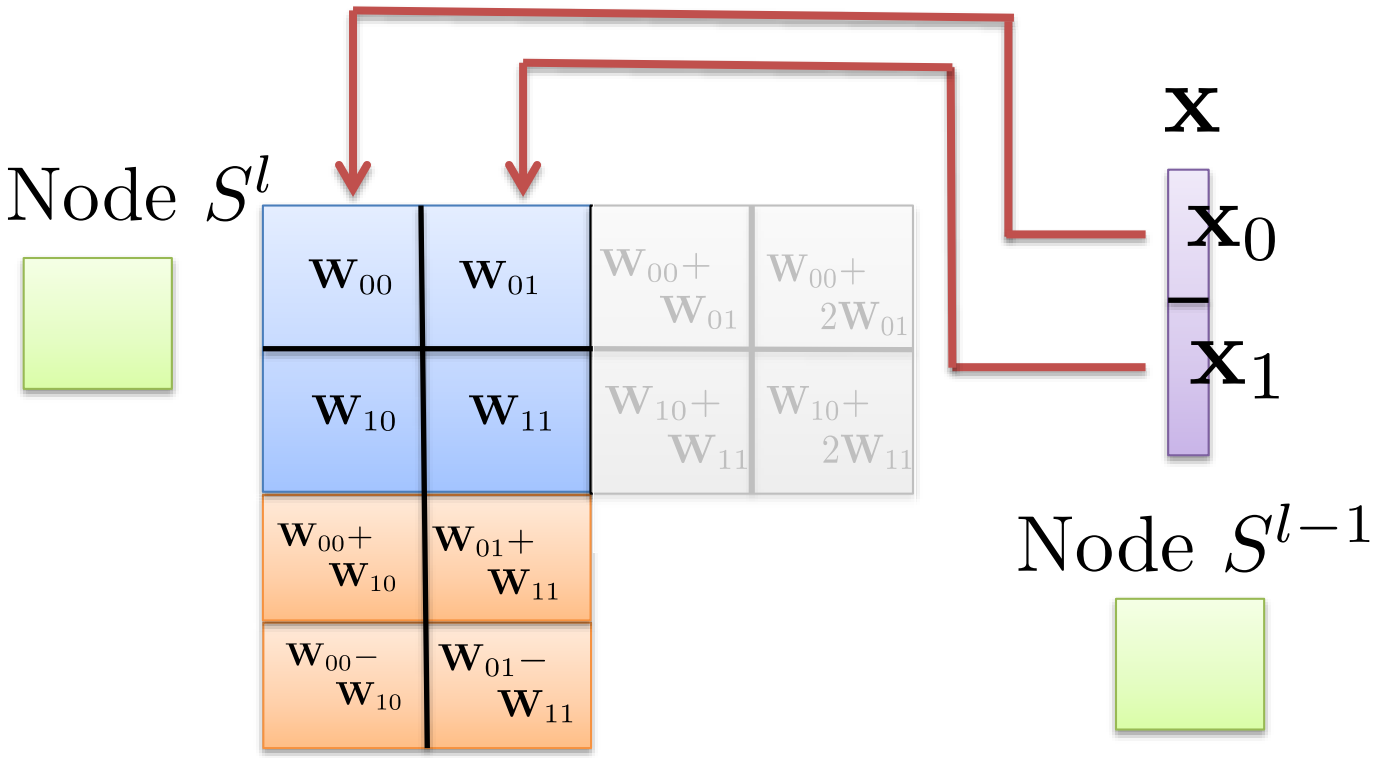


P=4



CodeNet Strategy

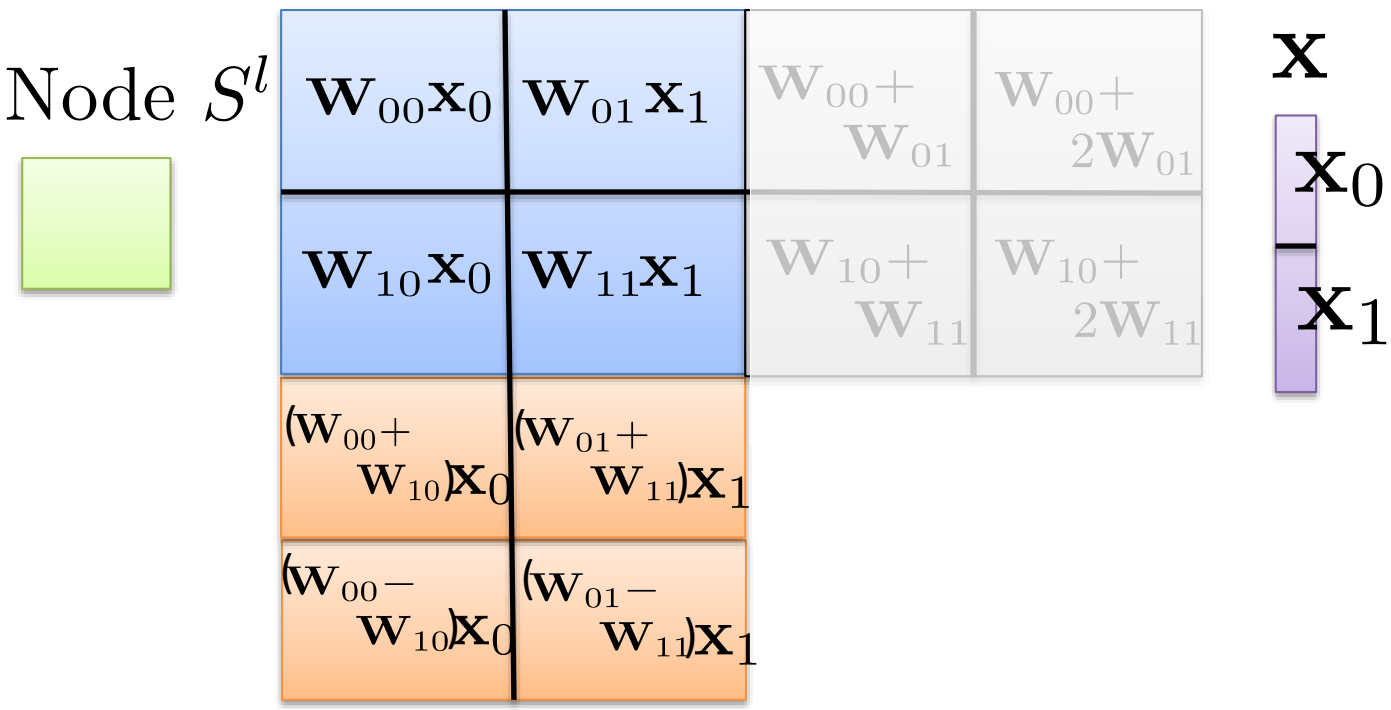
- Feedforward Stage Compute: $\mathbf{W}\mathbf{x}$



CodeNet Strategy

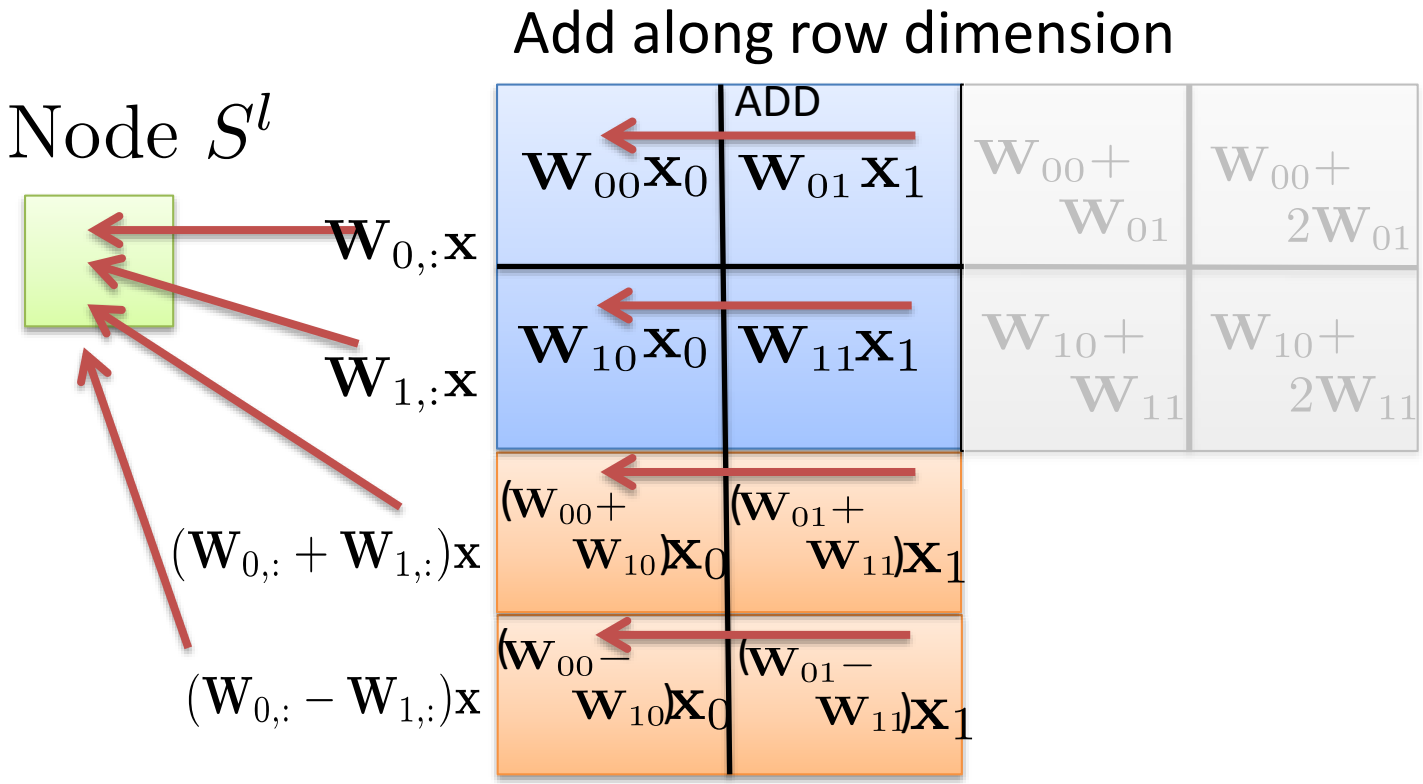
- Feedforward Stage $\text{Compute: } \mathbf{W}\mathbf{x}$

