Distributed Training for Large-Scale Machine Learning Problems

Jyotikrishna Dass

dass dot jyotikrishna at tamu dot edu

http://people.tamu.edu/~jyoti1991/

Advisor: Prof. Rabi Mahapatra

COMPUTER SCIENCE
& ENGINEERING
TEXAS A&M UNIVERSITY

Table of Contents

Introduction

Machine Learning is Everywhere

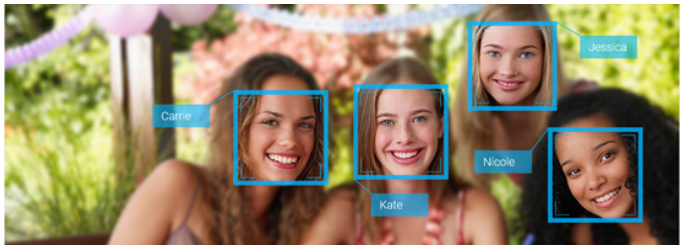


Figure: Face Recognition

Figure: Machine Translation

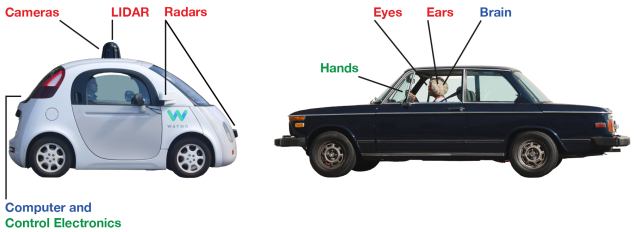
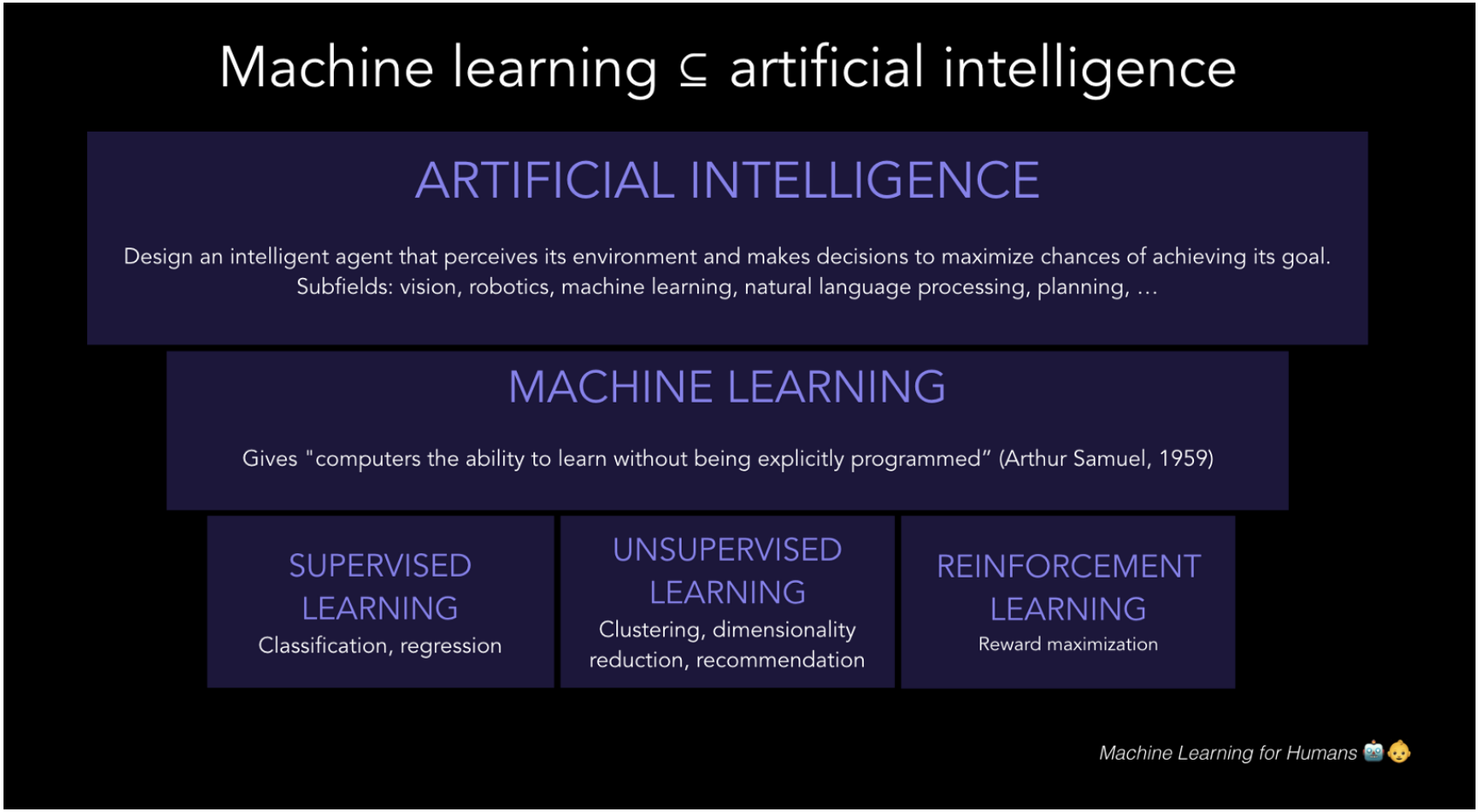


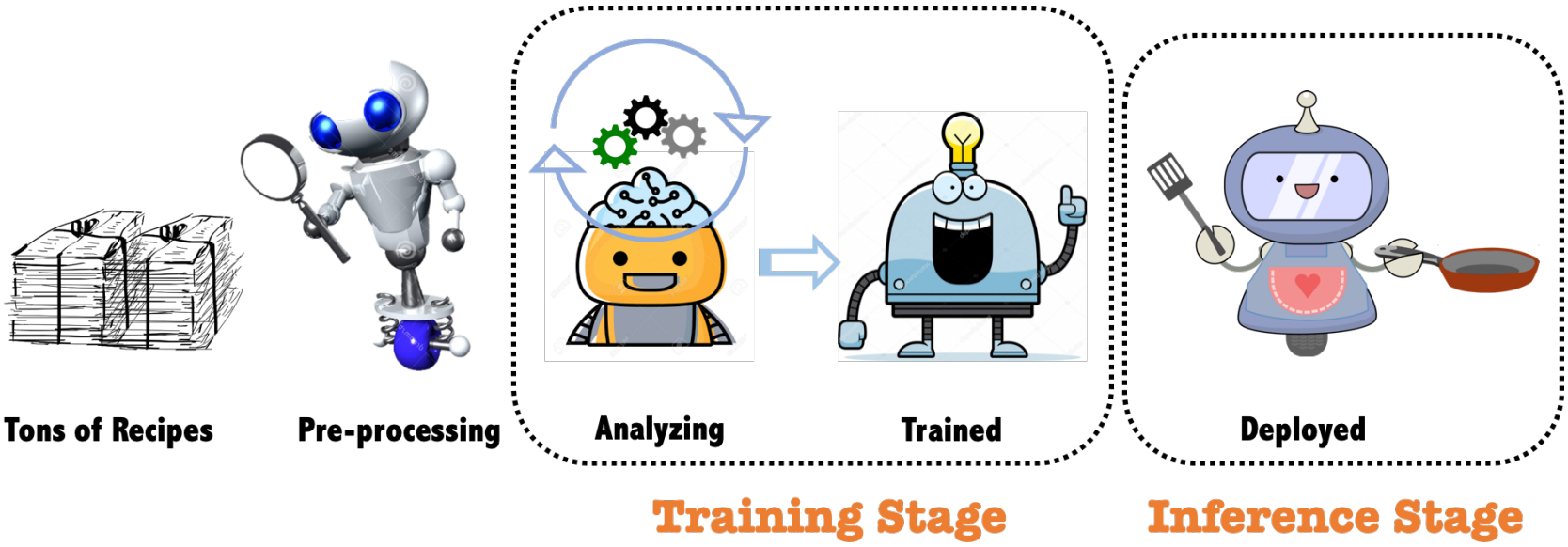
Figure: Self Driving Cars

Figure: Speech Processing



source: `https://medium.com/machine-learning-for-humans/why-machine-learning-matters-6164faf1df12`

Large-Scale Machine Learning Process



What is ML Training ?

- Mathematically, it is solving an ML **optimization** problem such as ridge regression, support vector machines, LASSO, sparse coding, neural networks etc
- For finding model parameters that **minimise** some objective (cost or loss) function
- Using a training **algorithm** which is generally iterative and sequential. For e.g. Interior Point Methods (IPM), Sequential Minimal Optimization (SMO), Stochastic Gradient Descent (SGD), etc

Challenges in ML training

ML Training, usually

- Requires high memory to store large-scale training dataset (bigdata)
- Involves lots of iterations
- Has huge computational cost
- Is inherently sequential

Challenges in ML training

ML Training, usually

- Requires high memory to store large-scale training dataset (bigdata)
- Involves lots of iterations
- Has huge computational cost
- Is inherently sequential

Training is EXPENSIVE !

Impact of Expensive Training

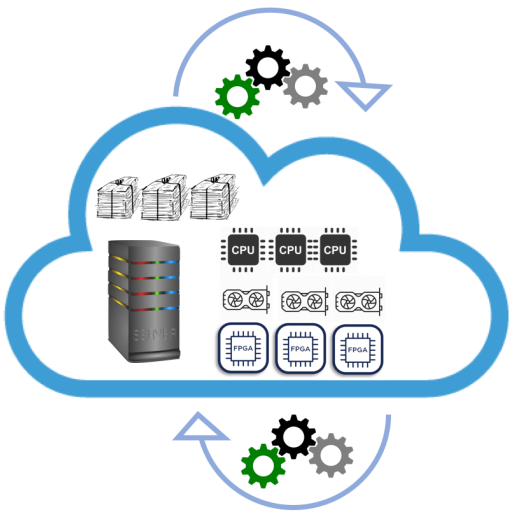
Training is EXPENSIVE !

Impact of Expensive Training

Training is EXPENSIVE !

$\Downarrow$

Training needs HPC cloud server !



HPC cloud server

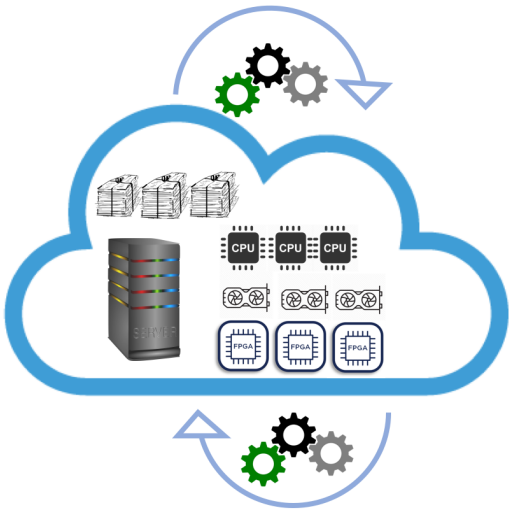
- Has massive storage
- Supports heavy computations
- Multiple high power CPUs, GPU, and FPGAs

Impact of Expensive Training

Training is EXPENSIVE !

⇓

Training needs HPC cloud server !



HPC cloud server

- Has massive storage
- Supports heavy computations
- Multiple high power CPUs, GPU, and FPGAs

⇓

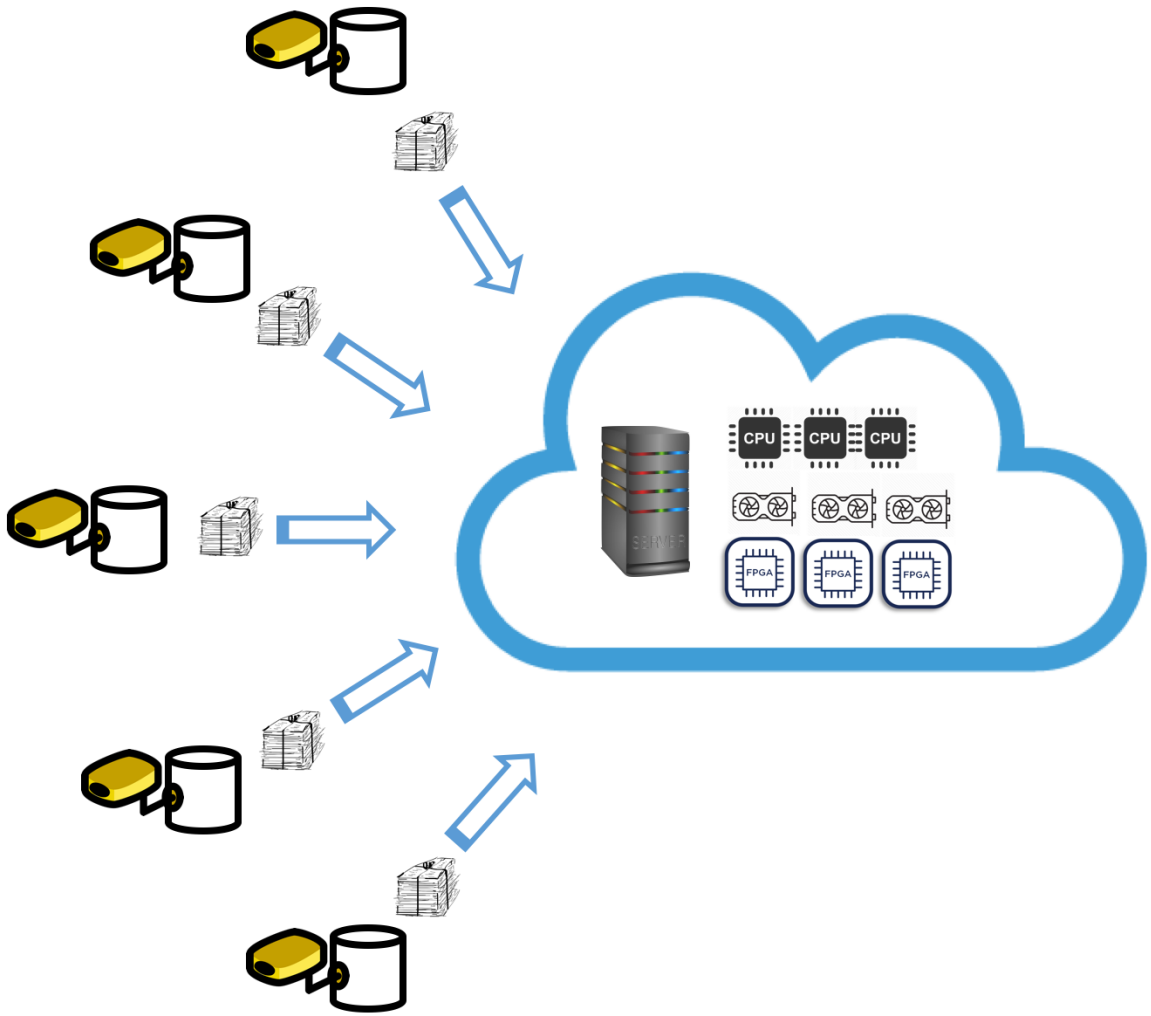
Training is CENTRALIZED !

A conventional large-scale Machine Learning framework

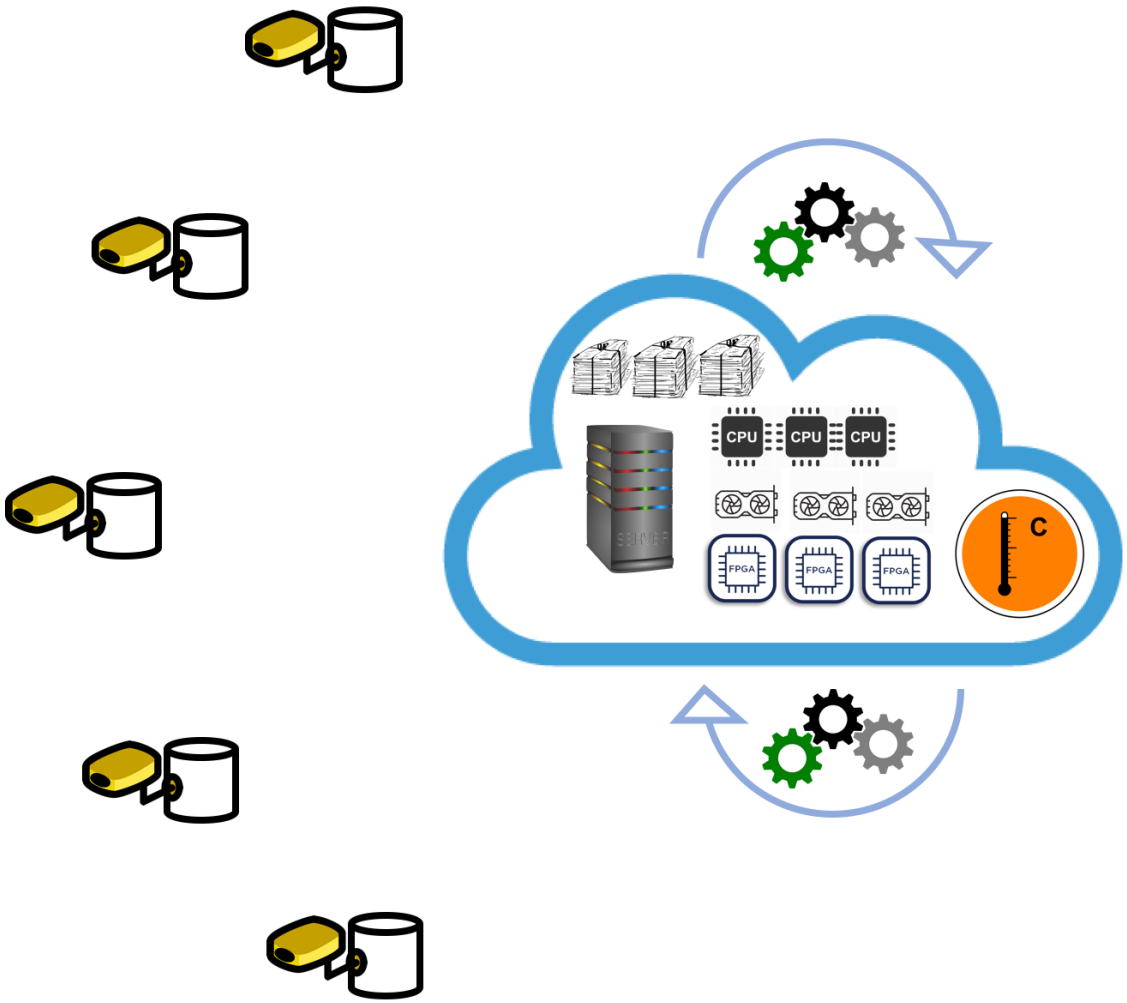
Centralized ML Framework

1. Collect data at Edge devices and transfer to HPC cloud server
2. Train ML model in the HPC cloud server using training algorithms
3. Deploy the trained model back at Edge devices for inference