Compute individually



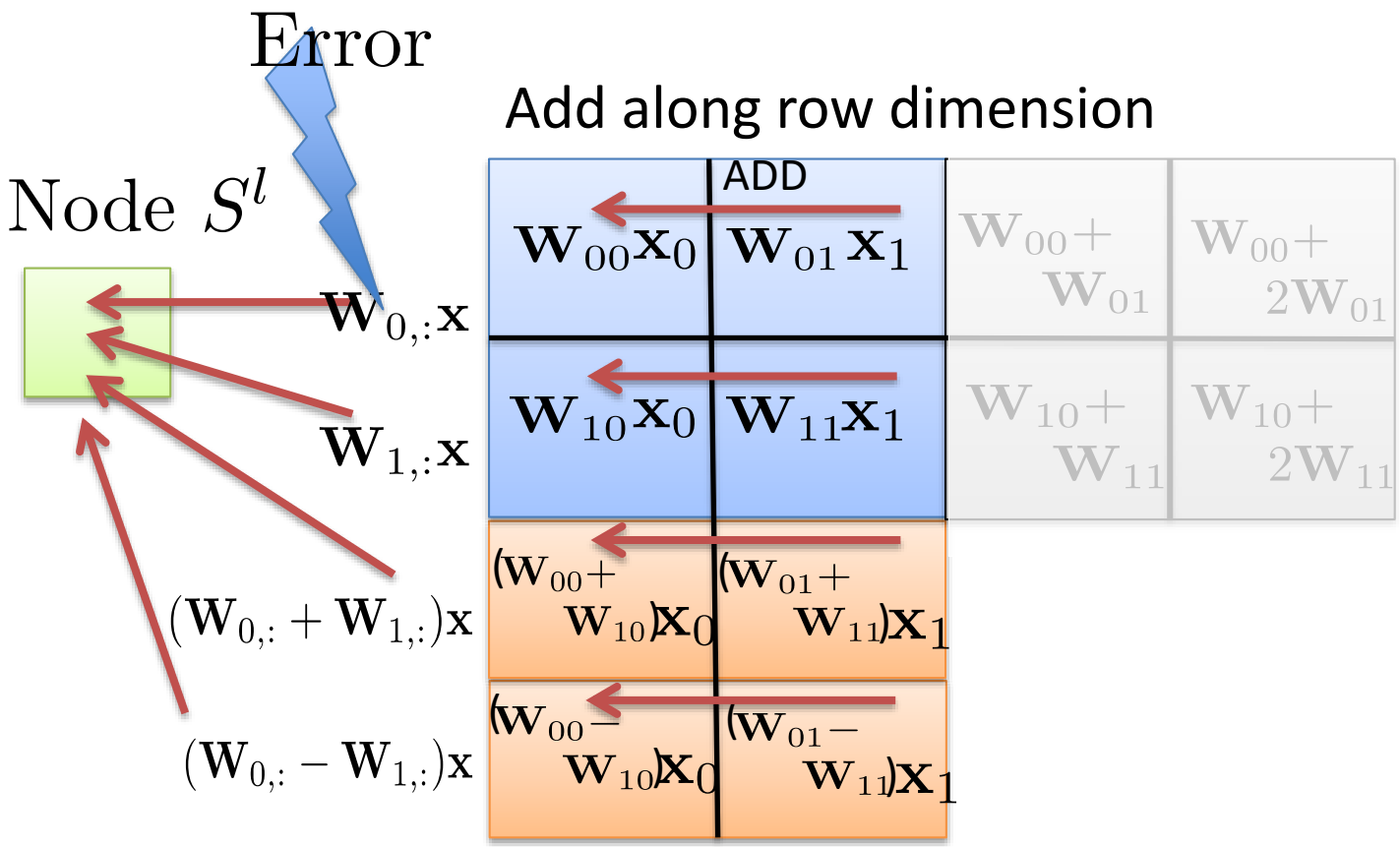
CodeNet Strategy

- Feedforward Stage $\text{Compute: } \mathbf{W}\mathbf{x}$



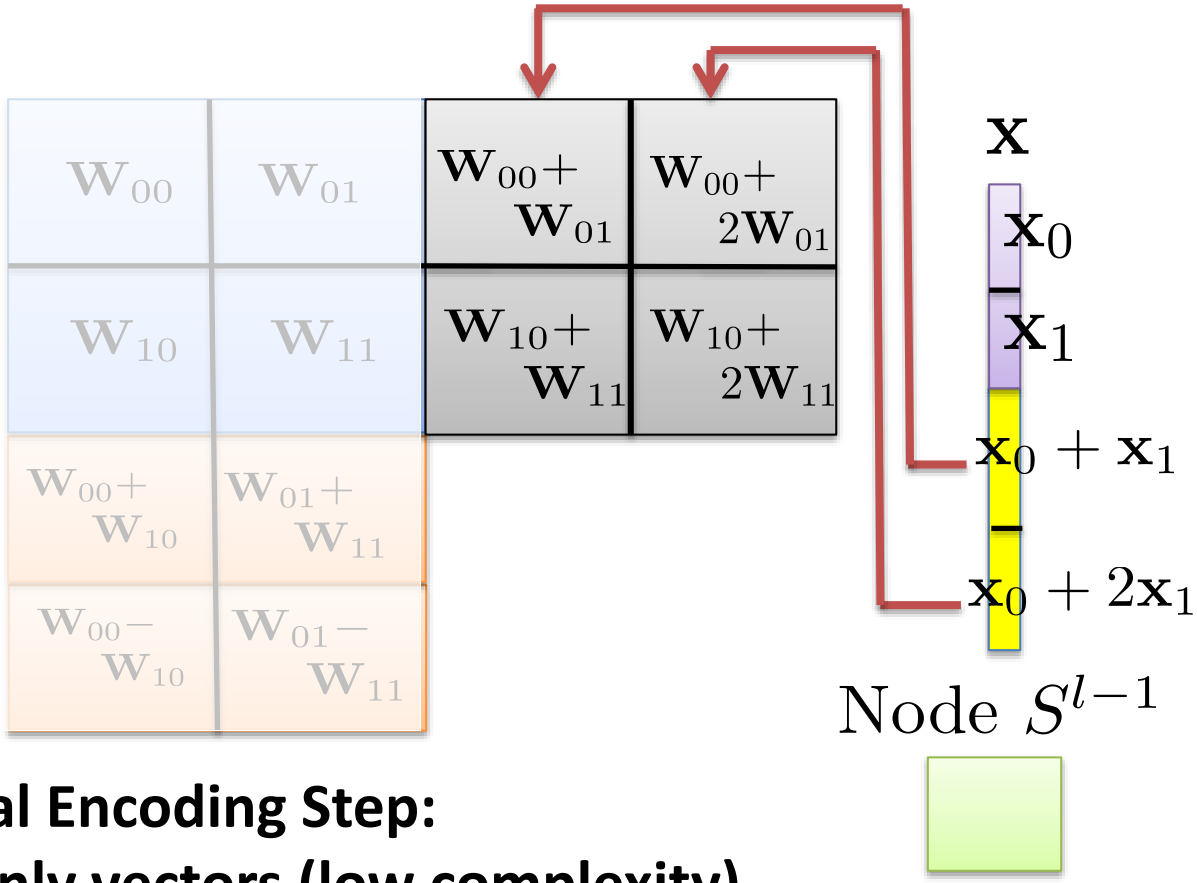
CodeNet Strategy

- Feedforward Stage Compute: $\mathbf{W}\mathbf{x}$



CodeNet Strategy

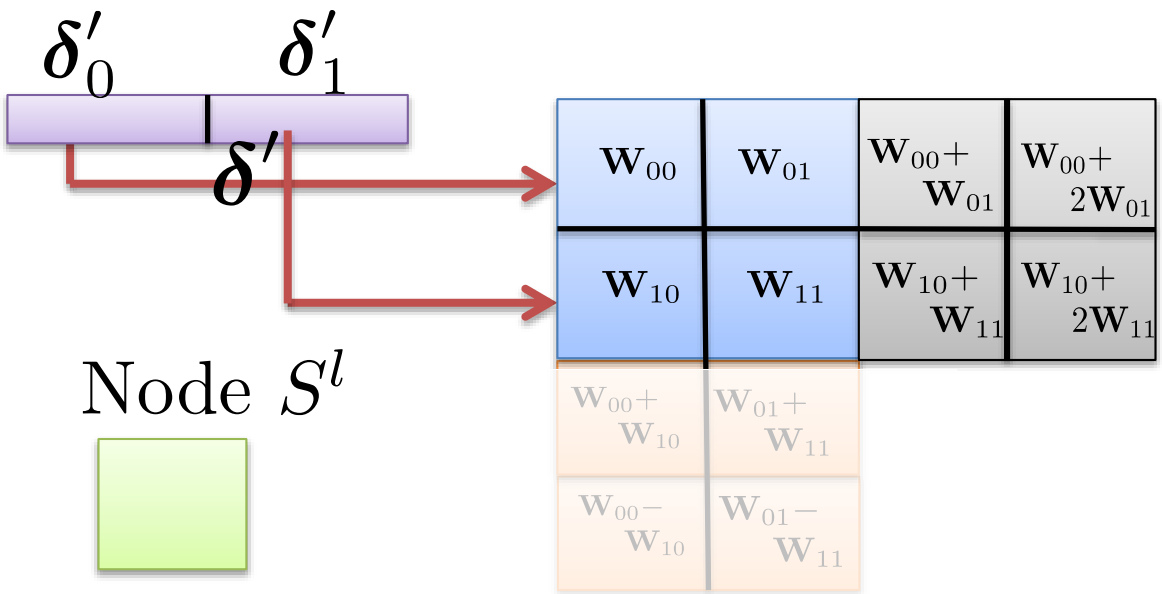
- Feedforward Stage Compute: $\mathbf{W}\mathbf{x}$



Additional Encoding Step:
Encode only vectors (low complexity)

CodeNet Strategy

- Backpropagation Stage Compute: $\boldsymbol{\delta}^T \mathbf{W}$



Steps are very similar to Feedforward Stage.

CodeNet Strategy

- Update Stage

$$\mathbf{W}_{00} + 2\mathbf{W}_{01} \leftarrow \mathbf{W}_{00} + 2\mathbf{W}_{01} + \mu\boldsymbol{\delta}_0(\mathbf{x}_0 + 2\mathbf{x}_1)'$$

$$\mathbf{W}_{00} \leftarrow \mathbf{W}_{00} + \mu\boldsymbol{\delta}_0\mathbf{x}_0$$

Every node is able to update itself with no additional complexity!!