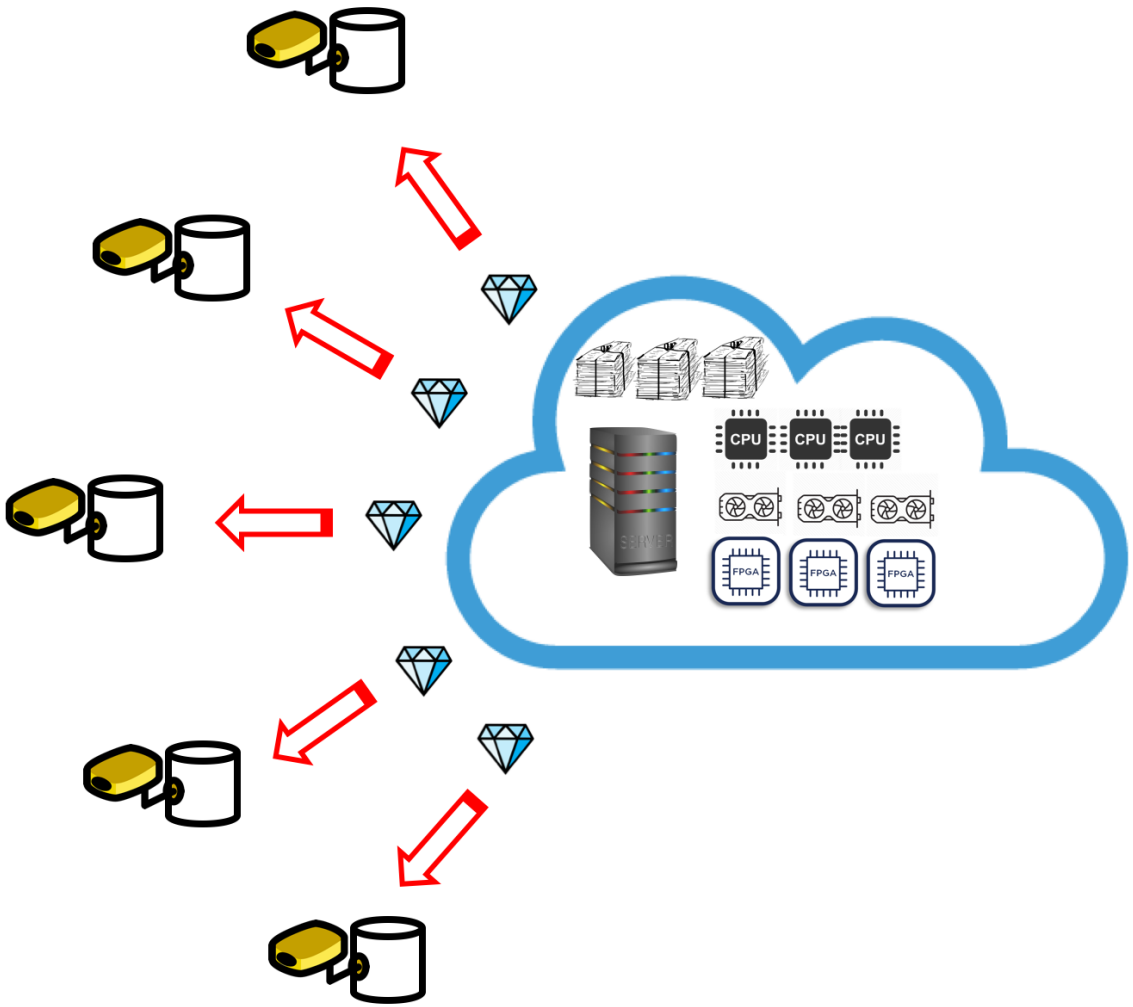
1. Data collection and transfer



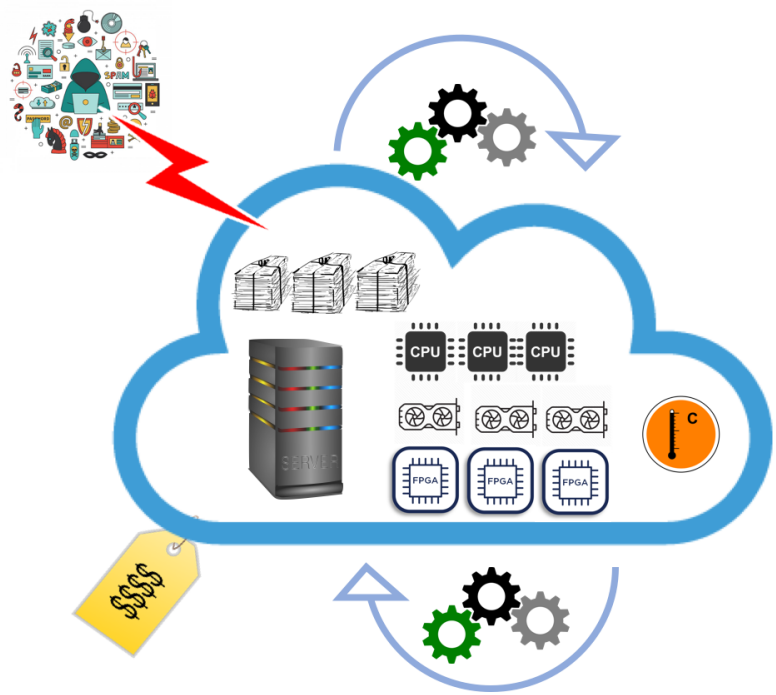
2. Training on HPC server: Centralized !



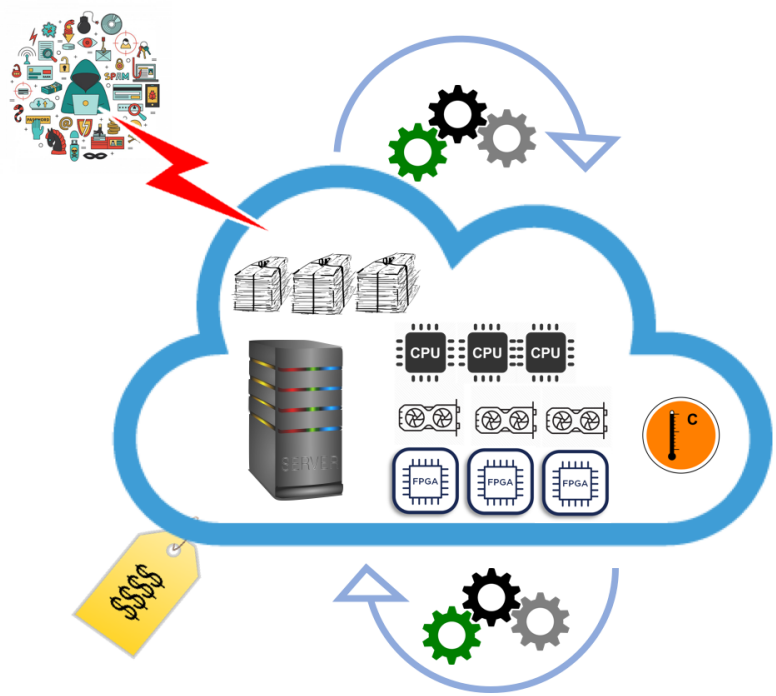
3. Deploying trained model for inference at edge



But, Centralized ML framework will not Scale! WHY ?



But, Centralized ML framework will not Scale! WHY ?

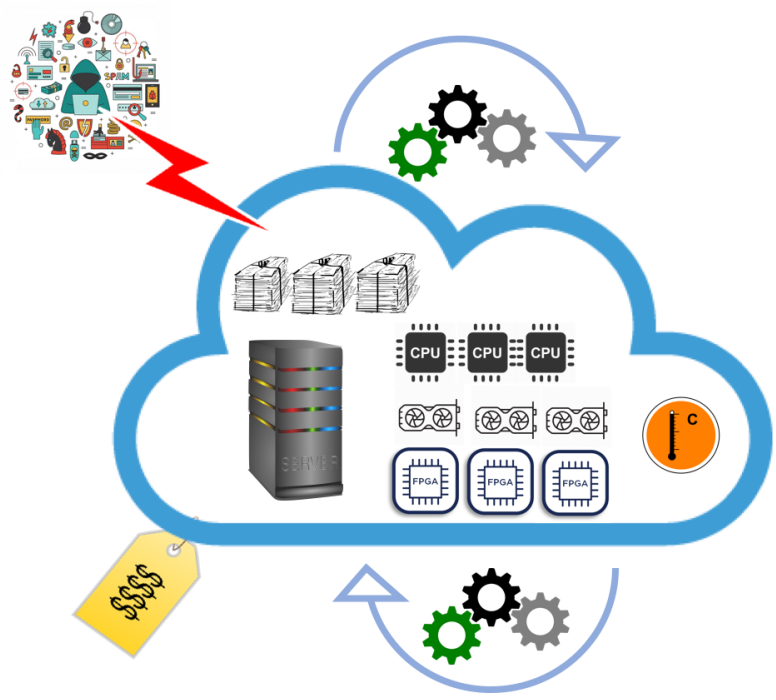


Because, there are issues with centralized training

At HPC cloud server

- Privacy at Risk
- High energy dissipation
- Expensive in cost $$$$
- High network latency

But, Centralized ML framework will not Scale! WHY ?



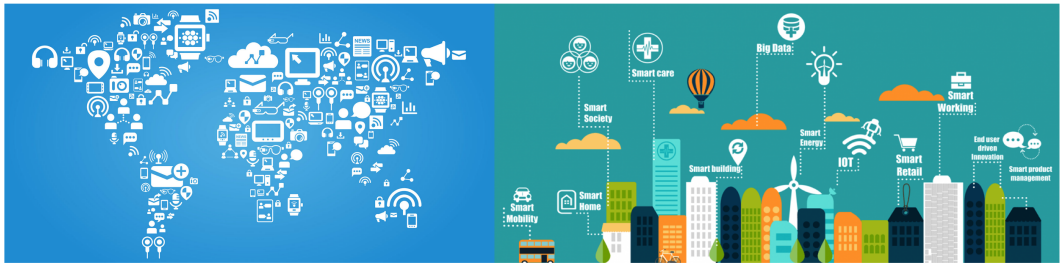
Because, there are issues with centralized training

At HPC cloud server

- Privacy at Risk
- High energy dissipation
- Expensive in cost $$$$
- High network latency

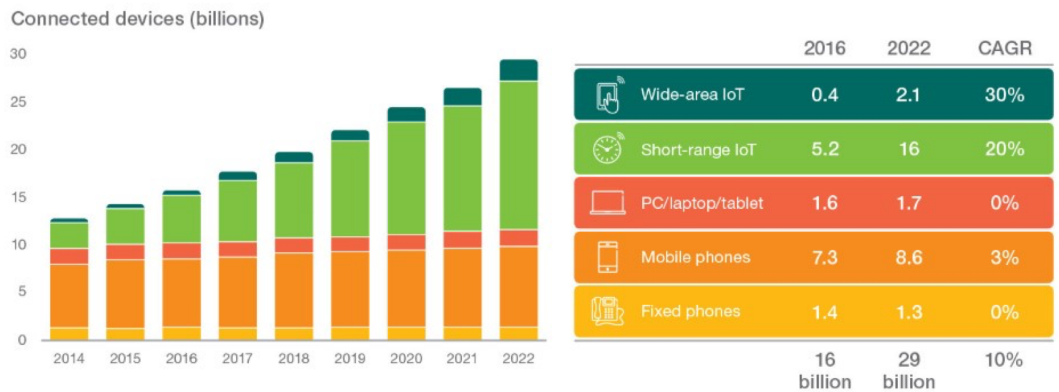
and, today there are Billions of connected small devices at Edge (IoT) having new requirements that can not be satisfied by Centralized framework

World of IoT



source: Ericsson

World of IoT



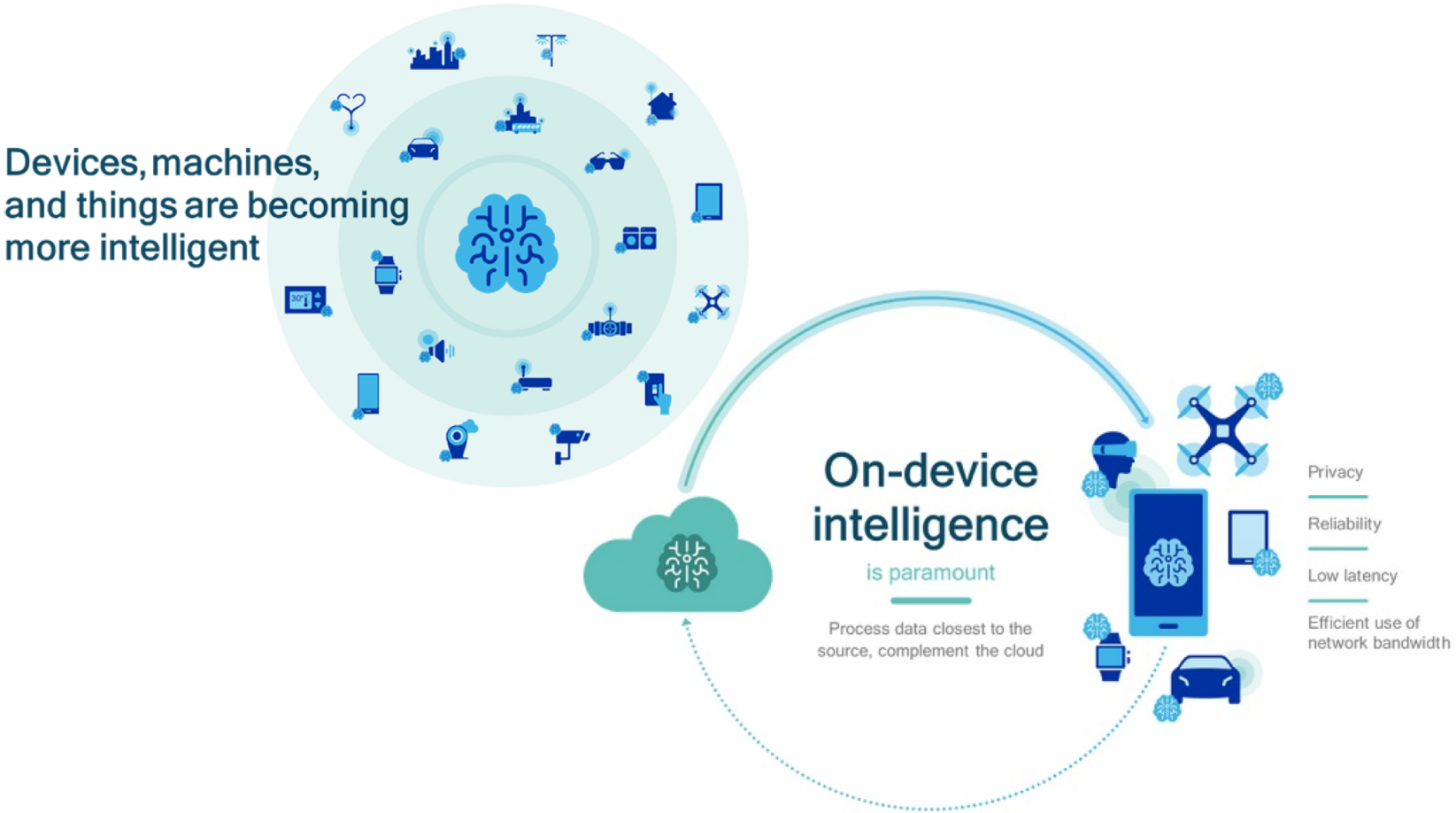
source: Ericsson

In IoT, there is need for

- Online/incremental learning to handle streaming data
- Real-time predictions for critical applications
- Fast and Energy-efficient computing models and architectures
- Efficient network utilization
- Keeping data private on device

Motivation

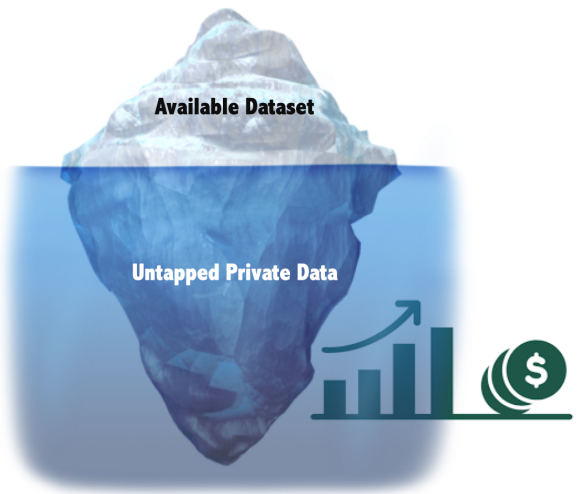
New Opportunities



Hidden Treasure

Untapped Private Data

- Available public data with big companies is just a tip of iceberg
- Utilize untapped private data in individual devices for ML with privacy protected



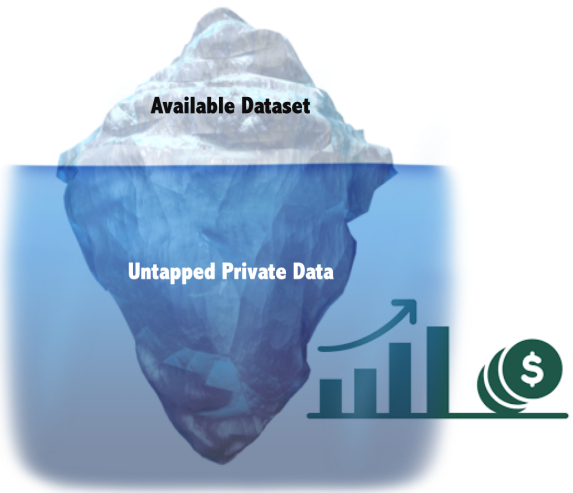
Hidden Treasure

Untapped Private Data

- Available public data with big companies is just a tip of iceberg
- Utilize untapped private data in individual devices for ML with privacy protected

Idle Processing Power

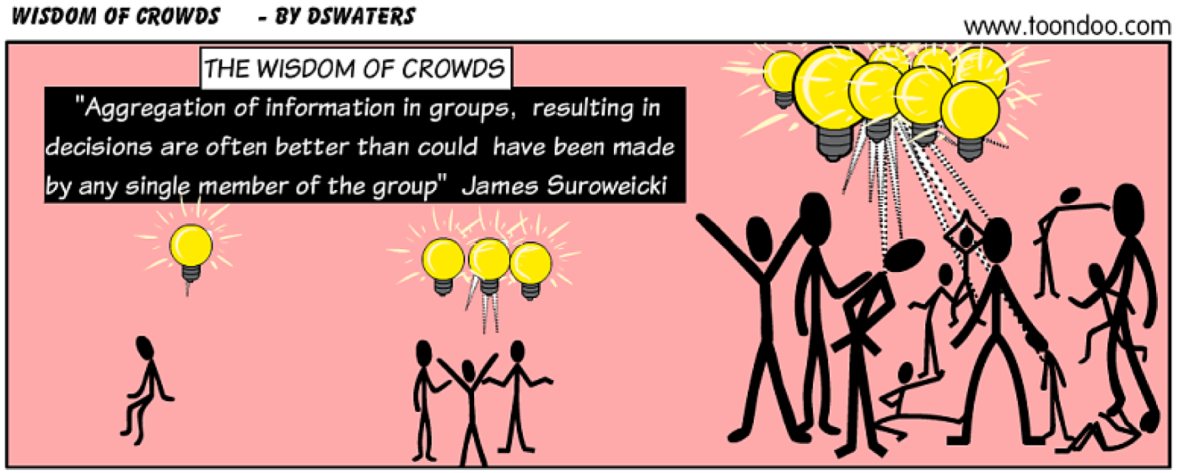
- Billions of connected devices with underutilized processing capacity
- Leverage their idle processing power to build Decentralized processor for ML !



source: https://decentralizedml.com/

Hypothesis

Decentralize ML(AI) and Give autonomy to the consumer
by bringing Training on Edge to
Unlock the potential of untapped private data, and to utilize idle processing power of edge devices connected over a distributed network



How to Decentralize ML ? Intuitively, by

How to Decentralize ML ? Intuitively, by

Making training CHEAP

How to Decentralize ML ? Intuitively, by

Making training CHEAP

$\Downarrow$

will REDUCE the need for HPC cloud server

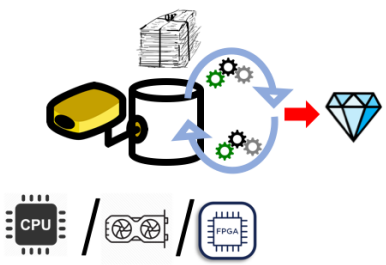
How to Decentralize ML ? Intuitively, by

Making training CHEAP

$\Downarrow$

will REDUCE the need for HPC cloud server

Bring training on edge



Edge Device

- Has less storage
- Handles light computations
- Single low power CPU/GPU/FPGA
- Low cost $$

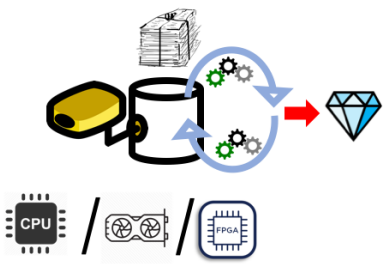
How to Decentralize ML ? Intuitively, by

Making training CHEAP

$\Downarrow$

will REDUCE the need for HPC cloud server

Bring training on edge



Edge Device

- Has less storage
- Handles light computations
- Single low power CPU/GPU/FPGA
- Low cost $$

$\Downarrow$

and AVOID centralization

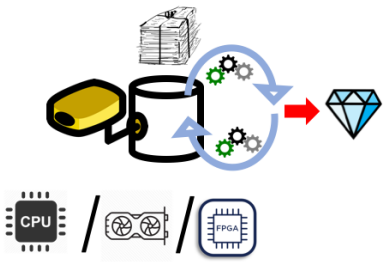
How to Decentralize ML ? Intuitively, by

Making training CHEAP

⇓

will REDUCE the need for HPC cloud server

Bring training on edge



Edge Device

- Has less storage
- Handles light computations
- Single low power CPU/GPU/FPGA
- Low cost $$

⇓

and AVOID centralization

Go for decentralized/distributed framework with parallel training

Various Decentralized Networks: Edge Server model

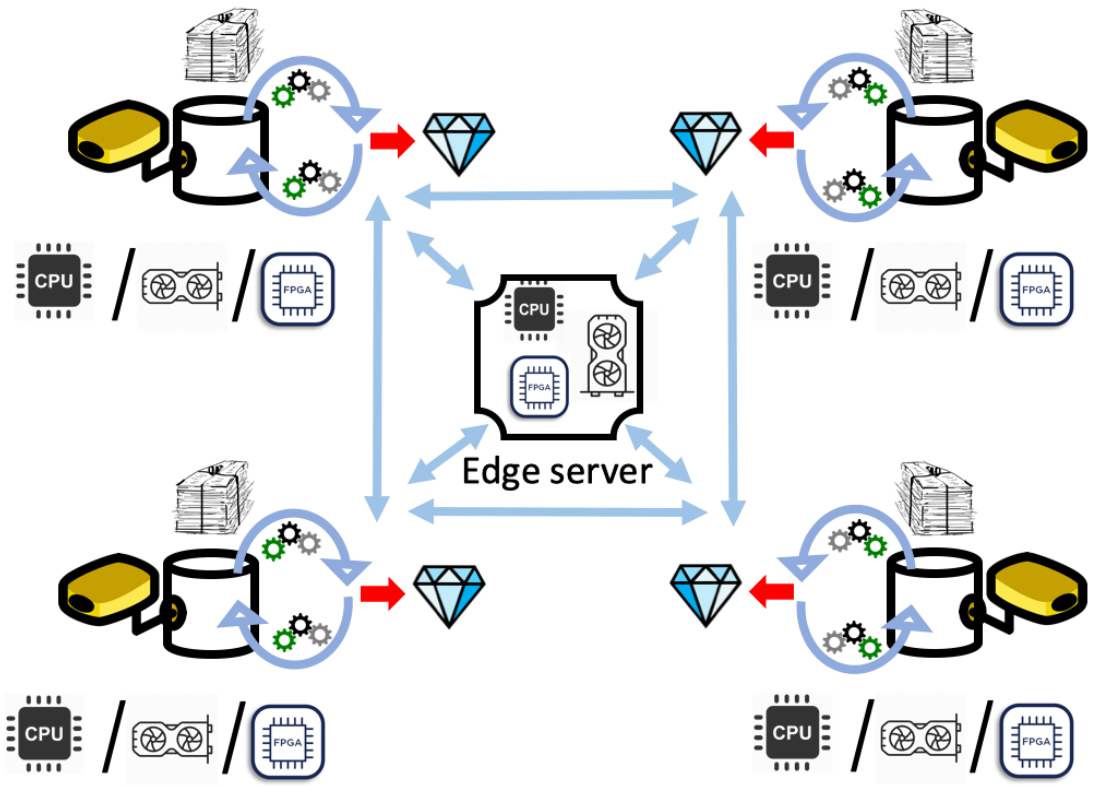


Figure: Edge devices collaborate with each other, and an Edge server if needed

Various Decentralized Networks: IoT model

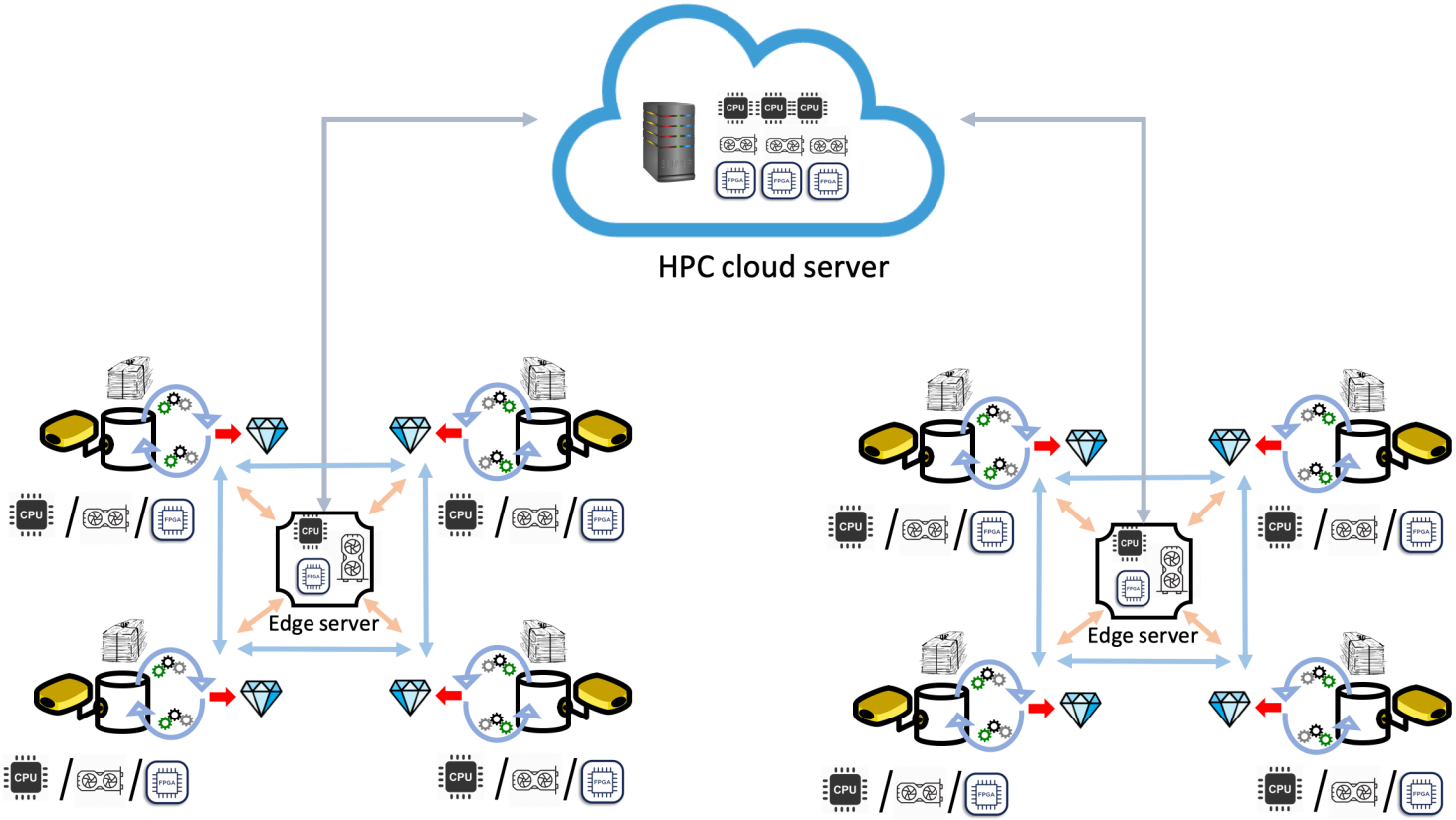


Figure: A typical IoT framework. Edge devices collaborate with each other, and an Edge server if needed. Multiple Edge servers may coordinate via HPC cloud server

Example: Federated Learning by Google AI

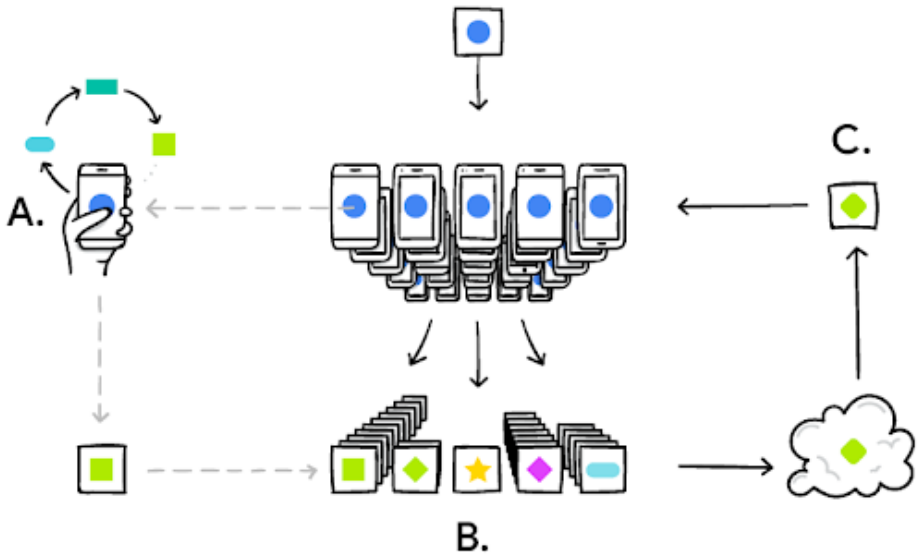


Figure: Your phone personalizes the model locally, based on your usage (A). Many users' updates are aggregated (B) to form a consensus change (C) to the shared model, after which the procedure is repeated.

source: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

Literature Review

Popular ML Training Algorithms

A. Interior Point Methods

B. Sequential Minimal Optimization

C. Stochastic Gradient Descent

A. Interior Point Methods

1. [Mehrotra, 1991]
   - minimize dual objective in a convex QP problem via primal-dual IPM
   - Incorporate Newton or Quasi-Newton methods
   - Memory cost: $O(n^2)$, where n : # samples
   - Computational cost: $O(n^3)$ per iteration
2. [Chang, 2007]
   - First study on parallel SVM for fast training (PSVM)
   - Parallel implementation of IPM
   - Incomplete Cholesky Factorization (ICF) per iteration which is difficult to parallelize
   - Memory cost: $O(\frac{n^{1.5}}{p})$, where p : # parallel processors
   - Computational cost: $O(\frac{n^2}{p})$ per iteration

B. Sequential Minimal Optimization

1. [Platt, 1998][Chang, 2001][Joachims, 2002]
   - State-of-the art SVM solver
   - Decomposes the large QP problem into a series of smallest possible sub-problems and solved analytically
   - Popular open-source packages LIBSVM [Chang, 2001] and SVM^{light} [Joachims, 2002]
   - Memory cost: Linear in n, where n : # samples. Trains large datasets on a single machine
   - Computational cost: between linear and quadratic in n as matrix computation is avoided.
2. [Zanghirati, 2003]
   - Parallel implementation of SVM^{light}
   - Maximum workload of 60K samples

C. Stochastic Gradient Descent

- Iterative method for optimizing a differentiable objective function
- Stochastic approximation of gradient descent optimization
- Popular approach for training linear SVM (in primal form), logistic regression, and artificial neural networks (deep learning) via backpropagation.
- [Zeyuan, 2009] solves *primal* form non-linear SVM in parallel using SGD (PpackSVM) but has expensive communication or synchronization overheads

Gaps in existing parallel ML training techniques

- Expensive **communication** (synchronization) overheads
- **Memory cost** is nearly **quadratic** with sample size
- **Computational cost** per iteration is **quadratic** with sample size
- Some parallel techniques are limited to small workload, thereby, can **not scale**

How to resolve these gaps ?

For efficient parallel ML training

- Reduce the communication (synchronization) overhead via asynchronous or relaxed synchronous techniques
- Incorporate memory-efficient representation of data by inducing sparsity
- Design highly separable and independent sub-problems amenable for parallelization
- Build algorithms than can scale linearly with number of parallel machines

Scope of Proposal

A new large-scale Machine Learning framework

Distributed ML Framework

1. Collect data at Edge devices and ~~transfer to HPC cloud server~~ retain it locally
2. Train ML model ~~in HPC cloud server sequentially~~ in parallel across multiple Edge devices (+ Edge server if needed)
3. Deploy the trained model ~~back~~ at Edge device for inference

Distributed training of large-scale ML

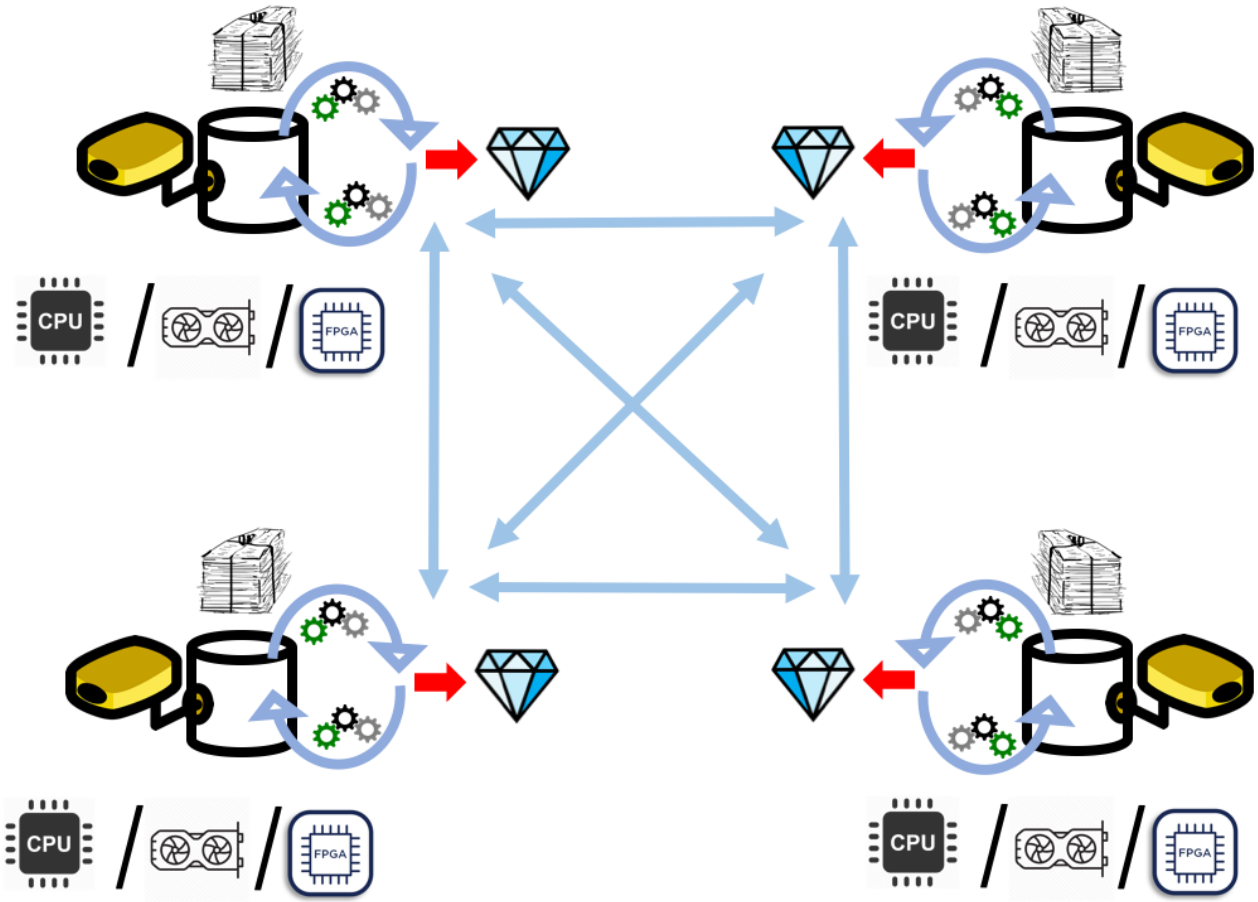


Figure: Decentralized network with no Edge server

Scope of Proposal

While current research in academia and industry is towards accelerating inference stage on edge devices, we target more computationally challenging stage of Training for Edge

Scope of Proposal

While current research in academia and industry is towards accelerating inference stage on edge devices, we target more computationally challenging stage of Training for Edge

Devise parallel algorithmic techniques with focus on

- Low latency and high throughput
- Less network communication overhead
- Memory-efficient representation
- Linear scalability
- Higher accuracy or atleast at par with state-of-the- art
- High energy-efficiency

Work so far

Work so far

Relaxed synchronization approach for solving parallel QP problems

IPDPS'16 [Lee, Dass, et al 2016]

Parallel and memory-efficient training of kernel SVM

ICDCS'17 [Dass, et al 2017]

Communication-efficient framework for scaling distributed SVM training

TPDS'18 [Dass, et al 2018]

Let's begin,

Relaxed synchronization approach for solving parallel QP problems

QP based Optimization Problems

Some applications of Quadratic Programming (QP) are

1. Least Square approximations
2. Regression Analysis
3. Least absolute shrinkage and selection operator (LASSO)
4. Support Vector Machines (SVM)

Parallel QP problem

Formulation

$$\min_{\boldsymbol{x}} \sum_{i=1}^{p} \left(\frac{1}{2} \boldsymbol{x}_i^T \boldsymbol{K}_i \boldsymbol{x}_i + \boldsymbol{c}_i^T \boldsymbol{x}_i \right)$$

$$\text{subject to} \quad \sum_{i=1}^{p} \boldsymbol{A}_i \boldsymbol{x}_i = \boldsymbol{b}$$

where,
$\boldsymbol{x}_i, \boldsymbol{c}_i \in \mathbb{R}^{\frac{n}{p}}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{A}_i \in \mathbb{R}^{m \times \frac{n}{p}}$, and $\boldsymbol{A} = [\boldsymbol{A}_1, \ldots, \boldsymbol{A}_p] \in \mathbb{R}^{m \times n}$
$\boldsymbol{K}_i \in \mathbb{R}^{\frac{n}{p} \times \frac{n}{p}}$ is symmetric positive definite matrix, and
$\boldsymbol{K} = \text{diag}(\boldsymbol{K}_1, \ldots, \boldsymbol{K}_p) \in \mathbb{R}^{n \times n}$
n : Number of variables, m : Number of constraints,
p : Number of separable (parallel) sub-problems

Solving parallel QP problems

Lagrangian $\mathcal{L}_i$ for each sub-problem, $i = 1, \ldots, p$

$$\mathcal{L}_i(\boldsymbol{x}_i, \boldsymbol{\beta}) = \left(\frac{1}{2}\boldsymbol{x}_i^T \boldsymbol{K}_i \boldsymbol{x}_i + \boldsymbol{c}_i^T \boldsymbol{x}_i\right) + \boldsymbol{\beta}^T(\boldsymbol{A}_i \boldsymbol{x}_i - \boldsymbol{b})$$

where, $\boldsymbol{\beta} \geq \mathbf{0}_m \in \mathbb{R}^m$ is a vector of Lagrangian dual variables

Parallel Dual Ascent

Dual sub-function: $g_i(\boldsymbol{\beta}) = \min_{\boldsymbol{x}_i} \mathcal{L}_i(\boldsymbol{x}_i, \boldsymbol{\beta})$

Dual problem: $\max_{\boldsymbol{\beta}} g(\boldsymbol{\beta})$

Parallel Dual Ascent steps

Gradient method - involves iterating through the following steps until convergence (error in $\boldsymbol{\beta}$ falls below stopping threshold)

Step 1: Minimization of Lagrangian for each sub-problem i

$$\begin{aligned} \boldsymbol{x}_i^{t+1} &= \arg\min_{\boldsymbol{x}_i} \mathcal{L}_i(\boldsymbol{x}_i, \boldsymbol{\beta}^t) \\ &= -\left(\boldsymbol{K}_i\right)^{-1} (\boldsymbol{A}_i^T \boldsymbol{\beta}^t + \boldsymbol{c}_i) \end{aligned}$$

solved in parallel using p machines which then broadcast local $\boldsymbol{A}_i \boldsymbol{x}_i^{t+1}$ during every iteration

Parallel Dual Ascent steps

Gradient method - involves iterating through the following steps until convergence (error in $\boldsymbol{\beta}$ falls below stopping threshold)

Step 1: Minimization of Lagrangian for each sub-problem i

$$\begin{aligned} \boldsymbol{x}_i^{t+1} &= \arg\min_{\boldsymbol{x}_i} \mathcal{L}_i(\boldsymbol{x}_i, \boldsymbol{\beta}^t) \\ &= -\left(\boldsymbol{K}_i\right)^{-1} (\boldsymbol{A}_i^T \boldsymbol{\beta}^t + \boldsymbol{c}_i) \end{aligned}$$

solved in parallel using p machines which then broadcast local $\boldsymbol{A}_i \boldsymbol{x}_i^{t+1}$ during every iteration

Step 2: Dual variable update (Global Gathering of $\boldsymbol{A}_i \boldsymbol{x}_i^{t+1}$)

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t + \eta(\sum_{i=1}^{p} \boldsymbol{A}_i \boldsymbol{x}_i^{t+1} - \boldsymbol{b})$$

$\eta > 0$ is the step size, $\boldsymbol{\beta}^{t=0} = \mathbf{0}_m$.