Updated matrix is automatically encoded!

$$\mathbf{W}_{00} + \mathbf{W}_{10} \leftarrow \mathbf{W}_{00} + \mathbf{W}_{10} + \mu(\boldsymbol{\delta}_0 + \boldsymbol{\delta}_1)\mathbf{x}_0'$$

Main Results

- Worst Case

Thm: CodeNet can correct any t errors in primary steps at any layer using $P + 8t\sqrt{P}$ nodes.

- Probabilistic

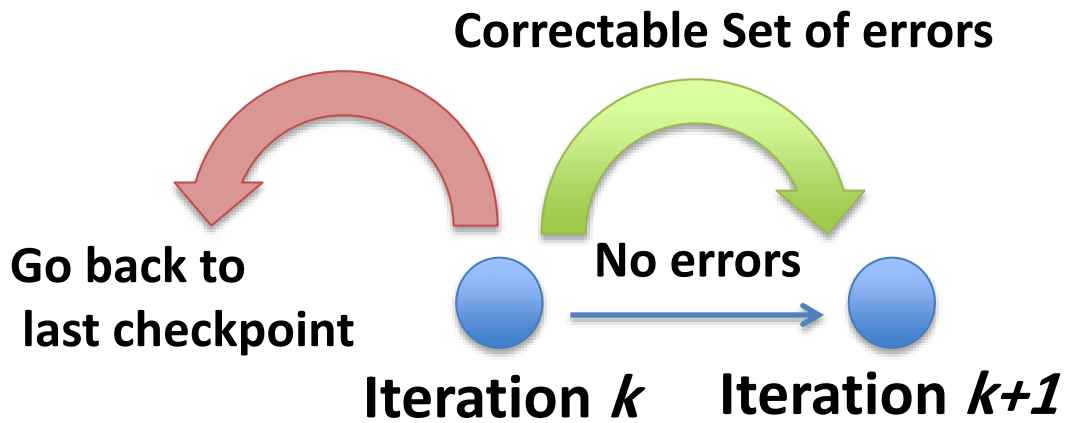
Thm: If errors are drawn from iid, continuous distributions, then CodeNet can detect errors in all steps with probability 1.

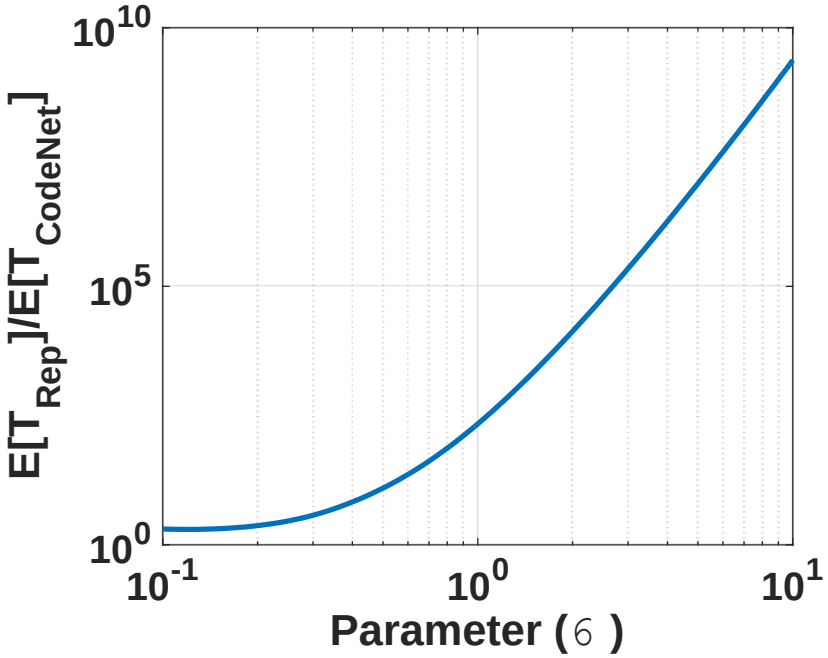
Outline

- Resilience in Model-parallel DNN Training
  - DNN Key Operations
  - CodeNet strategy
  - **Theoretical Expected Time gains**
  - Simulations on Amazon clusters
- Redundancy Techniques in Data-parallel Training
  - Synchronous and Asynchronous SGD variants
  - SGD Variants and their runtimes
  - Error-Runtime Trade-offs under straggling
  - Staleness Compensation using Asynchronous SGD