Parallel Dual Ascent steps

Gradient method - involves iterating through the following steps until convergence (error in $\boldsymbol{\beta}$ falls below stopping threshold)

Step 1: Minimization of Lagrangian for each sub-problem i

$$\begin{aligned} \boldsymbol{x}_i^{t+1} &= \arg\min_{\boldsymbol{x}_i} \mathcal{L}_i(\boldsymbol{x}_i, \boldsymbol{\beta}^t) \\ &= -\left(\boldsymbol{K}_i\right)^{-1} (\boldsymbol{A}_i^T \boldsymbol{\beta}^t + \boldsymbol{c}_i) \end{aligned}$$

solved in parallel using p machines which then broadcast local $\boldsymbol{A}_i \boldsymbol{x}_i^{t+1}$ during every iteration

Step 2: Dual variable update (Global Gathering of $\boldsymbol{A}_i \boldsymbol{x}_i^{t+1}$)

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t + \eta(\sum_{i=1}^{p} \boldsymbol{A}_i \boldsymbol{x}_i^{t+1} - \boldsymbol{b})$$

$\eta > 0$ is the step size, $\boldsymbol{\beta}^{t=0} = \mathbf{0}_m$.

Synchronization is Necessary and Unavoidable in Step 2

Distributed Computing: Synchronization as Bottleneck

Facts

- Synchronization across multiple machines results in idle process time
- Leads to waste of computing time for large-scale parallel computing
- Need to relax the synchronization penalty

How to avoid synchronization latency ?

1. Asynchronous Computing
   - Proceed with computation without waiting for values i.e. no synchronization
   - Asynchrony causes randomness in computing values
   - Cannot guarantee the numerical stability and convergence of solution

How to avoid synchronization latency ?

1. Asynchronous Computing
   - Proceed with computation without waiting for values i.e. no synchronization
   - Asynchrony causes randomness in computing values
   - Cannot guarantee the numerical stability and convergence of solution
2. Relaxed Synchronization
   - Do not synchronize at every iteration
   - Communicate data periodically
   - Objective is to minimize the number (frequency) of communication

We propose a relaxed synchronization approach for Dual variable update

LSDA: Lazily Synchronized Dual Ascent

Relaxed Synchronization approach: LSDA

TSDA: Tightly Synchronized Dual Ascent

$$\beta^{t+1} = \beta^t + \eta(\sum_{i=1}^{p} \boldsymbol{A}_i \boldsymbol{x}_i^{t+1} - \boldsymbol{b}) = \beta^t + \eta \sum_{i=1}^{p} \left(\boldsymbol{A}_i \boldsymbol{x}_i^{t+1} - \frac{1}{p} \boldsymbol{b} \right)$$

Relaxed Synchronization approach: LSDA

TSDA: Tightly Synchronized Dual Ascent

$$\beta^{t+1} = \beta^t + \eta(\sum_{i=1}^{p} \boldsymbol{A}_i \boldsymbol{x}_i^{t+1} - \boldsymbol{b}) = \beta^t + \eta \sum_{i=1}^{p} \left(\boldsymbol{A}_i \boldsymbol{x}_i^{t+1} - \frac{1}{p} \boldsymbol{b} \right)$$

LSDA: Lazily Synchronized Dual Ascent

$$\beta^{t+1} = \beta^t + \eta \sum_{i=1}^{p} \left(\boldsymbol{A}_i \boldsymbol{x}_i^{kP+1} - \frac{1}{p} \boldsymbol{b} \right), \quad kP \leq t < (k+1)P,$$

where, $k \in \mathbb{N}$, and $P \geq 1$ is the synchronization period

Synchronize across p parallel processes after every P iterations

Convergence of LSDA

LSDA dual variable converges to TSDA solution [Lee, Dass, et al 2016]

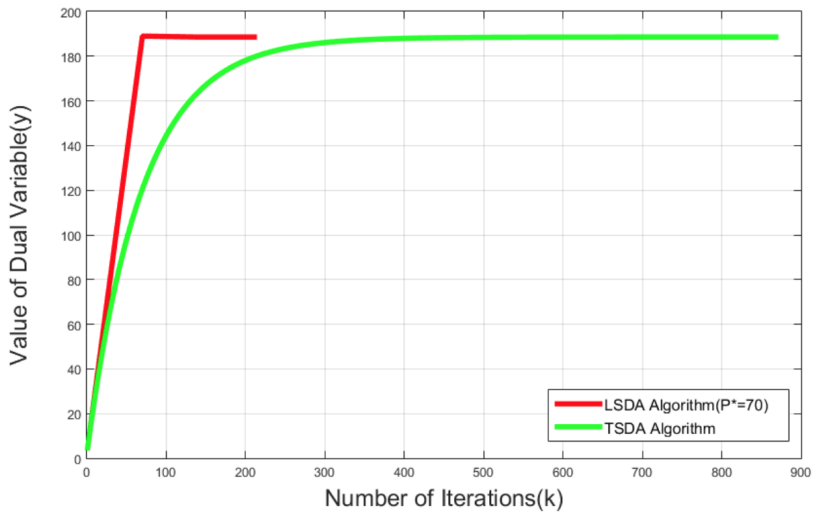


Figure: Dual variable solution vs Number of iterations. LSDA algorithm converges to the optimal solution of the dual variable significantly faster than the TSDA algorithm

Experimental Results

The data was synthetically generated with random values uniformly distributed over $[-1, 1]$. The problem specifics are as follows:

Experimental Setup

1) Number of instances in synthetic dataset, $d = 200,000$.
2) Step size, $\alpha = 0.27$.
3) Optimal Synchronization Period, $P^* = 70$.
4) Stopping threshold, $\epsilon = 10^{-5}$.
5) Cluster Size, $N = \{10, 20, 32, 40\}$.

Computing Performance Comparison between TSDA & LSDA algorithm

	TSDA algorithm	LSDA algorithm
No. of iteration	868	211
Sync. period	1	70
No. of Sync.	868 (=868/1) times	3 (=211/70) times
Comm. delay reduction	99.65%	
Speedup	160 times	

Work so far

Relaxed synchronization approach for solving parallel QP problems

[IPDPS'16] K. Lee, R. Bhattacharya, J. Dass, V. N. S. P. Sakuru, R. N. Mahapatra, "A Relaxed Synchronization Approach for Solving Parallel Quadratic Programming Problems with Guaranteed Convergence,"

Next, let's solve a popular QP-based ML model in parallel

Parallel and memory-efficient training of kernel SVM

Support Vector Machines (SVM)

We focus on distributed data analytics using SVM

- Supervised machine learning model (data+label)
- Widely used for data classification for its high efficiency
- Popular for multivariate non-linear datasets (kernel SVM)
- Also used for regression analysis (SVR)

SVM as a QP problem

- SVM is a convex optimization problem (QP)
- Solves for maximal separating hyperplane as a classifier

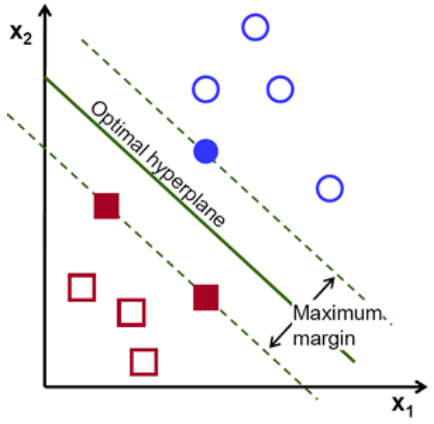


Figure: Linear SVM

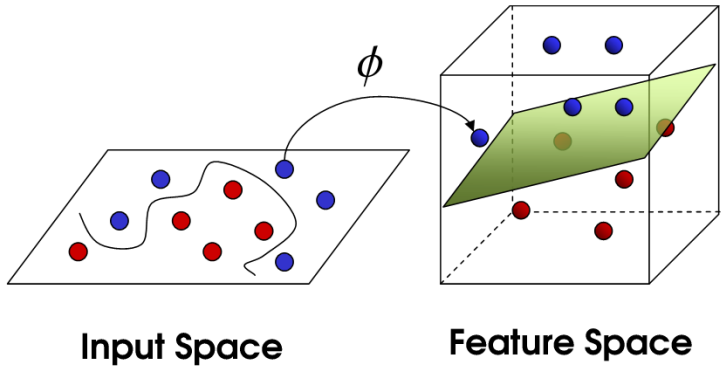


Figure: Non-Linear/Kernel SVM

- Maps training data into a high dimensional feature space via a nonlinear function (kernel SVM)
- Hence, solving for *dual* (rather than *primal*) form is preferred using "kernel trick"

SVM formulation

training dataset, $\mathcal{D} = \{(\mathbf{x_i}, y_i), i = 1, ..., n\}$
input data matrix, $\mathbf{X} = \{\mathbf{x_i} \in \mathbb{R}^d, i = 1...n\}$, d-dimensional space
class label vector, $\mathbf{y} = \{y_i \in \{-1, 1\}, i = 1...n\}$

SVM *primal*

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \max\left(1 - y_i\mathbf{w}^T\mathbf{x_i}, 0\right)^2$$

where, $\mathbf{w} \in \mathbb{R}^d$ is normal to the classifying hyperplane
$\mathbf{w}^T\mathbf{x_i} + b = 0$
$C > 0$ is penalty parameter for misclassification
For simplicity: Bias term b is appended with $\mathbf{w}$

SVM formulation

The equivalent dual formulation is as follows

SVM *dual*

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T\left(\mathbf{diag}(y) \times \mathbf{K} \times \mathbf{diag}(y)^T\right)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\left(\frac{1}{2C}\mathbf{I}_n\right)\boldsymbol{\alpha} + \mathbf{e}^T\boldsymbol{\alpha}$$

$$\text{subject to} \quad -\mathbf{I}_n\boldsymbol{\alpha} \leq \mathbf{0}_n$$

where, $\boldsymbol{\alpha} \in \mathbb{R}^n$ is a vector of *dual* variables

$\mathbf{e} = -\mathbf{1}_n$

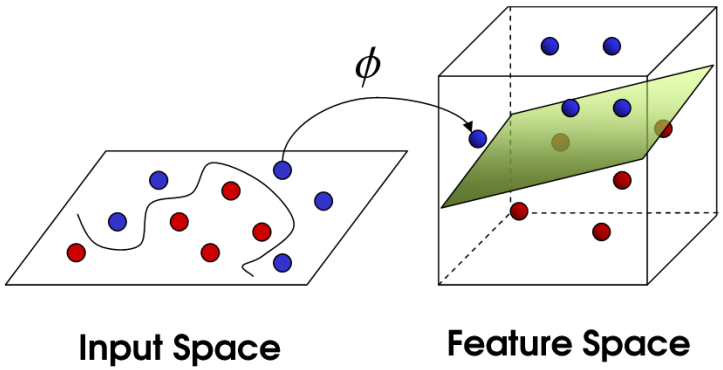
$\mathbf{K} = \{k(\mathbf{x_i}, \mathbf{x_j}), \forall i, j = 1...n\}$ is positive definite matrix (mostly)

$k()$: $\mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ represents the Mercer kernel function - linear/non-linear

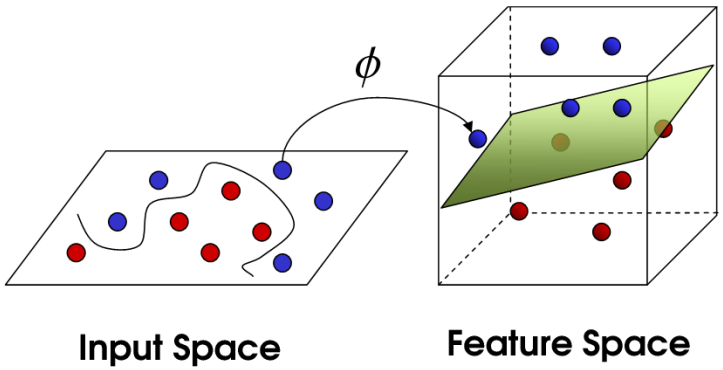
Kernel function:

$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$

where, $\phi()$ is a mapping generally not known or inefficient to compute.



Kernel function:
$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$
where, $\phi()$ is a mapping generally not known or inefficient to compute.



However,
$k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is known and easier to compute ("Kernel trick").

- Linear kernel : $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$
- Radial Basis Function (RBF) kernel:
  $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$, where, γ is hyperparameter

Measures "similarity" between two data points in the Feature space

Challenge

For large sample size n, Kernel methods become unfeasible because

1. $\mathbf{K}$ requires $O(n^2)$ memory and
2. it incurs computational cost of $O(n^3)$ to solve such problems

Low Rank Kernel Approximation !

Low rank approximation of $\mathbf{K}$

$$\mathbf{K} \approx \mathbf{A}\mathbf{A}^T, \text{ where, } \mathbf{A} \in \mathbb{R}^{n \times k} \text{ and } k \ll n.$$

We use MEKA [Si, 2014] for memory efficient and lower approximation error compared to Nyström methods etc.

Recall,

SVM *dual*

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T\Big(\mathbf{diag}(y) \times \boldsymbol{K} \times \mathbf{diag}(y)^T\Big)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\Big(\frac{1}{2C}\mathbf{I}_n\Big)\boldsymbol{\alpha} + \mathbf{e}^T\boldsymbol{\alpha}$$

$$\text{subject to} \quad -\mathbf{I}_n\boldsymbol{\alpha} \leq \mathbf{0}_n$$

Recall,

SVM *dual*

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T\Big(\mathbf{diag}(y) \times \boldsymbol{K} \times \mathbf{diag}(y)^T\Big)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\Big(\frac{1}{2C}\mathbf{I}_n\Big)\boldsymbol{\alpha} + \mathbf{e}^T\boldsymbol{\alpha}$$

$$\text{subject to} \quad -\mathbf{I}_n\boldsymbol{\alpha} \leq \mathbf{0}_n$$

Substitute, $\mathbf{K} \approx \mathbf{A}\mathbf{A}^T$ and define, $\hat{\mathbf{A}} = \mathbf{diag}(y) \times \mathbf{A}$

SVM *dual* with Kernel Approximation

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T\Big(\hat{\mathbf{A}}\hat{\mathbf{A}}^T\Big)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\Big(\frac{1}{2C}\mathbf{I}_n\Big)\boldsymbol{\alpha} + \mathbf{e}^T\boldsymbol{\alpha}$$

$$\text{subject to} \quad -\mathbf{I}_n\boldsymbol{\alpha} \leq \mathbf{0}_n$$

Goal

To devise a fast and memory-efficient distributed framework to train large-scale SVM in parallel

Goal

To devise a fast and memory-efficient distributed framework to train large-scale SVM in parallel

We propose,

1. QRSVM: QR decomposition framework for **memory-efficient** modeling and training of SVM
2. Optimal step size calculation for **fast convergence**
3. Distributed QRSVM for decomposing SVM into parallel equivalent sub-problems that are trained **in parallel**

Propose: QRSVM framework for SVM training

- Uses QR decomposition technique on Kernel matrix
- Reformulates the SVM problem to be memory-efficient
- Renders SVM separable and amenable for parallel training

Comprises of 2 stages

1. QR decomposition
2. Dual ascent

QR Decomposition of a Matrix

$\hat{\mathbf{A}} \in \mathbb{R}^{n \times k}$ with $k \ll n$ has a tall and skinny (TS) structure

QR decomposition

$\hat{\mathbf{A}} = \mathbf{QR}$, where,
$\mathbf{Q} \in \mathbb{R}^{n \times n}$ is Orthogonal matrix
$\mathbf{R} \in \mathbb{R}^{n \times k}$ is Upper Triangular matrix

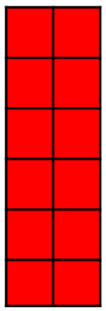


Figure: $\hat{\mathbf{A}}$

$\mathbf{Q}$, $O(n^2)$ stored as set of $k-$ Householder reflector vectors, $O(nk)$

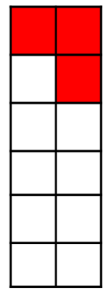


Figure: $\mathbf{R}$

Formulation

Recall,

SVM *dual* with Kernel Approximation

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T\left(\hat{\mathbf{A}}\hat{\mathbf{A}}^T\right)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\left(\frac{1}{2C}\mathbf{I}_n\right)\boldsymbol{\alpha} + \mathbf{e}^T\boldsymbol{\alpha}$$

$$\text{subject to} \quad -\mathbf{I}_n\boldsymbol{\alpha} \leq \mathbf{0}_n$$

Not Separable into Independent and Parallel sub-problems

Formulation

Substituting $\hat{A} = QR$

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T\left(\mathbf{QRR}^T\mathbf{Q}^T\right)\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^T\left(\frac{1}{2C}\mathbf{I}_n\right)\boldsymbol{\alpha} + \mathbf{e}^T\boldsymbol{\alpha}$$

$$\text{subject to} \quad -\mathbf{I}_n\boldsymbol{\alpha} \leq \mathbf{0}_n$$

Let us define,
$\hat{\boldsymbol{\alpha}} = \mathbf{Q}^T\boldsymbol{\alpha}$, $\hat{\mathbf{e}} = \mathbf{Q}^T\mathbf{e}$ and using $\mathbf{Q}^T\mathbf{Q} = \mathbf{I_n}$

Formulation

QRSVM: Memory-efficient modeling of traditional SVM

$$\min_{\hat{\alpha}} \frac{1}{2}\hat{\alpha}^T\left(\boldsymbol{R}\boldsymbol{R}^T + \frac{1}{2C}\boldsymbol{I}_n\right)\hat{\alpha} + (\hat{\boldsymbol{e}})^T\hat{\alpha}$$

$$\text{subject to} \quad -\boldsymbol{Q}\hat{\alpha} \leq \boldsymbol{0}_n$$

Recall, the Hessian of the objective function was Dense $\left(\boldsymbol{K} + \frac{1}{2C}\boldsymbol{I}_n\right)$ and, then $\left(\hat{\boldsymbol{A}}\hat{\boldsymbol{A}}^T + \frac{1}{2C}\boldsymbol{I}_n\right)$

Memory-efficient modeling

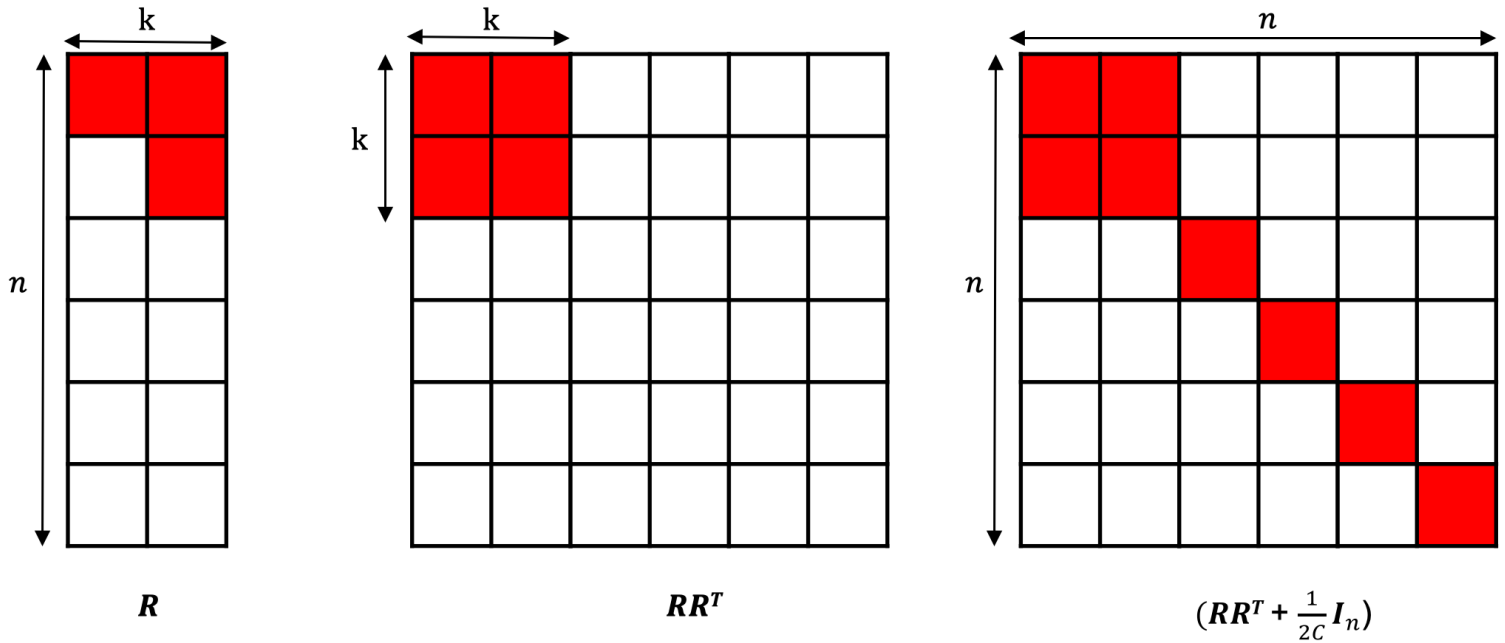


Figure: Non-colored denoted zeros. Ideally, $k \ll n$

Solve Optimization problem

Lagrangian $\mathcal{L}$ of QRSVM

$$\mathcal{L}(\hat{\alpha}, \beta) = \frac{1}{2}\hat{\alpha}^T\left(\mathbf{R}\mathbf{R}^T + \frac{1}{2C}\mathbf{I}_n\right)\hat{\alpha} + (\hat{\mathbf{e}})^T\hat{\alpha} + \beta^T(-\mathbf{Q}\hat{\alpha})$$

where, $\beta \geq \mathbf{0}_n$ is the Lagrangian dual variable.

Dual Decomposition via projected sub-gradient

Dual function: $g(\beta) = \min_{\hat{\alpha}} \mathcal{L}(\hat{\alpha}, \beta)$

Dual Problem: $\max_{\beta} g(\beta)$

Iterative Steps

Projected sub-gradient method - involves iterating through the following steps, followed by ensuring *dual* variable is thresholded component wise (ReLU function) until convergence (error in $\boldsymbol{\beta}$ falls below stopping threshold)

Step 1: Minimization of Lagrangian

$$\begin{aligned}\hat{\boldsymbol{\alpha}}^{t+1} &= \arg\min_{\hat{\boldsymbol{\alpha}}} \mathcal{L}(\hat{\boldsymbol{\alpha}}, \boldsymbol{\beta}^t) \\ &= -\left(\mathbf{R}\mathbf{R}^T + \frac{1}{2C} \times \mathbf{I}_n\right)^{-1} (-\mathbf{Q}^T \boldsymbol{\beta}^k + \hat{\mathbf{e}})\end{aligned}$$

Step 2: Dual variable update

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t + \eta(-\mathbf{Q}\hat{\boldsymbol{\alpha}}^{t+1})$$

$\eta > 0$ is the step size, $\boldsymbol{\beta}^0 = \mathbf{0}_n$.

QRSVM Workflow

Two stages of QRSVM

1. **QR decomposition**: Computational cost $O(nk^2)$
2. **Dual Ascent method**: Computational cost $O(nk)$ per iteration

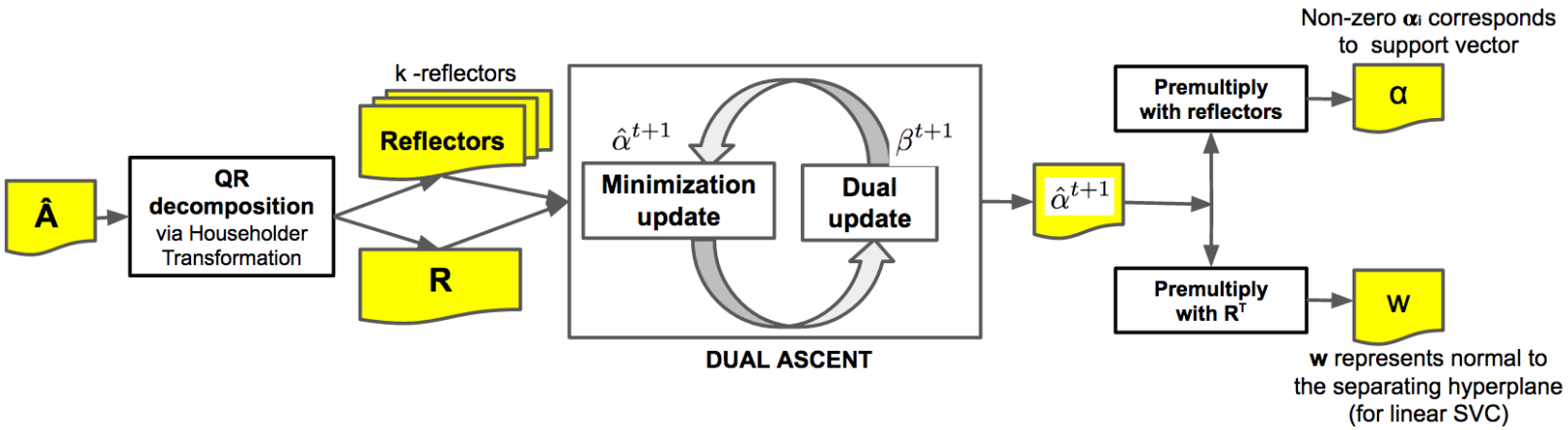


Figure: QRSVM workflow

Propose: Optimal Step Size for fast convergence

Based on optimal synchronization period defined for Lazily Synchronous Dual Ascent method , Theorem 1
[Lee, Dass, et al 2016]

Scaling factor for optimal step size

Theorem

To ensure the minimum number of iterations involving the dual variable update step, the scaling factor $P^\star$ for optimal step size is obtained by

$$P^\star = \max \operatorname*{argmin}_{P \in \mathbb{N}} \max\{|1 - \lambda_{min}(\boldsymbol{M})P|, |1 - \lambda_{max}(\boldsymbol{M})P|\}$$

$$\boldsymbol{M} := \eta\left(\boldsymbol{R}\boldsymbol{R}^T + \tfrac{1}{2C}\boldsymbol{I}_n\right)^{-1},$$

$\eta > 0$ is step size

$\lambda_{min}(\cdot)$ and $\lambda_{max}(\cdot)$ eigenvalues of matrix $\boldsymbol{M}$

Optimal step size

Lemma

For any $\eta > 0$, the optimal step size $\eta^\star$ can be computed using

$$\eta^\star = P^\star \eta, \quad P^\star \in \mathbb{N}$$

where,

$$P^\star = \begin{cases} 1 & \text{if} \quad 0 < \bar{\lambda}^{-1} < 2 \\ \lfloor \bar{\lambda}^{-1} \rfloor & \text{if} \quad \bar{\lambda}^{-1} \geq 2 \end{cases}$$

and $\bar{\lambda} = (\lambda_{max}(\mathbf{M}) + \lambda_{min}(\mathbf{M}))/2$

Now that SVM is modeled as a **memory-efficient QRSVM** and we can solve it with **fast convergence** using an **optimal step size**,

Is it possible to decompose SVM into multiple independent and parallel sub-problems for our Decentralized ML framework ?

Can we make SVM separable?

Can we make SVM separable?

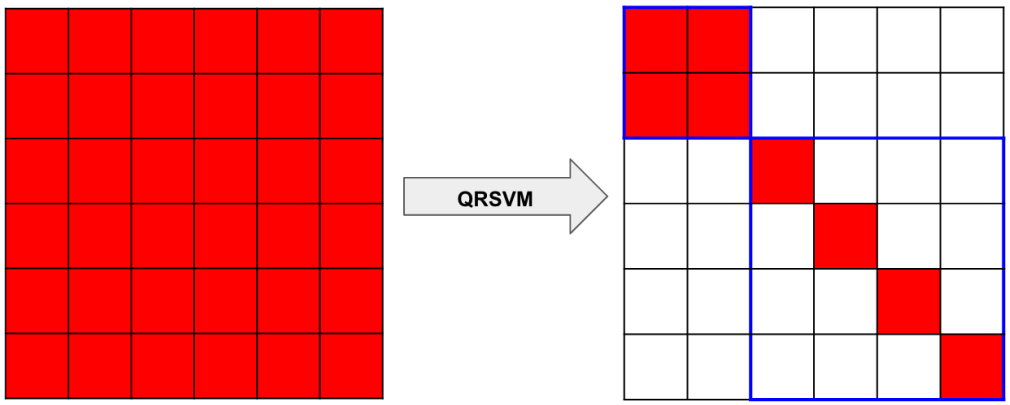
YES, QRSVM renders SVM separable into independent sub-problems!

Can we make SVM separable?

YES, QRSVM renders SVM separable into independent sub-problems!

Structure of Hessian matrix

$$\left(\hat{\mathbf{A}}\hat{\mathbf{A}}^T + \frac{1}{2C}\mathbf{I}_n\right) \Rightarrow \left(\mathbf{R}\mathbf{R}^T + \frac{1}{2C}\mathbf{I}_n\right)$$



Dense $O(n^2)$
Non-separable

Sparse $O(k^2)$
Block diagonal separable

Proposed: Distributed QRSVM

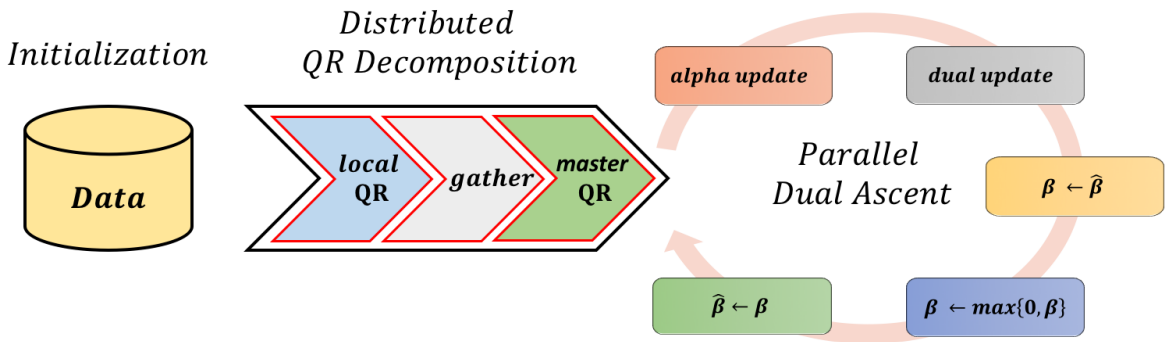
- Leverages separable structure of QR-based SVM into independent and identical sub-problems
- Trains SVM model by solving these sub-problems in parallel on multiple machines

Proposed: Distributed QRSVM

- Leverages separable structure of QR-based SVM into independent and identical sub-problems
- Trains SVM model by solving these sub-problems in parallel on multiple machines

Comprises of 2 stages

1. Distributed QR decomposition
2. Parallel Dual ascent



Stage 1: Distributed QR decomposition

Partition data horizontally, $\hat{\boldsymbol{A}}_{\boldsymbol{i}} \in \mathbb{R}^{\hat{n} \times k}$ on p computing cores such that

$$k \ll \hat{n} = \frac{n}{p} \implies p \ll \frac{n}{k}$$

Pseudocode

1. $\hat{\boldsymbol{A}}_{\boldsymbol{i}} = \boldsymbol{Q_i R_i}$ at each worker core i
2. Gather all $\boldsymbol{R}_i$ at Master core
3. $[\boldsymbol{R_1}^T, \ldots, \boldsymbol{R_p}^T]^T = \boldsymbol{Q_g R_g}$ at Master core

where, $\boldsymbol{Q_i}$ and $\boldsymbol{Q_g}$ - orthogonal matrices, and $\boldsymbol{R_i}$ and $\boldsymbol{R_g}$ - upper triangular matrices.

One can represent the factors $\boldsymbol{Q}$ and $\boldsymbol{R}$ of the complete $\hat{\boldsymbol{A}}$ as

$$\boldsymbol{Q} = \text{diag}(\boldsymbol{Q_1}, \boldsymbol{Q_2}, .. \boldsymbol{Q_i}, .., \boldsymbol{Q_p}) \times \boldsymbol{Q_g}$$

$$\boldsymbol{R} = \boldsymbol{R_g}$$

Implementation of Distributed QR decomposition

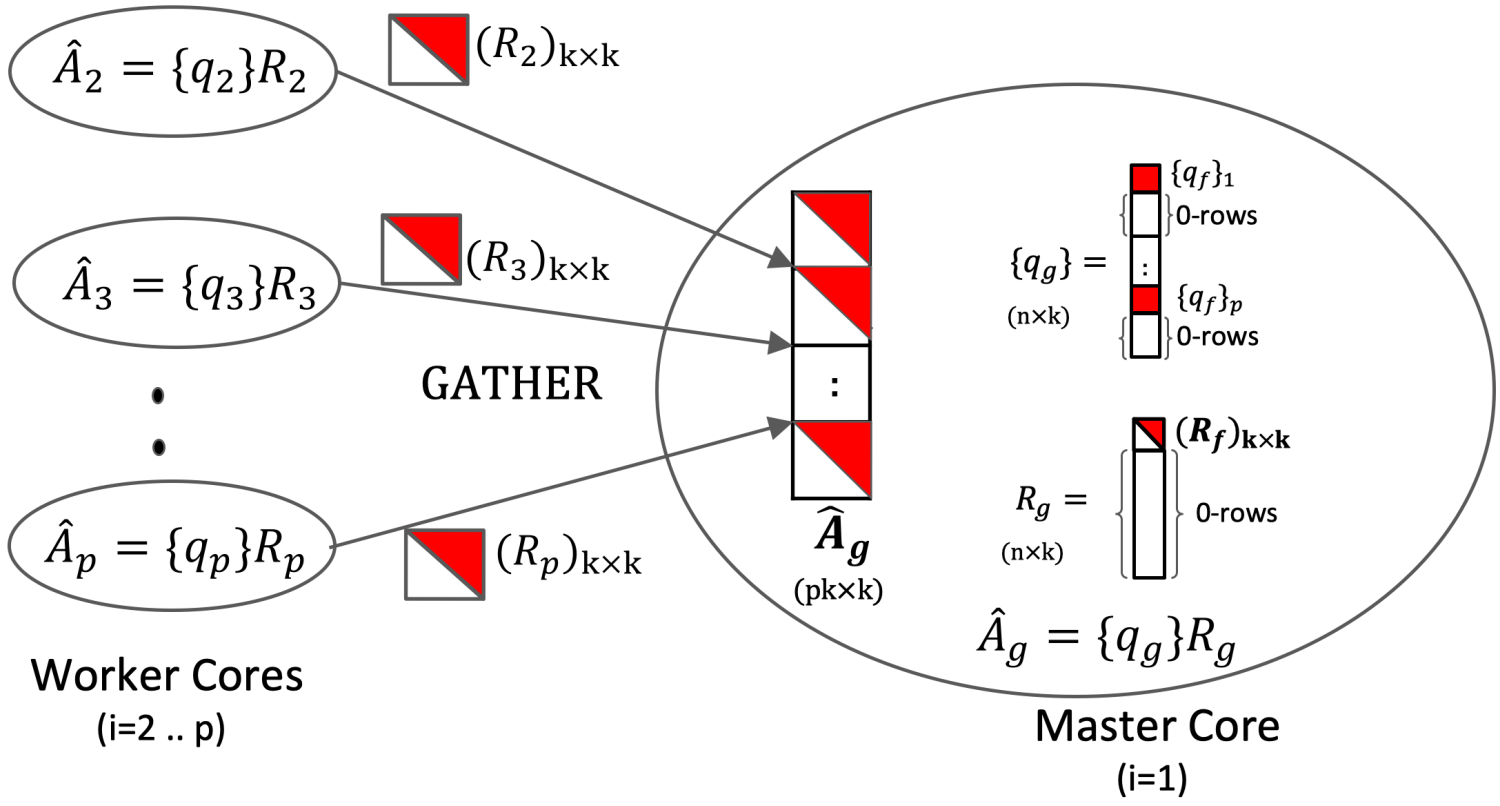


Figure: Two-level implementation of Stage 1. $\boldsymbol{Q_i}$ and $\boldsymbol{Q_g}$ stored as sets of their Householder reflectors, denoted as $\{\boldsymbol{q_i}\}$ and $\{\boldsymbol{q_g}\}$. Computational time complexity at each worker is $O(\frac{nk^2}{p})$ and at master is $O(pk^3)$

Stage 2: Parallel Dual Ascent

Define, $\boldsymbol{F} = -\left(\boldsymbol{R_g}\boldsymbol{R_g}^T + \frac{1}{2C}\boldsymbol{I}_n\right)$

Each core i solves a partition, $F_i \in \mathbb{R}^{\hat{n}\times\hat{n}}$

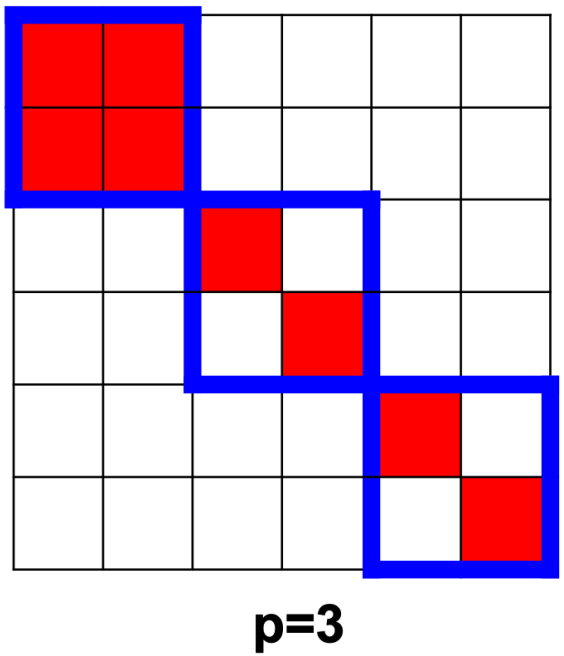


Figure: Block separable into F_i where $\hat{n} = \frac{n}{p} = \frac{6}{3} = 2$

Stage 2: Parallel Dual Ascent

Step 1: Minimization of Lagrangian - In Parallel

At core $i = \{1, \ldots, p\}$

$$\hat{\boldsymbol{\alpha}}_{\boldsymbol{i}}{}^{t+1} = \boldsymbol{F_i}^{-1}(-\hat{\boldsymbol{\beta}}_{\boldsymbol{i}}{}^{t} + \hat{\boldsymbol{e}}_{\boldsymbol{i}})$$

where,

$$\boldsymbol{F_i}^{-1} = \begin{cases} \boldsymbol{F_1}^{-1} & \text{if} \quad i = 1 \\ -2C & \text{if} \quad i = 2, \ldots, p \end{cases}$$

Step 2: Dual variable update - In Parallel

At core i

$$\hat{\boldsymbol{\beta}}_{\boldsymbol{i}}{}^{t+1} = \hat{\boldsymbol{\beta}}_{\boldsymbol{i}}{}^{t} + \eta^{\star}(-\hat{\boldsymbol{\alpha}}_{\boldsymbol{i}}{}^{t+1})$$

where, $\eta^{\star}$ is the optimal step size, and $\hat{\boldsymbol{\beta}}^{t} = \boldsymbol{Q}^{T}\boldsymbol{\beta}^{t}$