Theoretical Expected Time Gains

- Checkpoint periodically after I_0 iterations



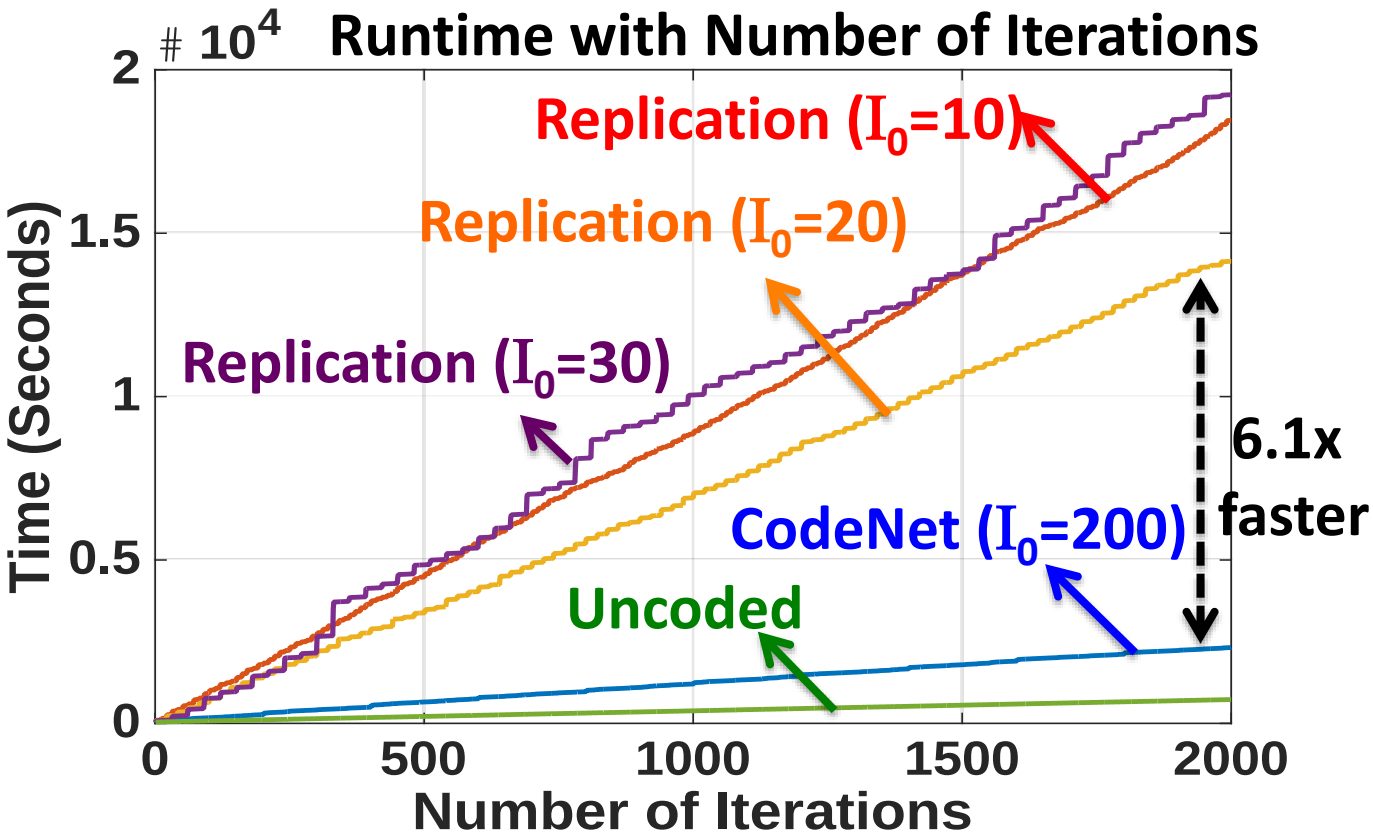


$$\text{No. of errors} \sim Poisson(\lambda)$$

Outline

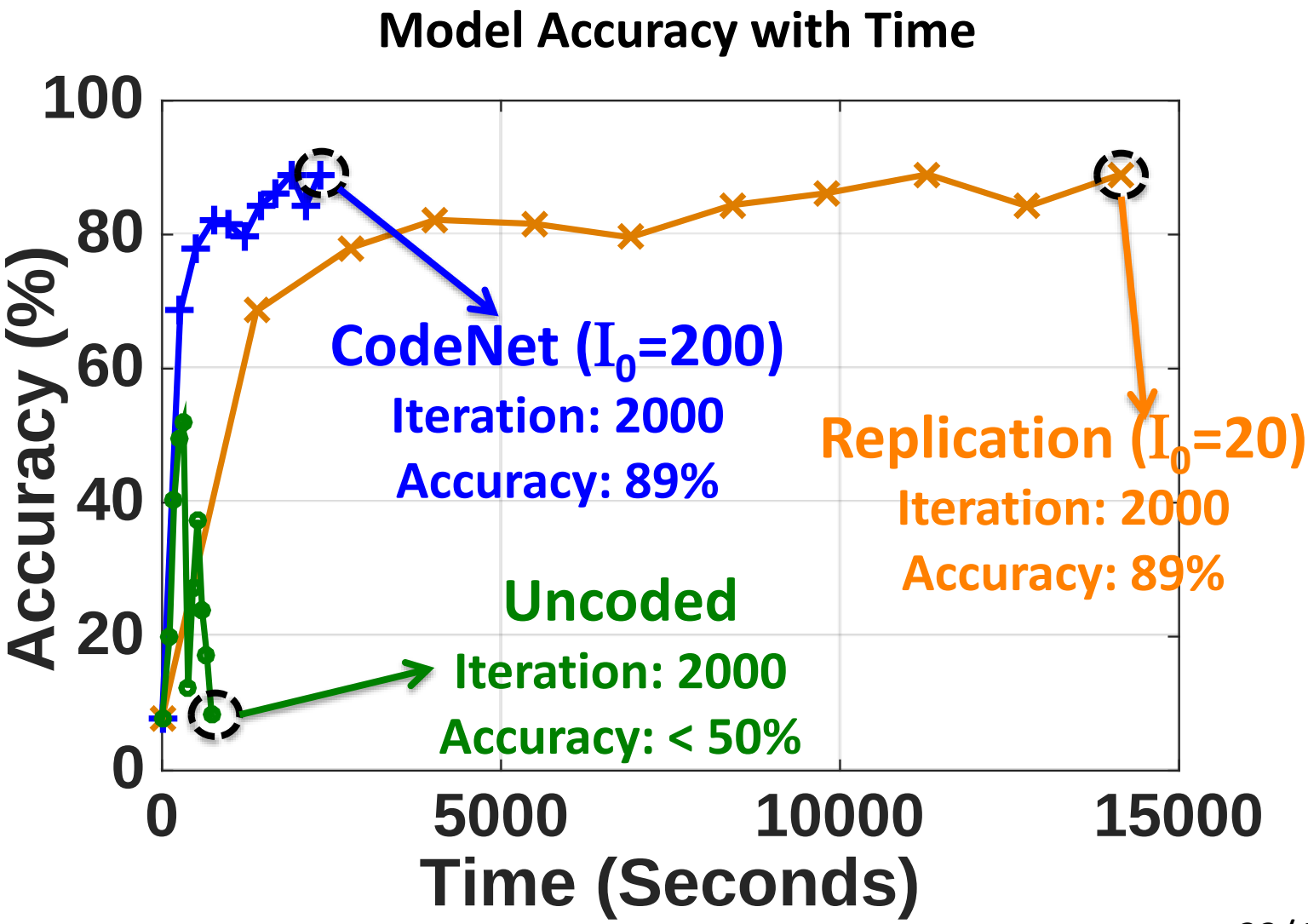
- Resilience in Model-parallel DNN Training
  - DNN Key Operations
  - CodeNet strategy
  - Theoretical Expected Time gains
  - **Simulations on Amazon clusters**
- Redundancy Techniques in Data-parallel Training
  - Synchronous and Asynchronous SGD variants
  - SGD Variants and their runtimes
  - Error-Runtime Trade-offs under straggling
  - Staleness Compensation using Asynchronous SGD

Simulation Results



MNIST dataset, Configuration [784 – 10^5 – 10^5 - 10],
Processors 40, Independently Fail with probability p=0.003,
For errors add a sparse matrix drawn from U[-500,500]

Simulation Results



Summary

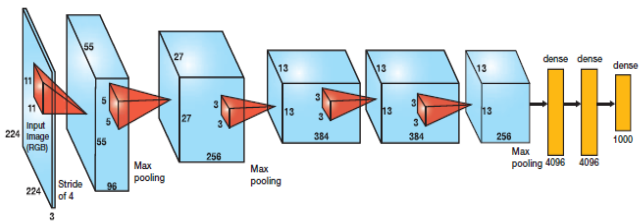
- Provide a novel strategy for error resilience in DNNs
- Negligible overheads
- Can detect errors in all steps
- Decentralized

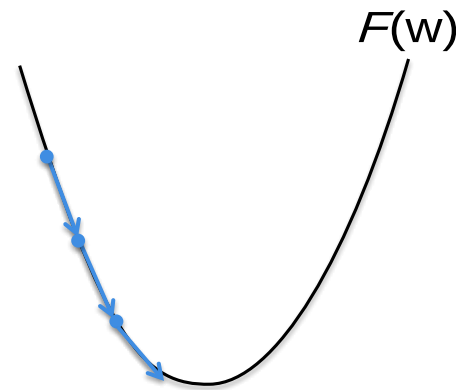
- Ongoing/Future Work: Better Coding techniques, Extension to CNNs etc.

Outline

- Resilience in Model-parallel DNN Training
  - DNN Key Operations
  - CodeNet strategy
  - Theoretical Expected Time gains
  - Simulations on Amazon clusters
- **Redundancy Techniques in Data-parallel Training**
  - **Synchronous and Asynchronous SGD variants**
  - SGD Variants and their runtimes
  - Error-Runtime Trade-offs under straggling
  - Staleness Compensation using Asynchronous SGD

Redundancy Techniques in Distributed SGD





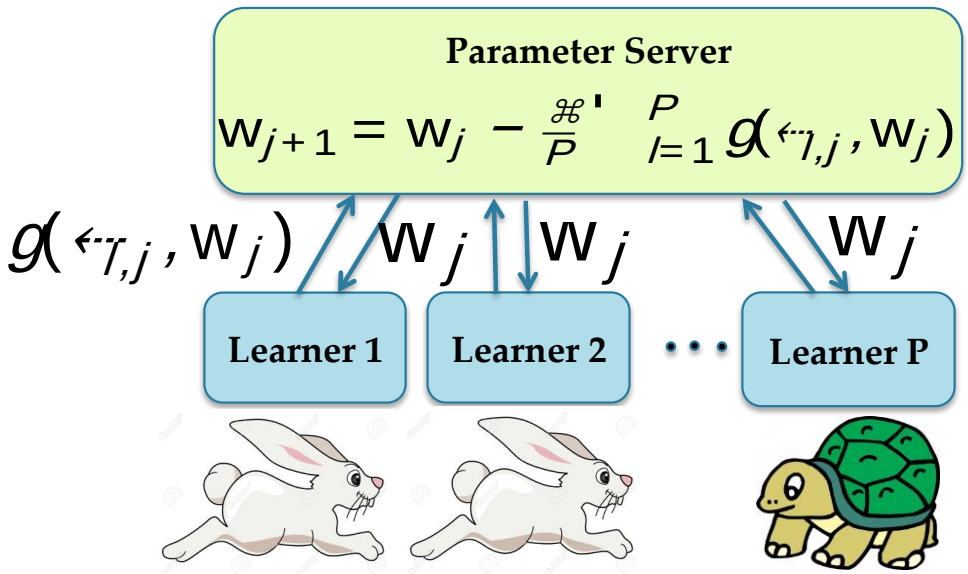
$$\mathbf{w}_{j+1} = \mathbf{w}_j - \eta \nabla F(\mathbf{w}_j)$$

Speeding Up SGD in data parallel setting is critical in several applications.

Key Issues:

- **Straggling Learners**
- **Gradient Staleness**

Distributed Synchronous SGD

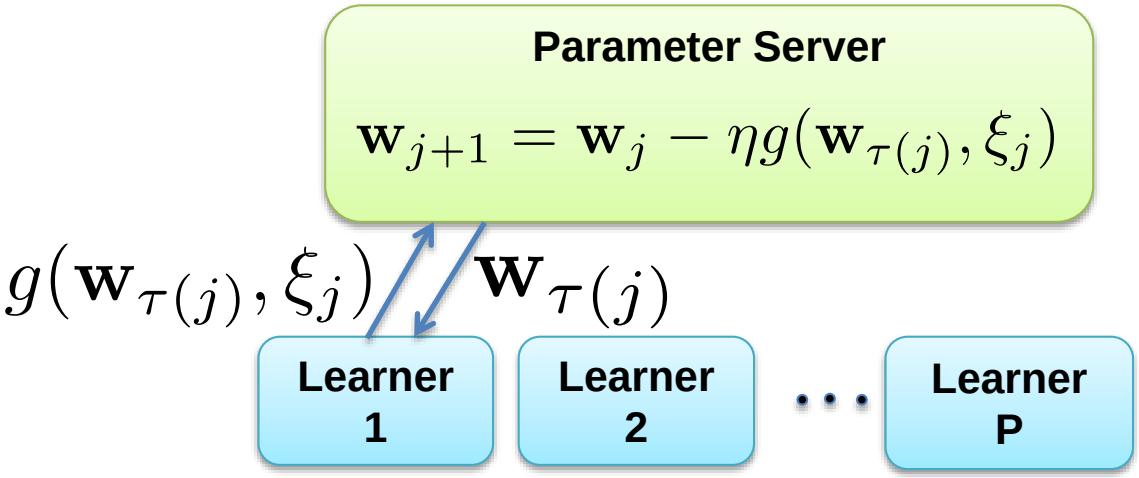


Wait for all learners!

Bottlenecked by the slowest

Stragglers affect the wall-clock computation time

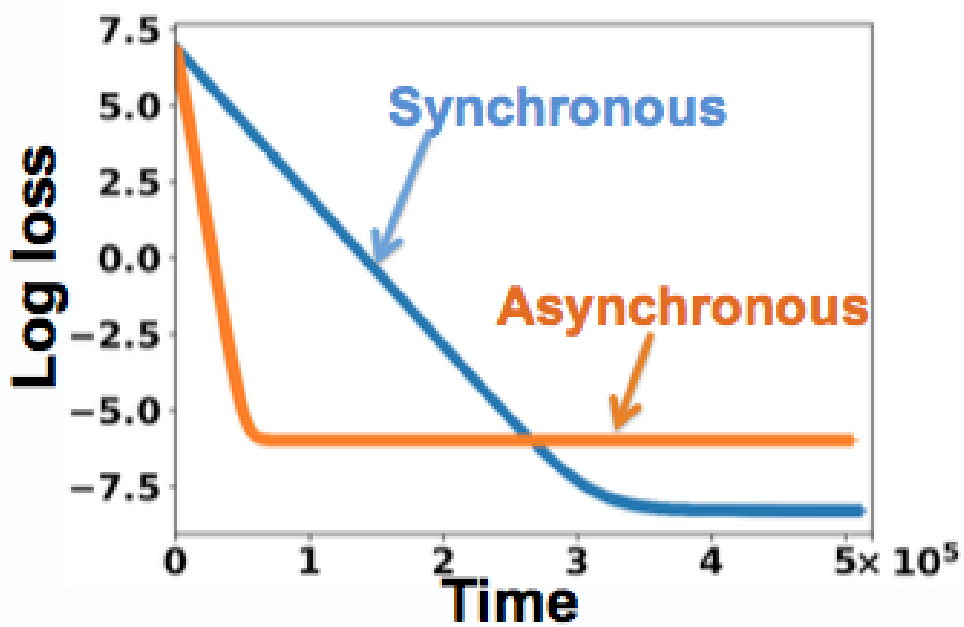
Distributed Asynchronous SGD



Any learner can update the PS irrespective of others.

But, stale gradients affect error convergence!

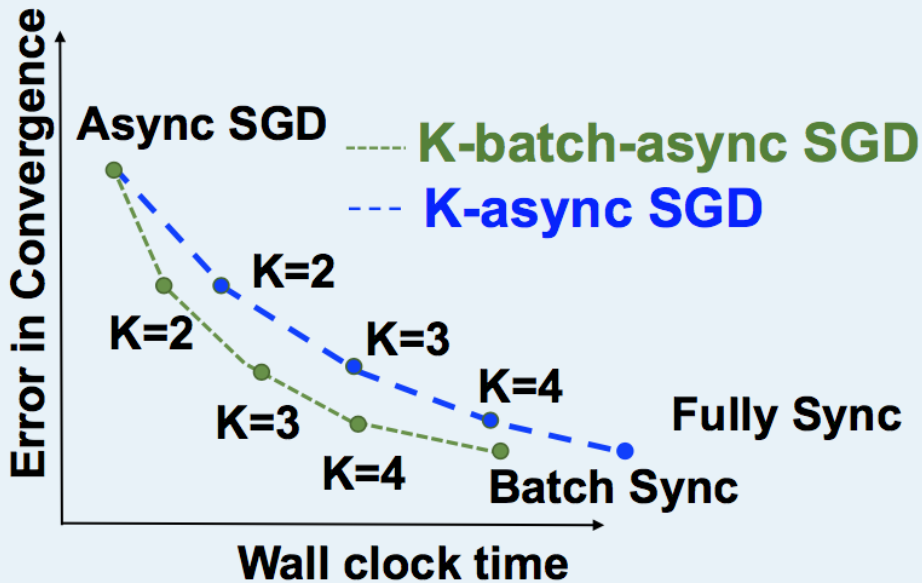
Main Takeaway



Need to understand convergence of error with wall-clock time instead of iterations or epochs!

Questions?

- Straggler mitigation using different SGD variants: How do the Error-Runtime trade-offs compare?



- Staleness compensation in Asynchronous SGD: Can we adapt the learning rate to improve convergence?

Model

ASSUMPTIONS:

- X_l : **Random Variable** denoting time taken by learner l for a mini-batch; iid across learners and mini-batches.
- Loss function $F(\boldsymbol{w})$: L-smooth, c-strongly convex.
- Standard unbiasedness and bounded var [Bottou et al. 2016] assumption on gradients evaluated over a mini-batch.

DEFINITIONS:

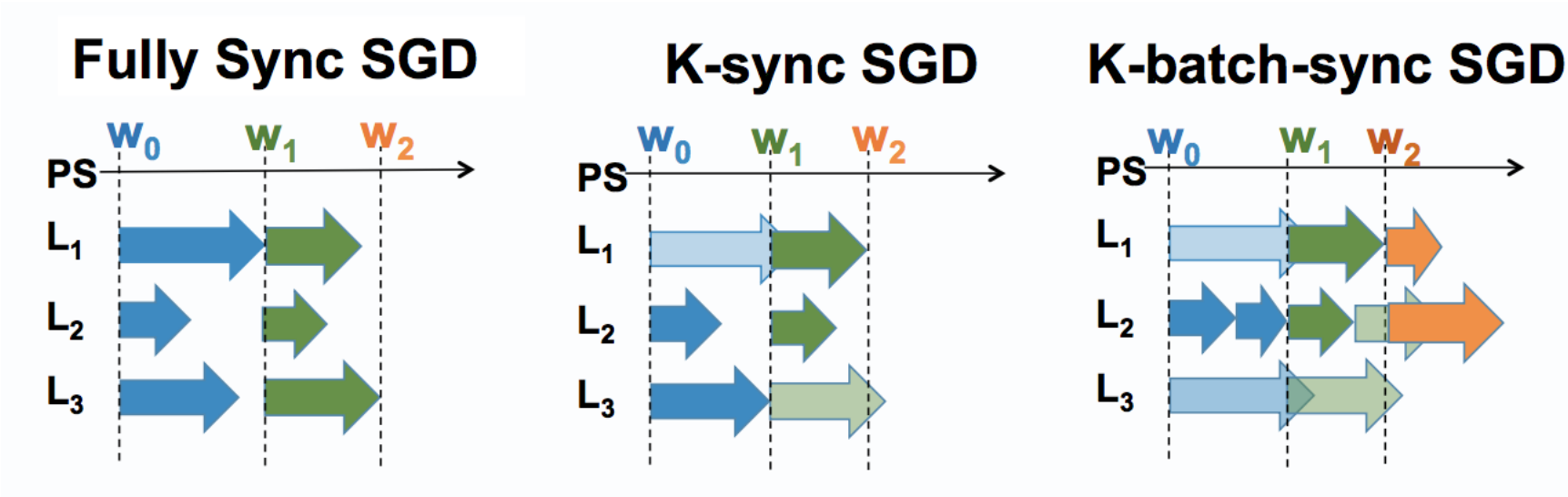
- **Error**: Expected Gap of risk function from its optimal value after J iterations, *i.e.*, $\mathbb{E}[F(\boldsymbol{w}_J) - F^*]$.
- **Runtime**: Expected Time to complete J iterations at the central PS.