Component-wise Thresholding of $\beta_i = \max\{0, \beta_i\}$

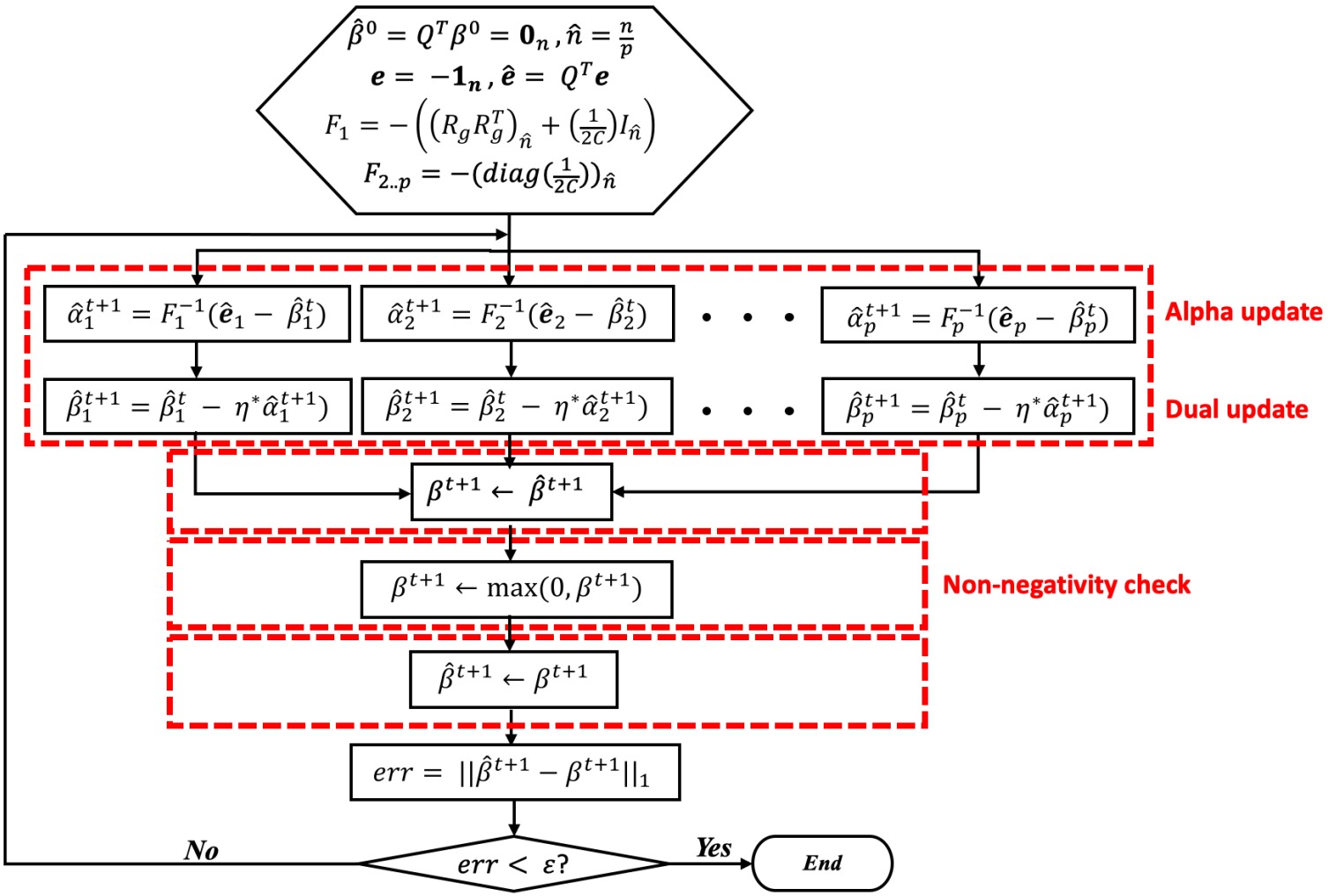
Recall, $\boldsymbol{Q} = \text{diag}(\boldsymbol{Q_1}, \boldsymbol{Q_2}, ..\boldsymbol{Q_i}, .., \boldsymbol{Q_p}) \times \boldsymbol{Q_g}$

Pseudocode

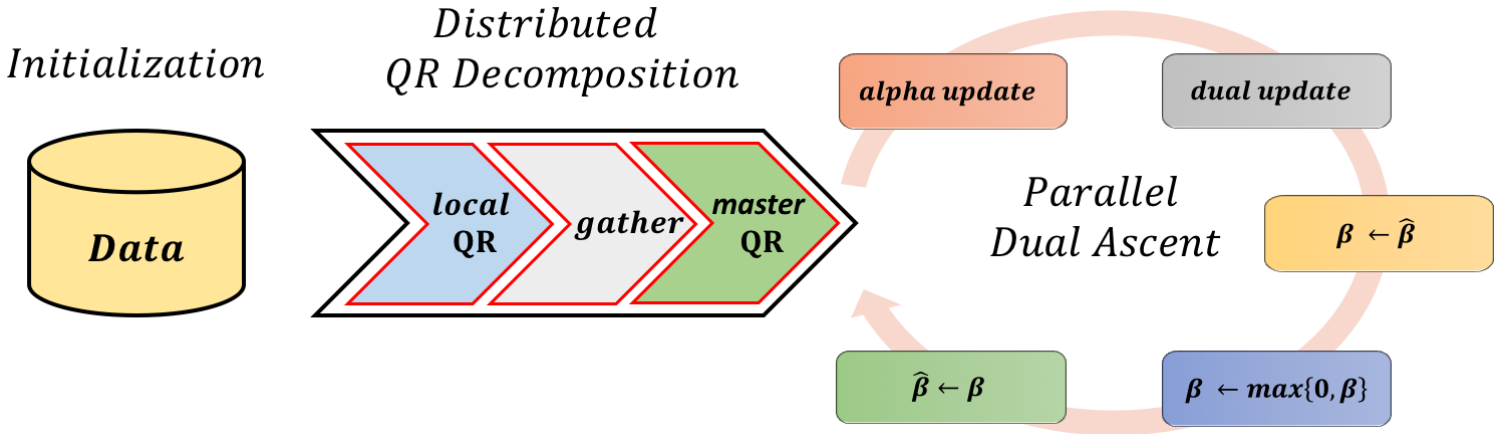
During every iteration

1. $\beta \leftarrow \hat{\beta}$ using $\boldsymbol{Q}\hat{\beta}$
   - Gather $\hat{\beta}_i$ at Master core
   - Compute $(\boldsymbol{Q}_g\hat{\beta})$ at Master core
   - Scatter $(\boldsymbol{Q}_g\hat{\beta})_i$ to each core i
   - Compute $\beta_{\boldsymbol{i}} = \boldsymbol{Q}_i \times (\boldsymbol{Q}_g\hat{\beta})_i$ at each core i
2. $\beta_i = \max(0, \beta_i)$ at each core i
3. $\hat{\beta} \leftarrow \beta$ using $\boldsymbol{Q}^T\beta$
   - Compute $(\boldsymbol{Q}_i^T\beta_i)$ at each core i
   - Gather $(\boldsymbol{Q}_i^T\beta_i)$ at Master core
   - Compute $\hat{\beta} = \boldsymbol{Q}_g^T \times \text{diag}(\boldsymbol{Q}_1^T, \ldots, \boldsymbol{Q}_p^T)\beta$ at Master core
   - Scatter $\hat{\beta}_i$ to each core i

Implementation of PDA and component-wise thresholding



Distributed QRSVM Workflow



Experimental Setup

Hardware

- *Ada* Supercomputing Cluster at TAMU
- Intel Xeon E5-2670 v2 (Ivy Bridge-EP), 10-core, 2.5GHz
- 64 GB/node and 16 cores/node
- Message-Passing Interface (MPI), InfiniBand interconnect

Dataset	n	d	**Description**
a9a	32560	123	predict annual income
covtype	464810	54	predict forest cover type

Convergence

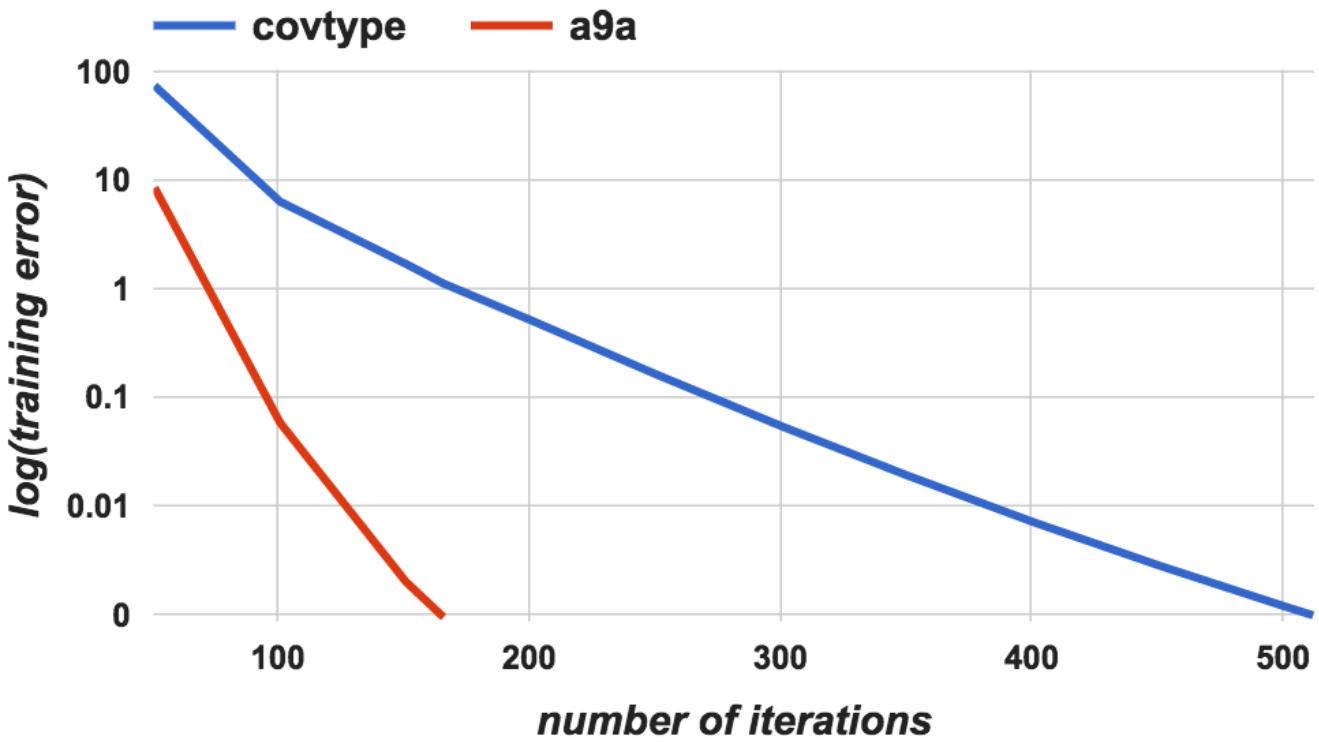


Figure: a9a: t=166 , covtype: t=512, threshold=10^{-3}

Scalability of QRSVM: $O(nk^2)$

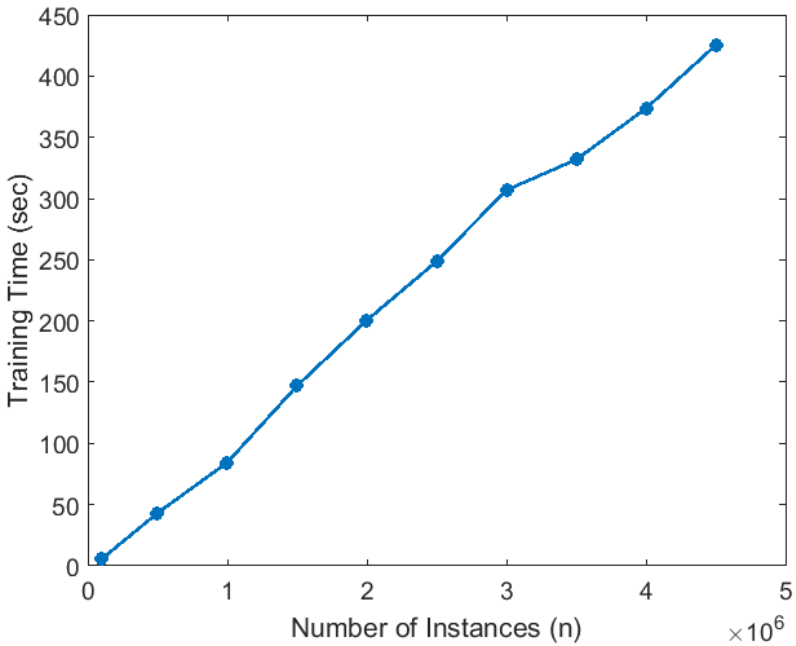


Figure: Scales linearly with n

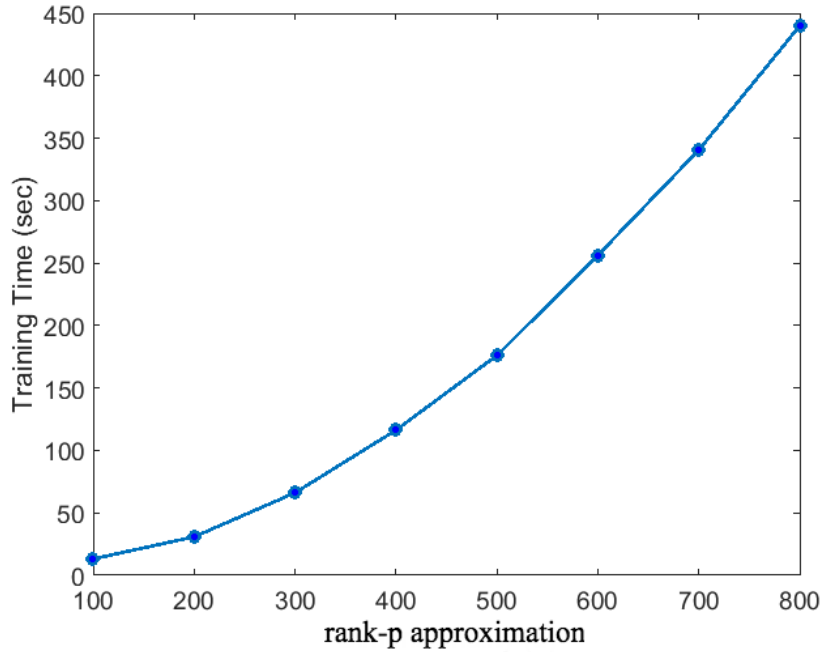


Figure: Scales quadratically with rank k

Optimal Step Size, η^*

Figure: a9a, $\eta^* = 1.9$

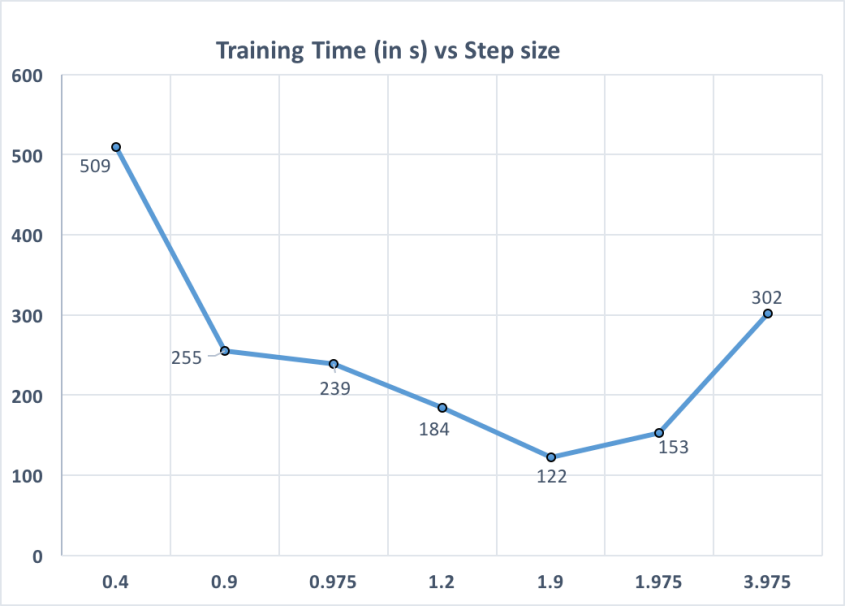


Figure: covType, $\eta^* = 1.9$

Distributed QRSVM: Timing Discussions for $p = 16$

Stage 1: Distributed QR

1. Computation: $T(p_{localQR}) + T(p_{masterQR})$
2. Communication: $T(c_{gatherR})$

Stage2: Parallel Dual Ascent

1. Computation: $T(p_{pda})$
2. Communication: $T(c_{pda})$ Gather+Scatter

Time details	**a9a (in ms)**	**covtype (in s)**
$T(p_{meka})$	460	2.1
$T(p_{localQR})$	24	1.89
$T(p_{masterQR})$	4	0.02
$T(c_{gatherR})$	0.5	0.04
$T(p_{pda})$	1628.1	120.18
$T(c_{pda})$	17.1	0.36
$T(train)$	1674.2	122.50

Comparison with PSVM and P-packSVM ($p = 16$)

PSVM [Chang, 2007], and **P-packSVM** [Zeyuan, 2009]

Dataset	dis-QRSVM	PSVM	P-packSVM
covType	2 min	20 min	16 min

Demerits of PSVM and P-packSVM

- PSVM uses Incomplete Cholesky Factorization (ICF) $\Rightarrow$ Difficult to parallelize and slow $\Rightarrow$ Unfit for distributed big data analytics
- PSVM training time is $O(n^2/p)$/iteration $\Rightarrow$ Limited scalability
- P-packSVM solves *primal* form $\Rightarrow$ Slow Convergence

Work so far

Relaxed synchronization approach for solving parallel QP problems

[IPDPS'16] K. Lee, R. Bhattacharya, J. Dass, V. N. S. P. Sakuru, R. N. Mahapatra, "A Relaxed Synchronization Approach for Solving Parallel Quadratic Programming Problems with Guaranteed Convergence,"

Parallel and memory-efficient training of kernel SVM

[ICDCS'17] J. Dass, V. N. S. P. Sakuru, V. Sarin and R. N. Mahapatra, "Distributed QR Decomposition Framework for Training Support Vector Machines",

Recall, in distributed QRSVM

- Memory-efficient modeling of SVM
- Specifically, $\boldsymbol{Q}$ as set of Householder reflector $\{\boldsymbol{q}\}$
- Also, using $(\boldsymbol{R})_{k\times k}$

Recall, in distributed QRSVM

- Memory-efficient modeling of SVM
- Specifically, $\boldsymbol{Q}$ as set of Householder reflector $\{\boldsymbol{q}\}$
- Also, using $(\boldsymbol{R})_{k\times k}$