Outline

- Resilience in Model-parallel DNN Training
  - DNN Key Operations
  - CodeNet strategy
  - Theoretical Expected Time gains
  - Simulations on Amazon clusters
- Redundancy Techniques in Data-parallel Training
  - Synchronous and Asynchronous SGD variants
  - **SGD Variants and their runtimes**
  - Error-Runtime Trade-offs under straggling
  - Staleness Compensation using Asynchronous SGD

SGD Variants and their runtimes

Sync variants



$$w_{j+1} = w_j - \frac{\eta}{K} \sum_{l=1}^{K} g(\xi_{l,j}, w_j)$$

[Gupta et al. ICDM 2016]

[Chen et al. 2016]

Runtime per iteration

K-sync SGD:

Expected Runtime per iteration: $\mathbb{E}[T] = \mathbb{E}[X_{K:P}]$

where $X_{K:P}$ is the K-th statistic of $X_1, X_2, \ldots X_P$.

Special Case: Exponential $X_l \sim \exp(\mu)$

$$\mathbb{E}[T] = \left(\frac{\log \frac{P}{P-K}}{\mu}\right)$$

K-batch-sync SGD:

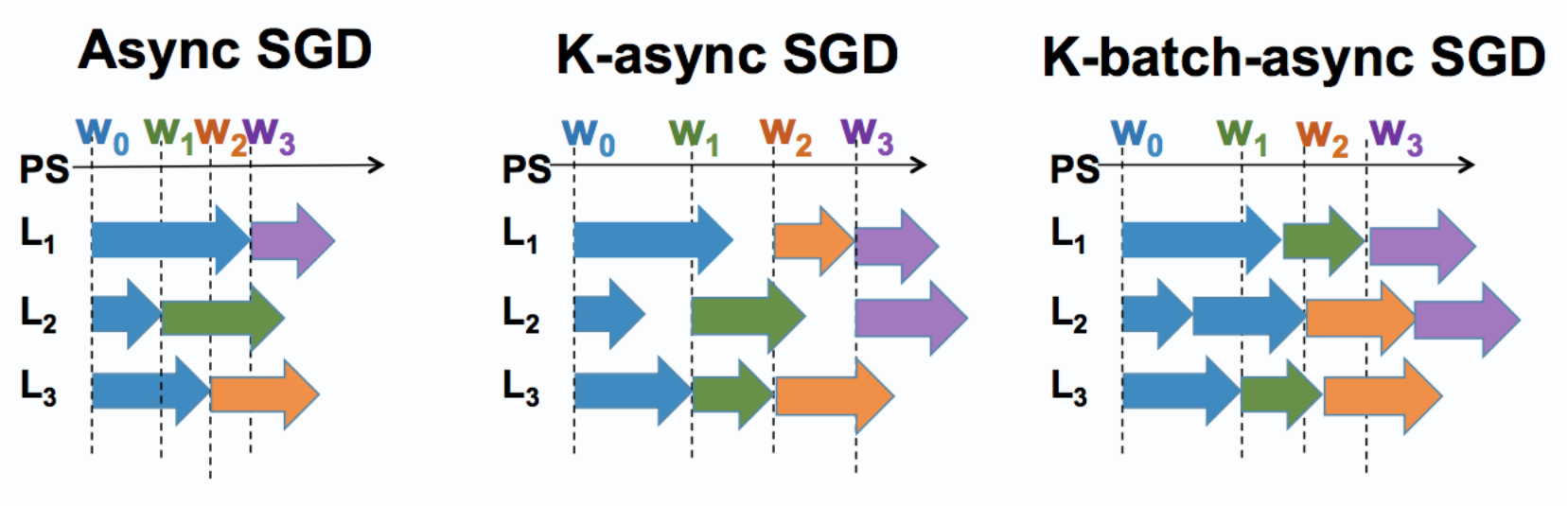
Expected Runtime per iteration: In general not tractable

Special Case: Exponential $X_l \sim \exp(\mu)$

$$\mathbb{E}[T] = \left(\frac{K}{P\mu}\right)$$

SGD Variants and their runtimes

Async variants



$$w_{j+1} = w_j - \frac{\eta}{K} \sum_{l=1}^{K} g(\xi_{l,j}, w_{\tau(l,j)})$$

[Lian et al. NIPS 2015]

Runtime per iteration

K-async SGD:

Expected Runtime per iteration: In general not tractable

Special Case: New-Longer-Than-Used X_l

$$\mathbb{E}[T] \leq \mathbb{E}[X_{K:P}]$$

Proof Sketch: $\Pr(X_l > u + t | X_l > t) \leq \Pr(X_l > u)$

Remaining time Y_l is stochastically dominated by X_l

For any weakly increasing function $h(.)$,

$$\mathbb{E}_{Y_l}[h(Y_l)] \leq \mathbb{E}_{X_l}[h(X_l)]$$

K-th statistic is weakly increasing.

Runtime per iteration

K-batch-async SGD:
Expected Runtime per iteration: $\mathbb{E}[T] = \frac{K}{P}\mathbb{E}[X]$

Proof Sketch: Elementary Renewal Theorem
$N_l(t)$: No. of pushes by learner l in time t
Avg. pushes/unit time $= \lim_{t\to\infty} \sum_{l=1}^{P} \frac{N_l(t)}{t} = \frac{P}{\mathbb{E}[X]}$
Avg. time/iteration = Time for K pushes $= \frac{K}{P}\mathbb{E}[X]$.

Outline

- Resilience in Model-parallel DNN Training
  - DNN Key Operations
  - CodeNet strategy
  - Theoretical Expected Time gains
  - Simulations on Amazon clusters
- Redundancy Techniques in Data-parallel Training
  - Synchronous and Asynchronous SGD variants
  - SGD Variants and their runtimes
  - **Error-Runtime Trade-offs under straggling**
  - Staleness Compensation using Asynchronous SGD

Obtaining Error-Runtime Trade-offs

Error after J iterations for sync variants

K-sync and K-batch-sync SGD:

$$\mathbb{E}(F(\boldsymbol{w}_J) - F^*) \leq \underbrace{\frac{\eta L \sigma^2}{2Kmc}}_{\text{Floor}} + \underbrace{(1-\eta c)^J}_{\text{Decay}} \left(F(\boldsymbol{w}_0) - F^* - \frac{\eta L \sigma^2}{2Kmc} \right)$$

[Bottou et al. 2016]

Obtaining Error-Runtime Trade-offs

Error after J iterations for Async variants

In general, difficult to analyze!

Asynchrony is like momentum [Mitliagkas et al. Allerton 2016]

Bounded delay [Recht et al. NIPS 2011] [Lian et al. NIPS 2015]

Our Assumption: For some γ in the interval $(0, 1)$,

$$\mathbb{E}[||\nabla F(\boldsymbol{w}_j) - \nabla F(\boldsymbol{w}_{\tau(l,j)})||_2^2] \leq \gamma \mathbb{E}[||\nabla F(\boldsymbol{w}_j)||_2^2]$$

Obtaining Error-Runtime Trade-offs

Error after J iterations for Async variants

K-async and K-batch-async SGD:

$$\mathbb{E}(F(\boldsymbol{w}_J) - F^*) \leq$$

$$\underbrace{\frac{\eta L \sigma^2}{2\gamma' K m c}}_{\text{Floor}} + \underbrace{(1 - \eta c \gamma')^J}_{\text{Decay}} \left(F(\boldsymbol{w}_0) - F^* - \frac{\eta \rho \sigma^2}{2\gamma' K m c} \right)$$

where $\gamma' = 1 - \gamma + \frac{p_0}{2}$ and $p_0 \leq$ conditional probability of 0 staleness given past delays and models.