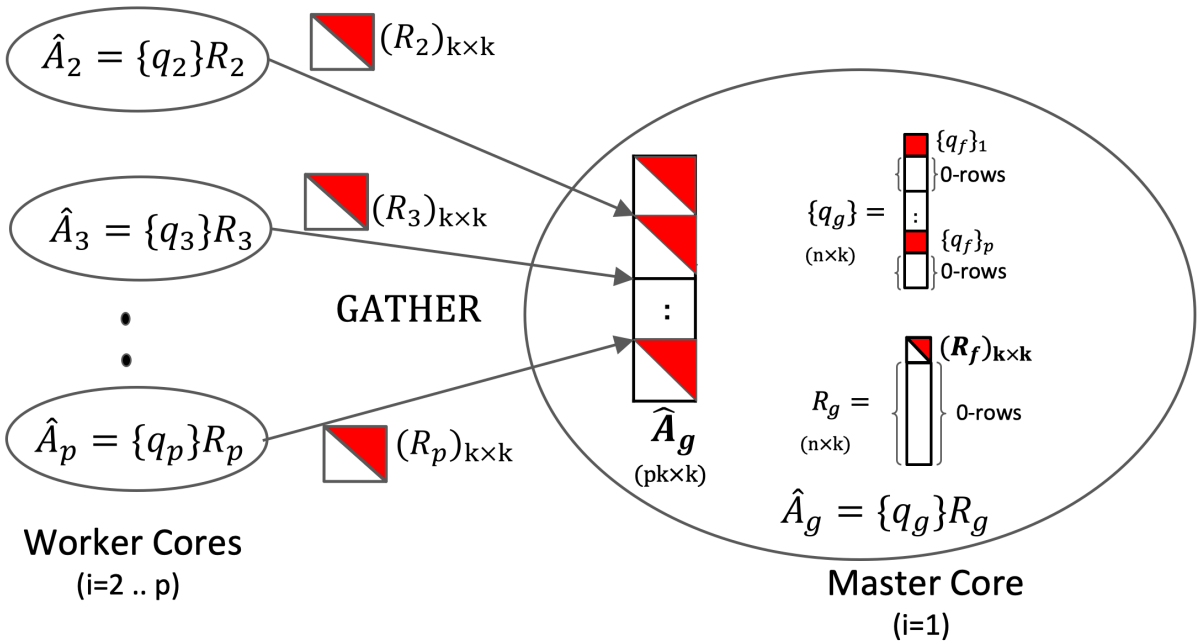
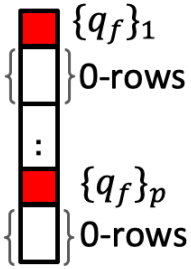


Figure: Stage 1: $\boldsymbol{Q_i}$ and $\boldsymbol{Q_g}$ stored as $\{\boldsymbol{q_i}\}$ and $\{\boldsymbol{q_g}\}$

Observe,

$\{q_g\}_{(n\times k)} =$ 

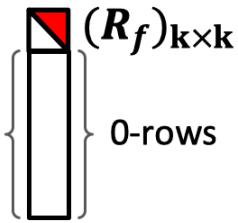
$R_{g\ (n\times k)} =$ 

Figure: At Master core

Also recall, $\boldsymbol{Q} = \text{diag}(\boldsymbol{Q_1}, \boldsymbol{Q_2}, ..\boldsymbol{Q_i}, .., \boldsymbol{Q_p}) \times \boldsymbol{Q_g}$

Observe,

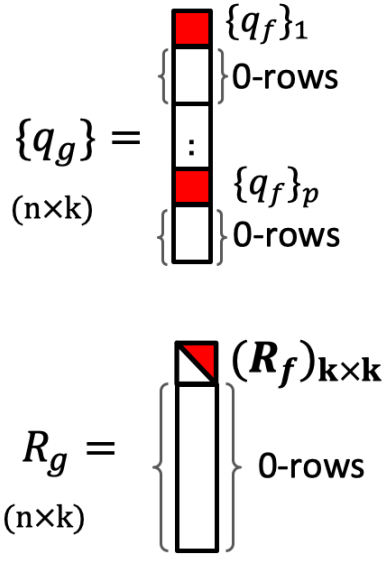


Figure: At Master core

Recall,

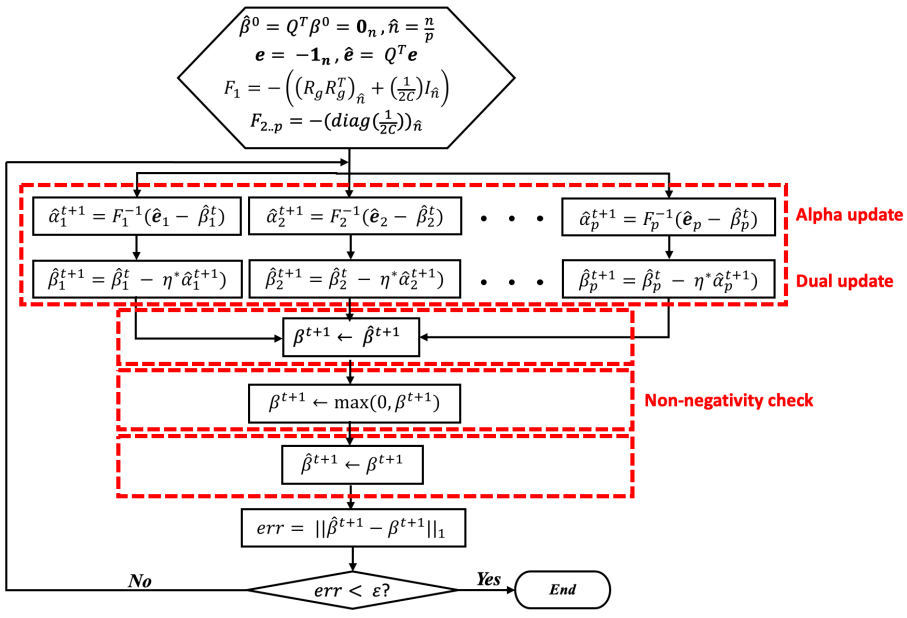


Figure: Parallel Dual Ascent

Also recall, $\boldsymbol{Q} = \text{diag}(\boldsymbol{Q_1}, \boldsymbol{Q_2}, ..\boldsymbol{Q_i}, .., \boldsymbol{Q_p}) \times \boldsymbol{Q_g}$

Observe,

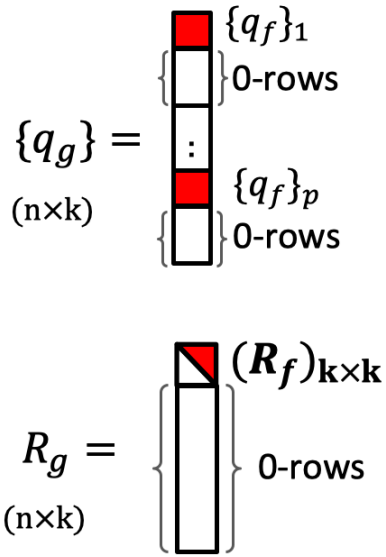


Figure: At Master core

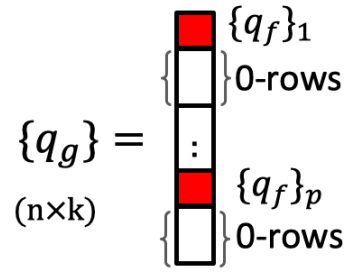
$\beta = Q\hat{\beta}$ using $\{q_g\}$

- Gather $\hat{\beta}_i$ at Master core
- Scatter $(Q_g\hat{\beta})_i$ to each core i

$\hat{\beta} = Q^T\beta$ using $\{q_g\}$

- Gather $(Q_i^T\beta_i)$ at Master core
- Scatter $\hat{\beta}_i$ to each core i

Observe,



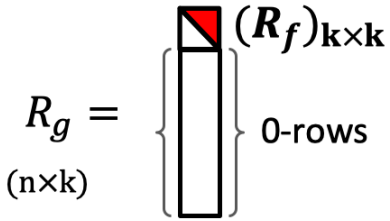


Figure: At Master core

$\beta = Q\hat{\beta}$ using $\{q_g\}$

- Gather $\hat{\beta}_i$ at Master core
- Scatter $(Q_g\hat{\beta})_i$ to each core i

$\hat{\beta} = Q^T\beta$ using $\{q_g\}$

- Gather $(Q_i^T\beta_i)$ at Master core
- Scatter $\hat{\beta}_i$ to each core i

Communication Overhead in distributed QRSVM

During every iteration of parallel dual ascent, each Gather and Scatter involves communicating **huge** $O(\frac{n}{p})$ data volume per core with Master core $\Rightarrow$ Poor Scalability

Now, let us create a

Communication-efficient framework for scaling distributed SVM training

Compact Representation

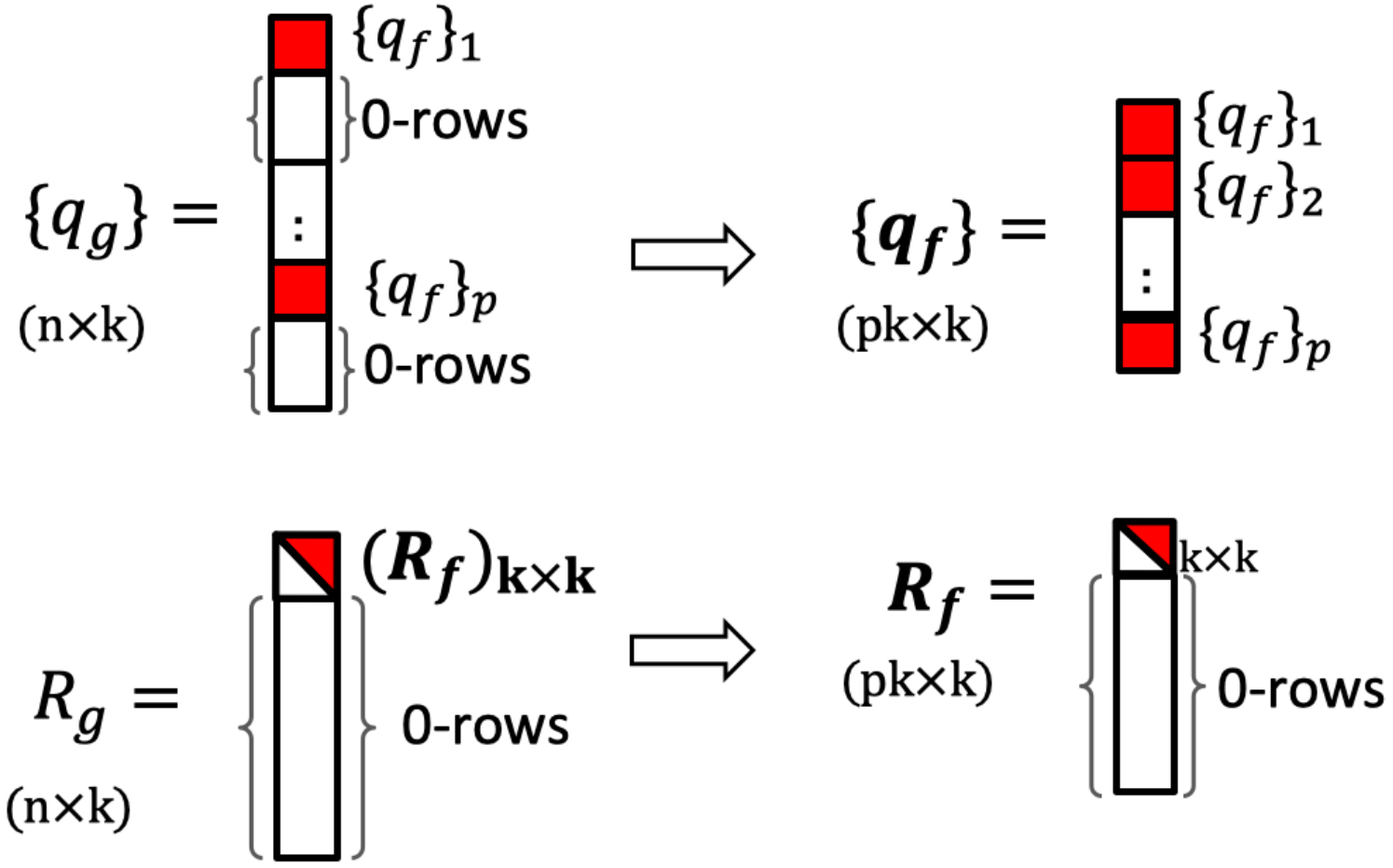


Figure: At Master core

Compact Representation

Note, $\{q_f\} \in \mathbb{R}^{pk \times k}$ and $R_f \in \mathbb{R}^{pk \times k}$ are obtained directly from QR decomposition of gathered R_i i.e. $\hat{A}_g$ in Stage 1

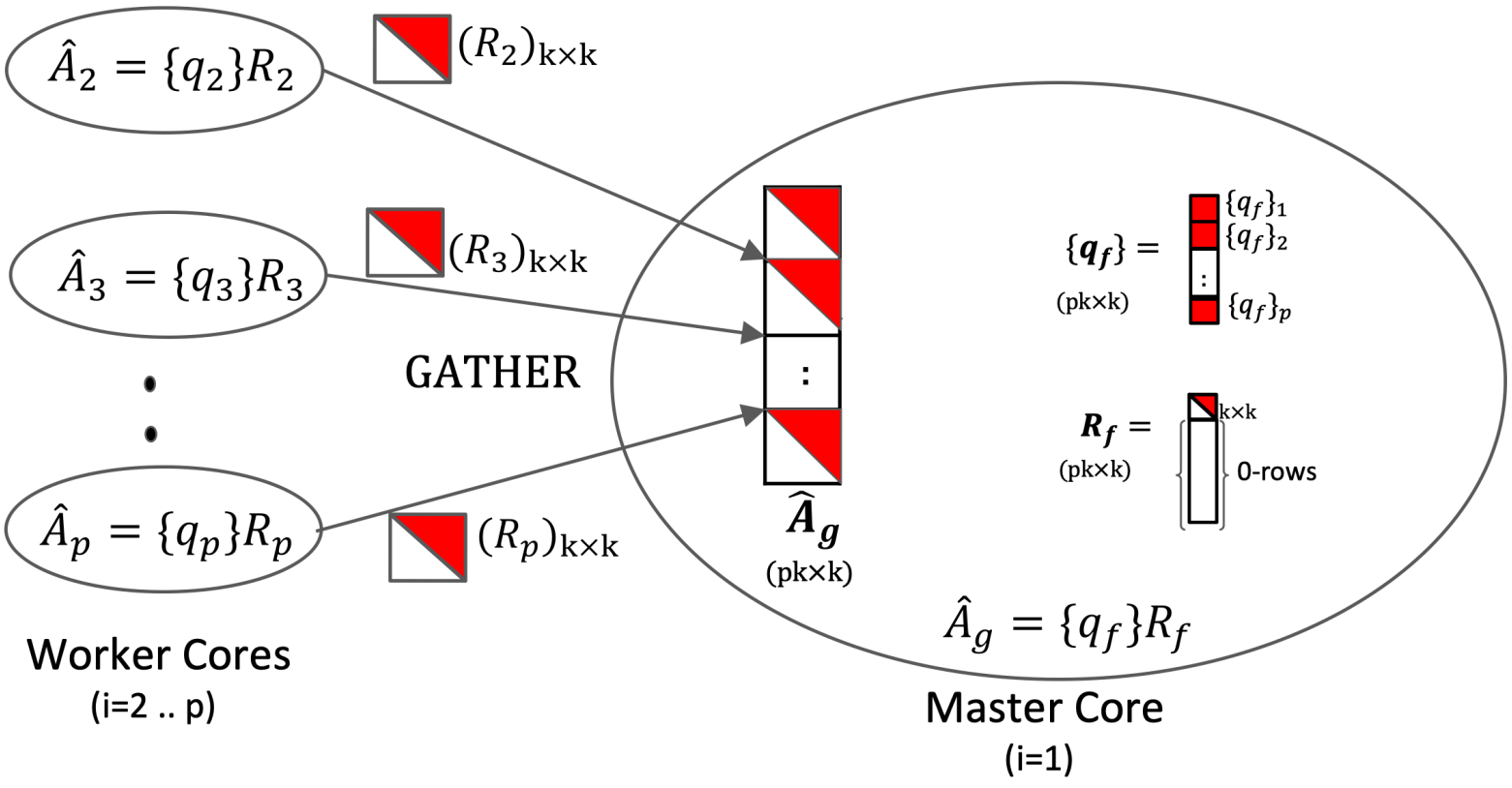


Figure: Stage 1: Q_i and Q_f stored as $\{q_i\}$ and $\{q_f\}$

Earlier,

Communication Overhead

- During every iteration of parallel dual ascent, each Gather and Scatter involves communicating **huge** $O(\frac{n}{p})$ data volume per core with Master core
- Does not scale to larger data sizes (n)

Communication-Efficiency

Now,

Communication Efficient $\Rightarrow$ High Scalability

- During every iteration of parallel dual ascent, each Gather and Scatter involves communicating **negligibly small** $O(k)$ data volume per core with Master core, where, $k \ll \frac{n}{p}$
- Scalable with number of cores (p) and handle larger datasets than previous implementation

Experimental Setup

TABLE 1: Benchmark Dataset Description

Dataset	**#training samples (n)**	**#features (d)**	***k*-rank approx.**
covtype.binary	464,810	54	64
webspam(unigram)	350,000	254	128
SUSY	5,000,000	18	128

TABLE 2: Distributed-QRSVM

Parameters	**covtype**	**webspam**	**SUSY**
C	1	1	1
γ	2^3	2^0	2^{-3}
#cores, p	16	32	64
stopping threshold	10^{-3}	10^{-3}	10^{-3}
optimal step size, η^*	0.9	0.9	0.9
#iterations, t_c	1075	569	2096

Convergence

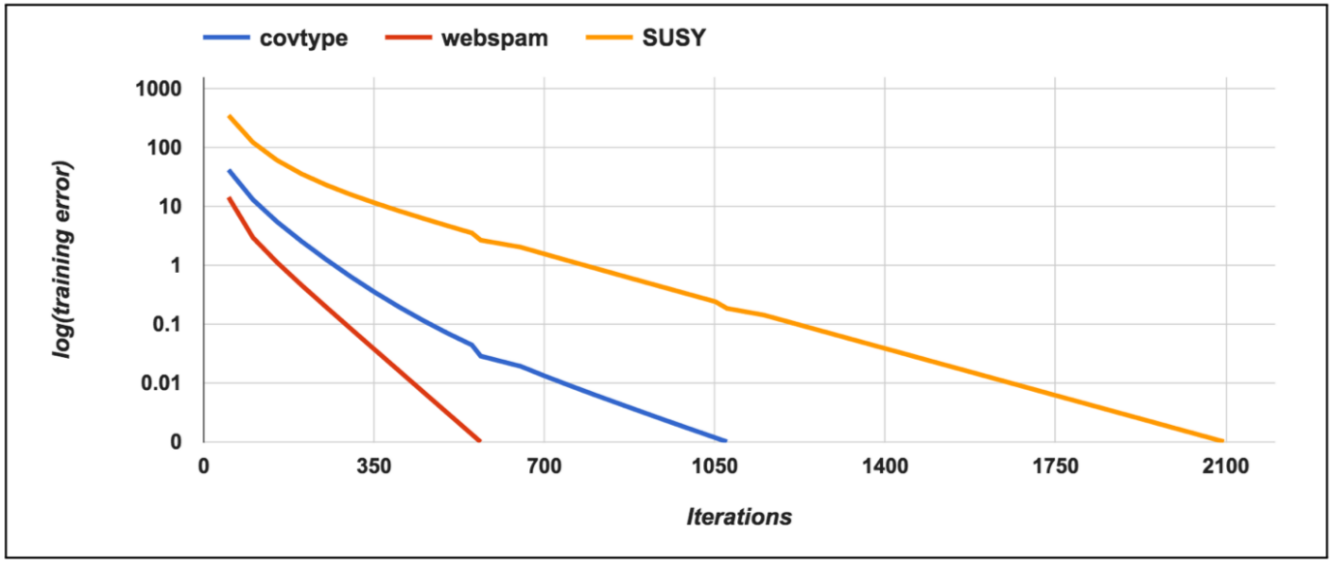


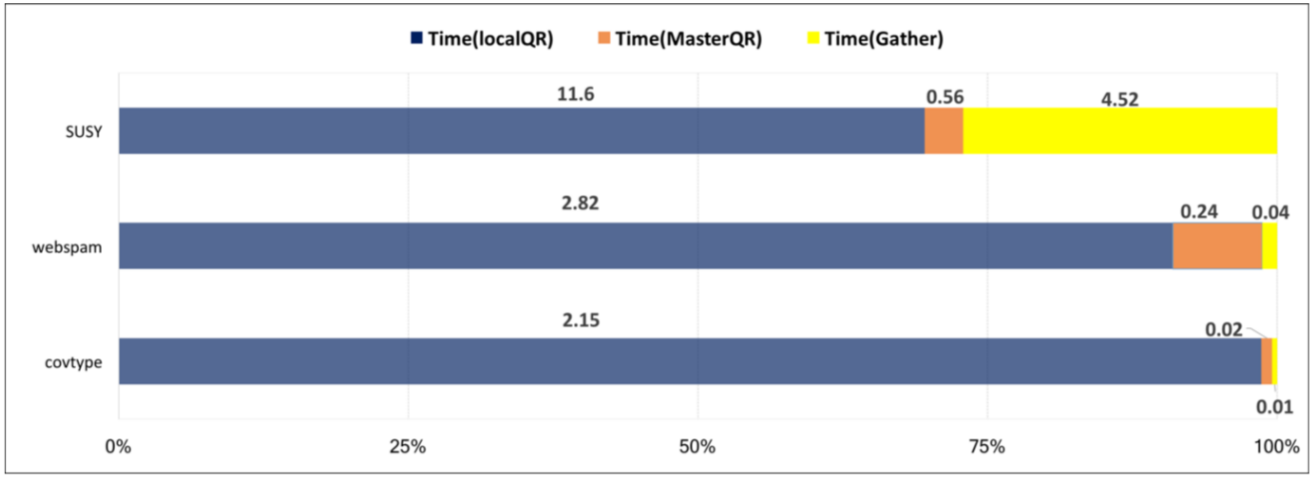
Figure: Convergence rate for covtype, webspam, and SUSY for stopping threshold 10^{-3}. $t_c = 1075$, $t_c = 569$, and $t_c = 2096$ iterations, respectively, using the optimal step size $\eta^\star = 0.9$