Error-Runtime Trade-offs

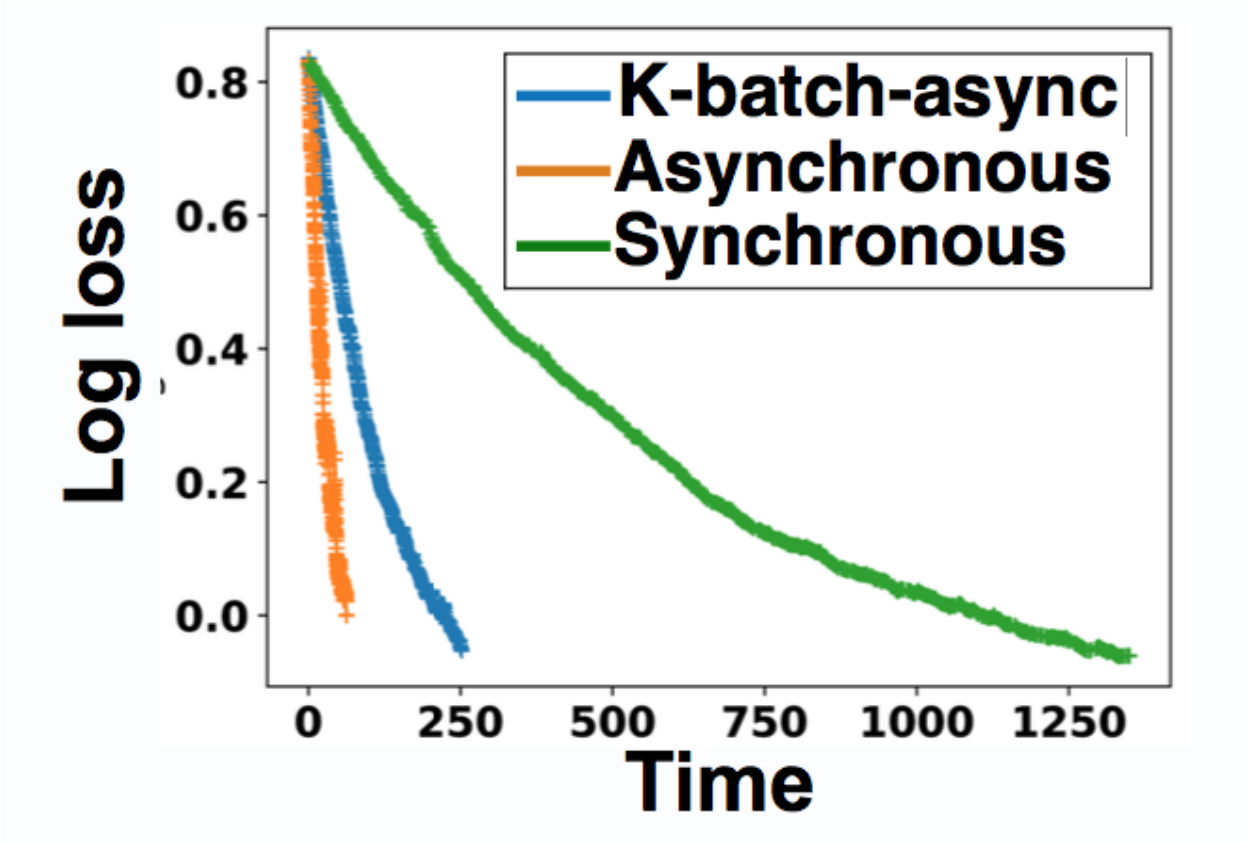


Figure: Error-Runtime trade-off on MNIST dataset: Parameters $X_i \sim \exp(1), P = 8, K = 4, m = 1$ and $\eta = 0.01$.

Outline

- Resilience in Model-parallel DNN Training
  - DNN Key Operations
  - CodeNet strategy
  - Theoretical Expected Time gains
  - Simulations on Amazon clusters
- Redundancy Techniques in Data-parallel Training
  - Synchronous and Asynchronous SGD variants
  - SGD Variants and their runtimes
  - Error-Runtime Trade-offs under straggling
  - **Staleness Compensation using Asynchronous SGD**

Staleness Compensation for Async SGD

Proposed Learning Rate: $\eta_j \leq \min\{\frac{C}{||w_{\tau(j)} - w_j||_2^2}, \eta_{max}\}$

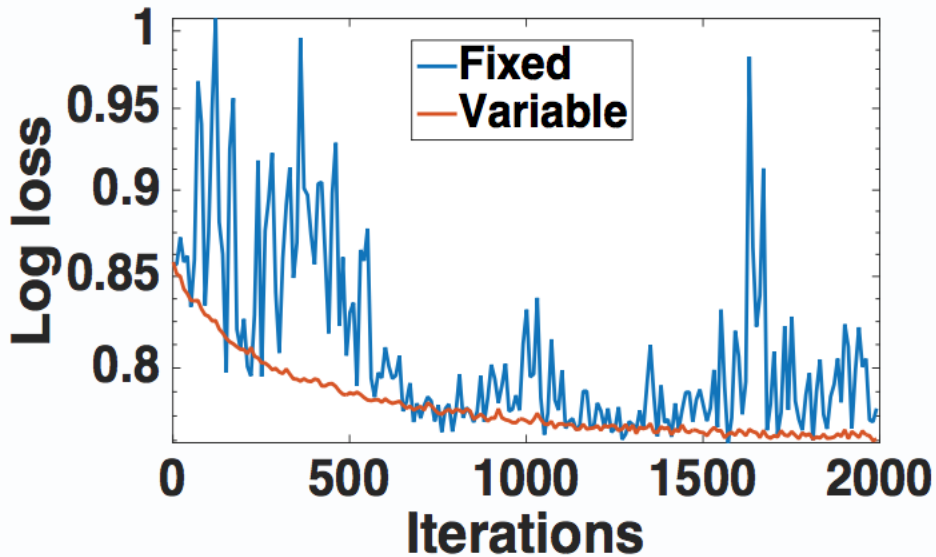
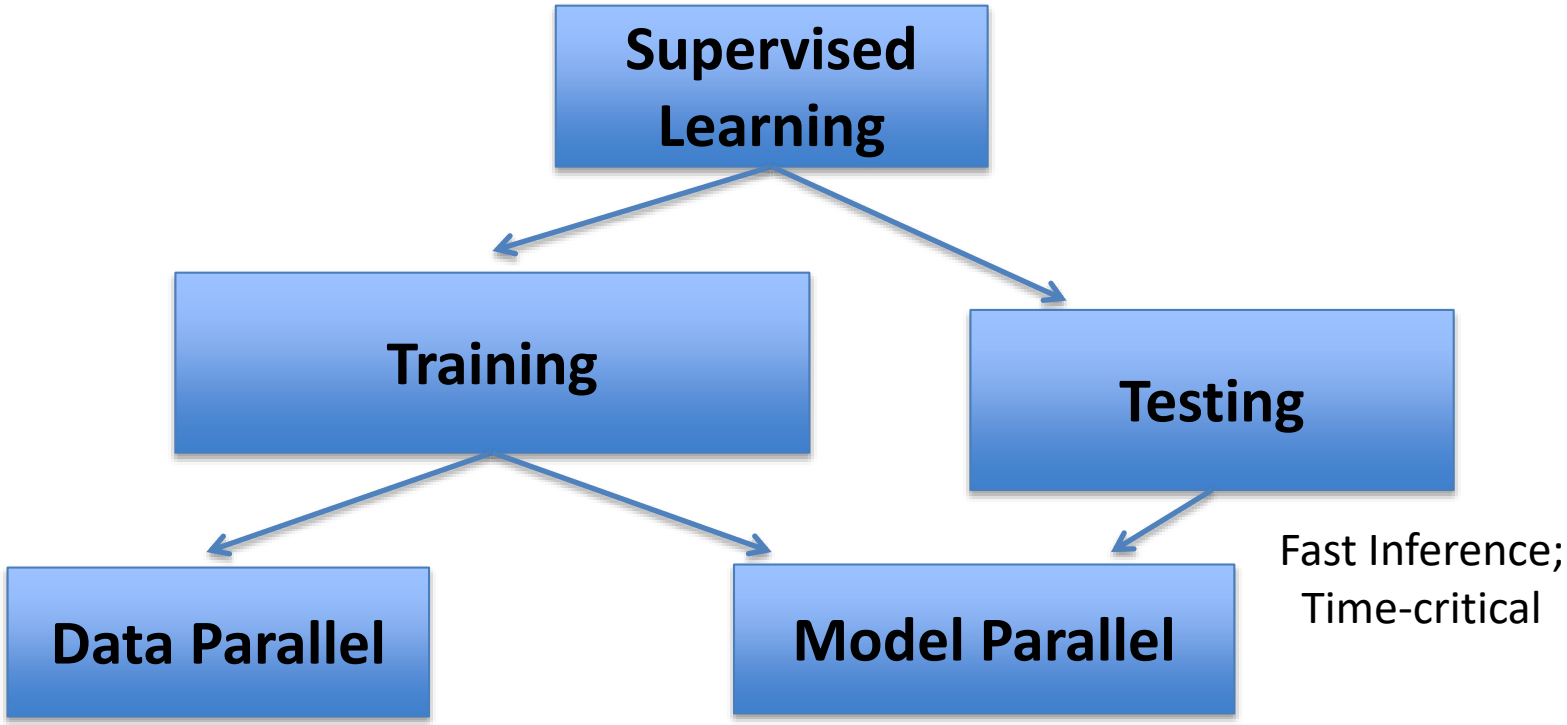


Figure: Error of Async SGD on CIFAR dataset: Parameters $X \sim \exp(20)$, $m = 250$ and $P = 40$. We compare fixed $\eta = 0.01$, and the variable schedule for $\eta_{max} = 0.01$ and $C = 0.005\eta_{max}$.



Straggler Problem

Critical Computation: matrix operations

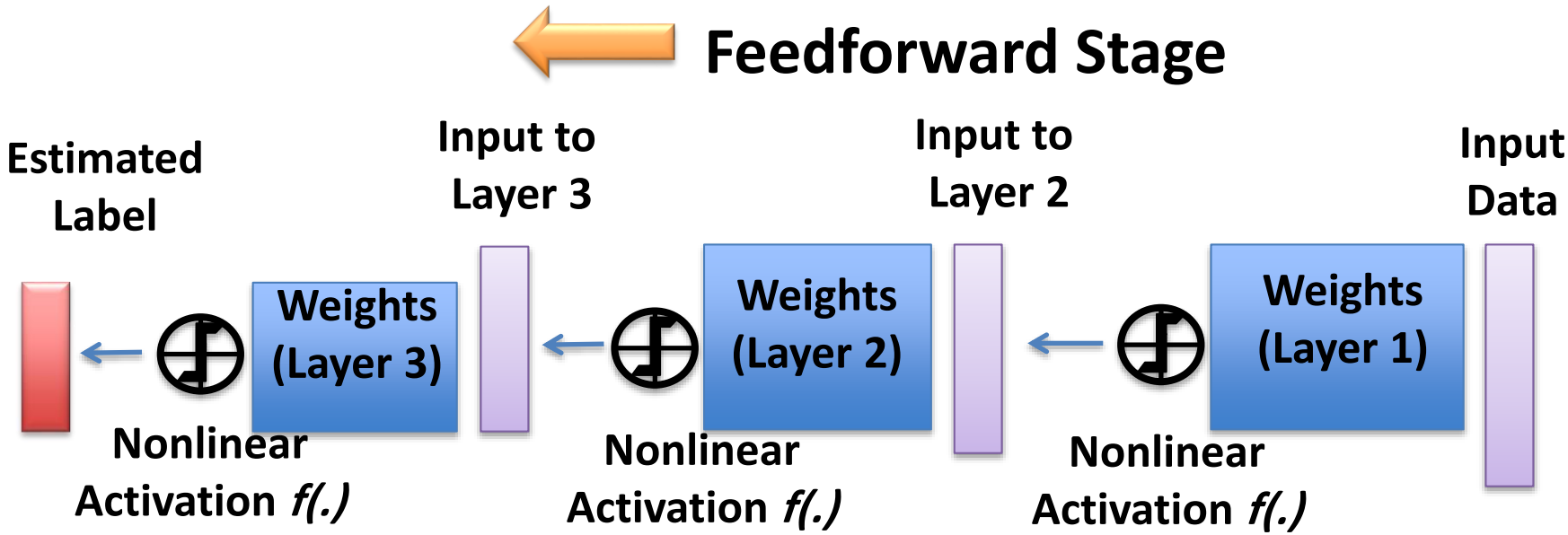
Redundancy Techniques and novel analysis of error-runtime trade-offs

Reliability in training DNNs using unreliable components.

Thank You

Extra Slides for understanding

DNN Key Operations



- Matrix-vector products: Complexity $O(N_l N_{l-1})$
- All other steps: Complexity $O(N_{l-1})$ or $O(N_l)$

DNN Key Operations

Backpropagated Error Calculation at last layer

and

used to compute

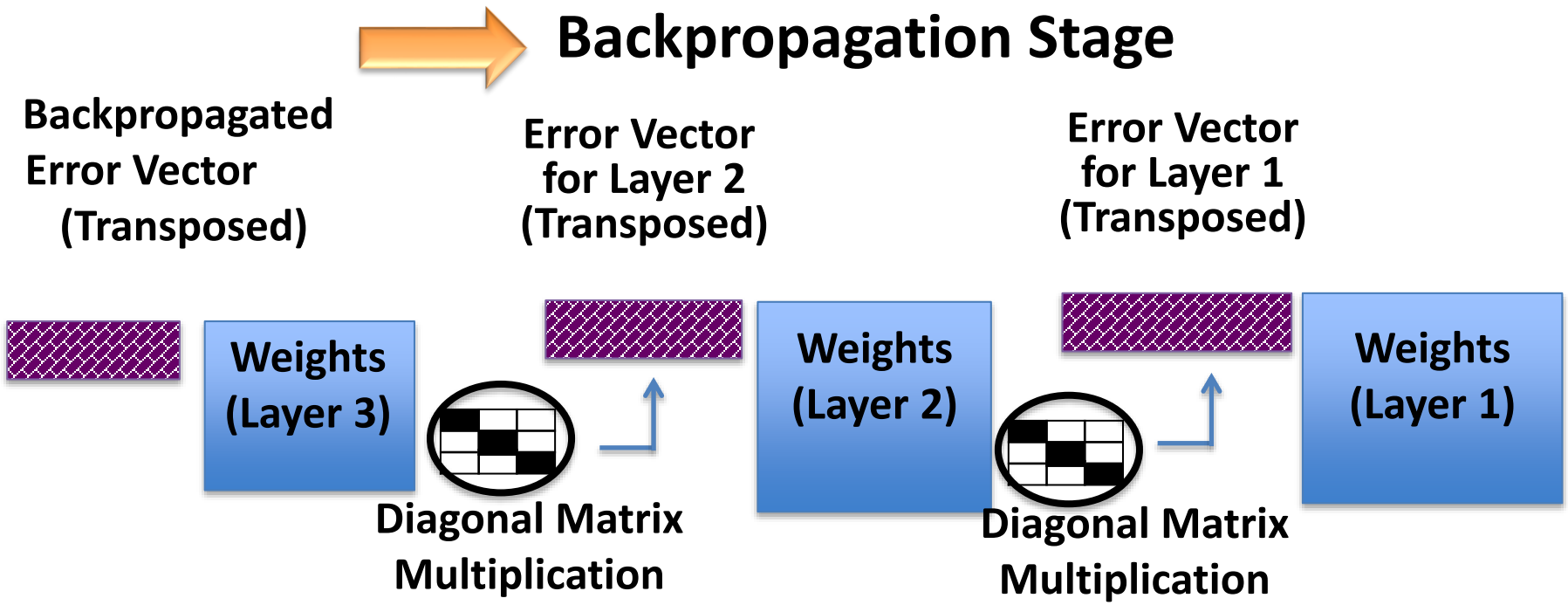
Estimated Label

True Label

Backpropagated Error Vector

Complexity $O(N_L)$

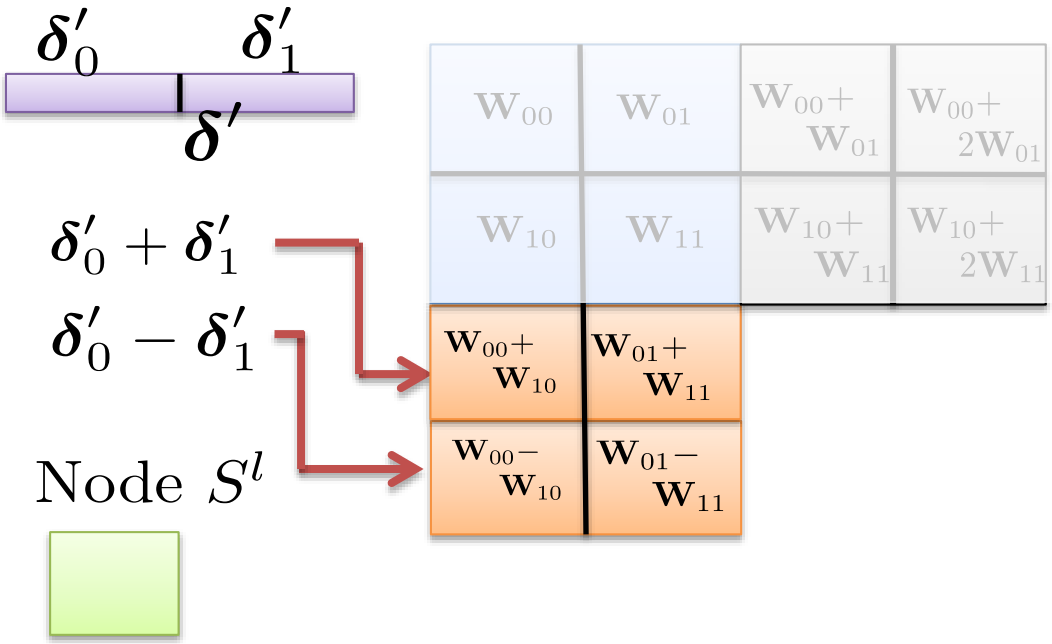
DNN Key Operations



- Matrix-vector products: Complexity $O(N_l N_{l-1})$
- All other steps: Complexity $O(N_{l-1})$ or $O(N_l)$

CodeNet Strategy

- Backpropagation Stage Compute: $\boldsymbol{\delta}^T \mathbf{W}$



Additional Encoding Step:
Encode only vectors (low complexity)

Background

Empirical Risk Function: $F(w) \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} f(\xi_i, w)$

- Batch Gradient Descent:

$$w_{j+1} = w_j - \underbrace{\frac{\eta}{N}\sum_{i=1}^{N} \nabla f(\xi_i, w_j)}_{\text{Too Expensive}}$$

- Stochastic Gradient Descent (SGD):

$$w_{j+1} = w_j - \underbrace{\eta \nabla f(\xi_j, w_j)}_{\text{Too Noisy}}$$

- Popularly use Mini-batch SGD:

$$w_{j+1} = w_j - \underbrace{\frac{\eta}{m}\sum_{\xi_i \in \xi_j, |\xi_j| = m} \nabla f(\xi_i, w_j)}_{\text{Noisy but less expensive}}$$