Scalability

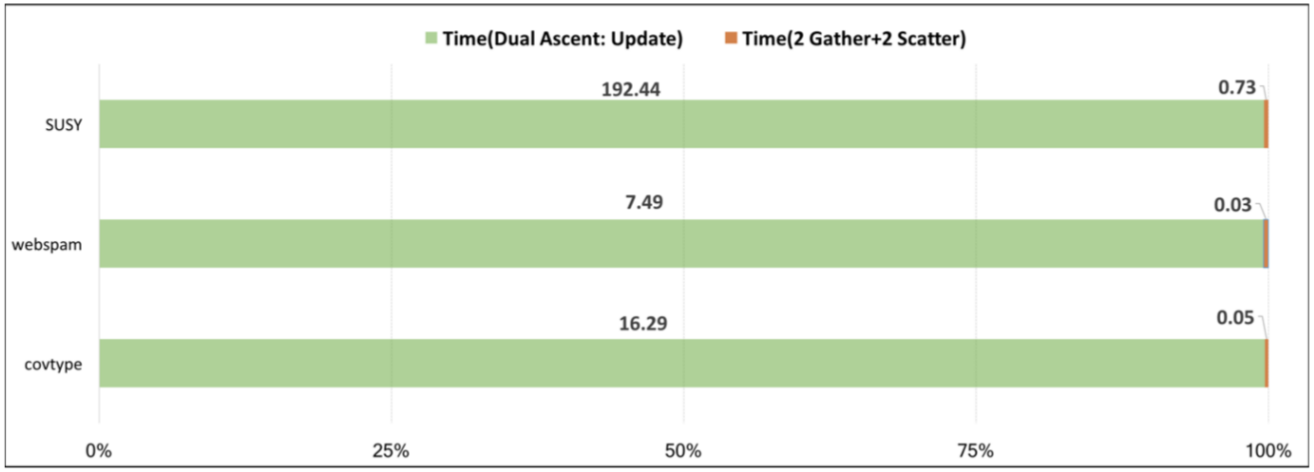
TABLE 3: Scalability of distributed-QRSVM.
Training time (in seconds) and Speedup, S_p wrt sequential-QRSVM (S_p is shown in parenthesis).

Dataset	sequential	p=2	p=4	p=8	p=16	p=32	p=64
covtype	268 (1x)	132 (2x)	64 (4x)	33 (8x)	18 (15x)	10 (27x)	6 (45x)
webspam	258 (1x)	120 (2x)	58 (4x)	32 (8x)	19 (14x)	11 (23x)	9 (29x)
SUSY	28,614 (1x)	11,284 (2x)	4,405 (7x)	1,686 (17x)	804 (36x)	380 (75x)	210 (136x)

Training Time (in seconds) analysis



(a) Stage 1: Distributed QR decomposition



(b) Stage 2: Parallel Dual ascent

Training Time (in seconds) analysis

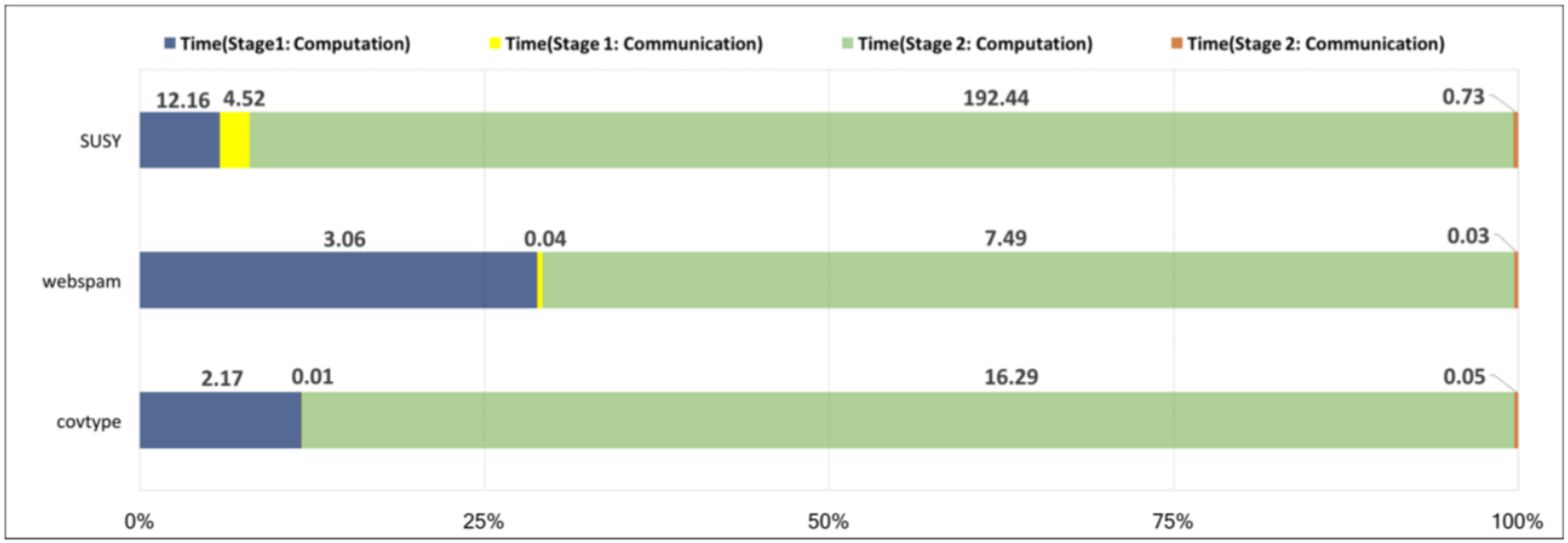


Figure: Overall training time. A majority of the training time is spent in iterative computations in Stage 2 (parallel dual ascent), while communication overhead is negligible.

Comparison with PSVM, and P-packSVM, and QRSVM[ICDCS'17]

TABLE 4: Comparing *dis*-QRSVM with PSVM, P-packSVM and [12] on T_{train} (in seconds) for *covtype* dataset.

Algorithm	p=2	p=4	p=8	p=16	p=32	p=64
PSVM	8,562	4,396	2,352	1,270	635	341
P-packSVM	-	-	2,019	1,022	295	110
dis-QRSVM [12]	390	309	271	261	256	454
dis-QRSVM	**132**	**64**	**33**	**18**	**10**	**6**

- data unavailable

TABLE 5: Comparing proposed *dis*-QRSVM with [12] on Stage 2 computation time, T_{update} and communication time, T_{2g+2s} (in seconds) for *covtype* dataset.

Time	p=2	p=4	p=8	p=16	p=32	p=64
T_{update} [12]	379	304	269	259	252	448
T_{update}(proposed)	**122**	**59**	**30**	**16**	**8**	**5**
T_{2g+2s} [12]	1.63	1.81	0.34	0.05	1.19	3.02
T_{2g+2s}(proposed)	**0.02**	**0.02**	**0.14**	**0.05**	**0.03**	**0.11**

- dis-QRSVM trains upto 70×, upto 60×, and upto 75× faster than PSVM, P-packSVM, and earlier [ICDCS'17] implementation, respectively
- dis-QRSM is communication-efficient and scales better on PDA with $p = \{2, 4, 8, 16, 32, 64\}$ than [ICDCS'17]
- dis-QRSVM can handle larger workloads than [ICDCS'17]

Work so far

Relaxed synchronization approach for solving parallel QP problems

[IPDPS'16] K. Lee, R. Bhattacharya, J. Dass, V. N. S. P. Sakuru, R. N. Mahapatra, "A Relaxed Synchronization Approach for Solving Parallel Quadratic Programming Problems with Guaranteed Convergence,"

Parallel and memory-efficient training of kernel SVM

[ICDCS'17] J. Dass, V. N. S. P. Sakuru, V. Sarin and R. N. Mahapatra, "Distributed QR Decomposition Framework for Training Support Vector Machines",

Communication-efficient framework for scaling distributed SVM training

[TPDS'18] J. Dass, V. Sarin and R. N. Mahapatra "Fast and Communication-Efficient Algorithm for Distributed Support Vector Machine Training," [Dass, et al 2018]

Work in Progress

WiP Outline

- Multi-FPGA based framework for Energy-efficient training of SVM
- Distributed framework for Incremental Learning in Kernel Ridge Regression
- Distributed training for Deep Learning on low-power and memory-constrained end devices

Multi-FPGA based framework for Energy-efficient training of SVM

- Edge devices are getting efficient in processing data
- Make them capable for accelerating ML training
- Help reduce latency for critical applications
- Lower the dependency on HPC server grade CPUs, GPUs
- Provide energy-efficient modeling and training
- First-of-its-kind work for training SVM in a multiple FPGA environment

Multi-FPGA based framework for Energy-efficient training of SVM

Motivation

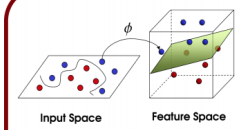
- Training non-linear Support Vector Machine model
  Computationally Expensive + Requires High Memory
- FPGA-based SVM training accelerators exist, but, for
  Sequential algorithms like SMO using single FPGA
- Data is generally **Generated in a distributed** manner at edge

Contribution

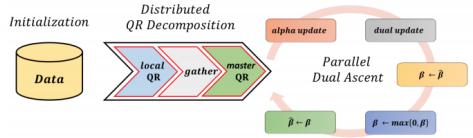
Train SVM model *in a* ***Distributed*** *manner using* ***Multiple FPGA network*** *while achieving*

- ✓ *Faster training time*
- ✓ *Memory-efficient data representations*
- ✓ *Negligible network communication overhead*
- ✓ *Linear Scalability with #FPGA units*
- ✓ *High Energy efficiency*

Algorithmic Design

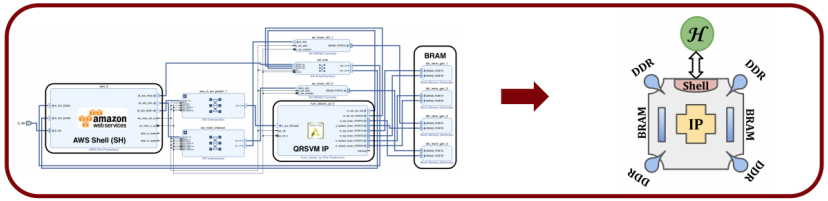


SVM learning



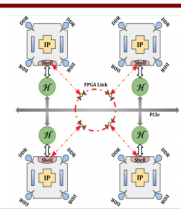
Process Flow for Distributed SVM algorithm (QRSVM)

A single QRSVM IP for FPGA



Multiple FPGA Network

Proof-of-Concept
Amazon AWS F1 instance with p={1,2,4,8}
16nm Xilinx Virtex Ultrascale+ VU9P FPGA units



Distributed framework for Incremental Learning in Kernel Ridge Regression

- KRR combines the ridge regression with kernel method for capturing non-linear relationships between predictor variables and responses.
- Addresses ill-posedness and overfitting in the ordinary least squares via L2 regularization
- DC-KRR [Zhang 2015] is SOTA solver for decomposing KRR into sub-problems and solving for independent KRR estimates. Suffers with $O(\frac{N^3}{p^2})$ in memory and $O(\frac{N^2}{p^2})$ in computational complexity.
- Propose: **incKRR** for distributed KRR to handle streaming data at edge via Incremental learning

Distributed framework for Incremental Learning in Kernel Ridge Regression

$$\hat{f} = \underset{f \in H}{\operatorname{argmin}} \left\{ \frac{1}{N} \sum_{i=1}^{N} (y_i - f(\boldsymbol{x_i})^2 + \lambda \|f\|_H \right\}$$

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{N} \beta_i \kappa(\boldsymbol{x_i}, \boldsymbol{x})$$

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{N} (\boldsymbol{Y} - \boldsymbol{\beta^T K})^T (\boldsymbol{Y} - \boldsymbol{\beta^T K}) + \lambda \boldsymbol{\beta^T K \beta} \right\},$$

where, $\boldsymbol{Y} = (y_1, \ldots, y_N)^T$, and $\boldsymbol{K} = [\kappa(\boldsymbol{x_i}, \boldsymbol{x_j})]_{1 \le i,j \le N}$ is kernel matrix.

$$(\lambda N \boldsymbol{I} + \boldsymbol{K})\boldsymbol{\beta} = \boldsymbol{Y}$$

$$(\lambda N \boldsymbol{I} + \boldsymbol{R R^T})\hat{\boldsymbol{\beta}} = \hat{\boldsymbol{Y}}$$

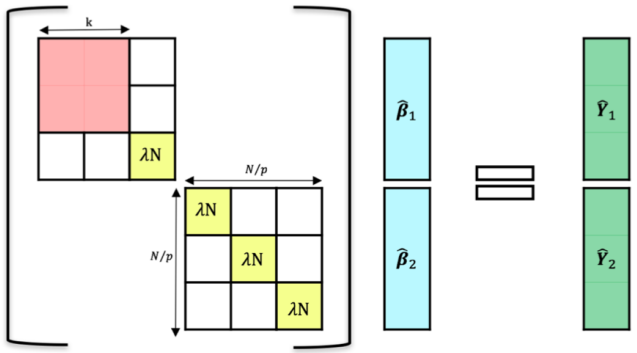


Figure 2: Partitioning (7) into $p = 2$ processes such that $k \ll \frac{N}{p}$. Uncolored cells represent zeros.

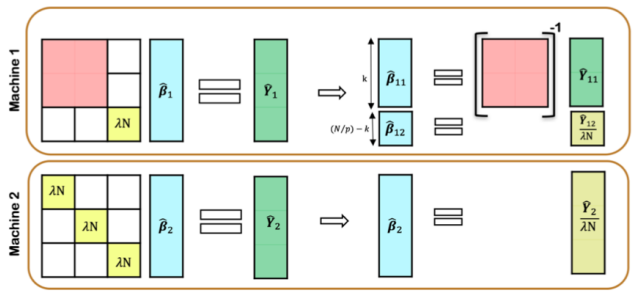


Figure 3: Solving (7) in $p = 2$ machines in parallel. For processes $i = 2, \ldots, p$, the computation is exactly similar to that of machine 2. Uncolored cells represent zeros.

Distributed training for Deep Learning on low-power and memory-constrained Edge devices

- DL delivers state-of-the-art accuracy on many AI tasks
- Training DL models is computationally heavy with learning many parameters over large # iterations
- Hence, pre-trained weights on standard datasets like CIFAR or ImageNet are used irrespective of the target application
- Accelerate DL training for Edge-based decentralized networks via quantization and sparsification of weights, gradients
- Create energy-efficient computing models and hardware designs for training with low latency and high throughput for resource-constrained devices
- Enable wide deployment of DL particularly for embedded applications such as mobile, Internet of Things (IOT),and drones

Concurrency in Training

- Model Consistency (sync, async, stale-sync)
- Parameter/Gradient Distribution (centralized PS, decentralized etc)
- Parameter/Gradient Compression (quantized, sparse) for Communication
- Model Consolidation (Averaging, Ensemble learning)

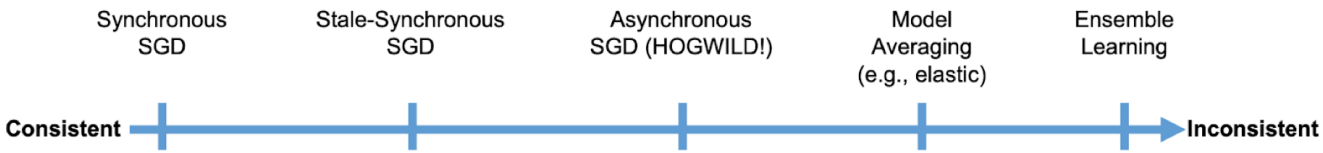
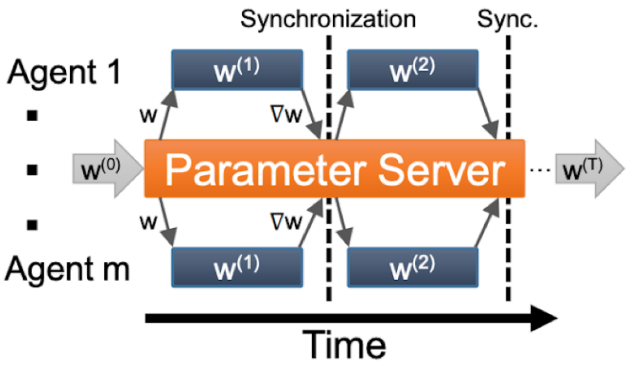
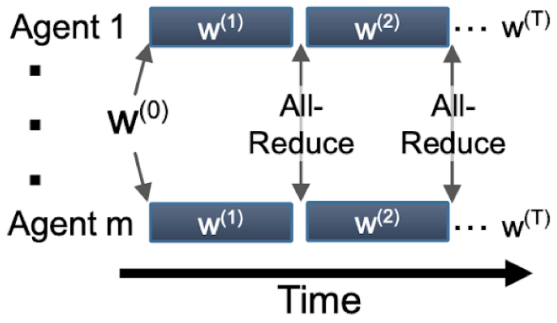


Figure: Parameter consistency spectrum

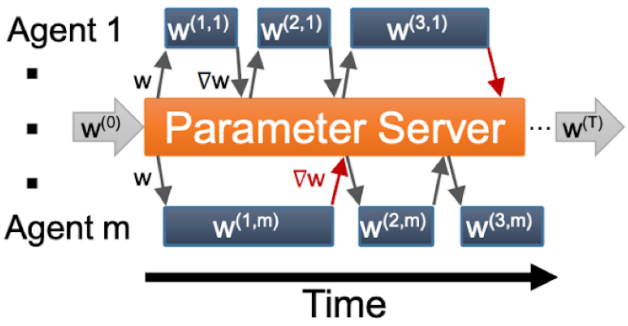
Model Consistency + Parameter/Gradient Distribution



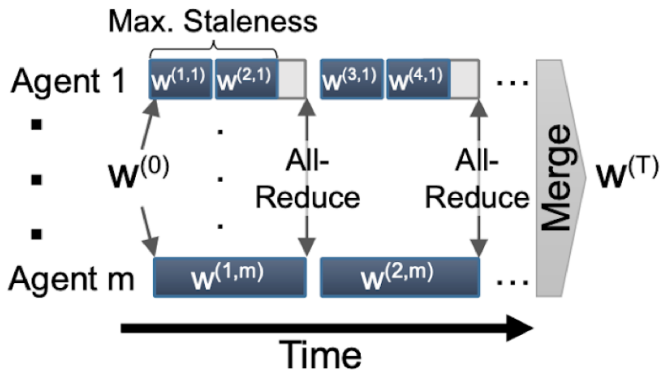
(a) Synchronous, Parameter Server



(b) Synchronous, Decentralized



(c) Asynchronous, Parameter Server



(d) Stale-Synchronous, Decentralized

source: Ben-Nun, Tal, and Torsten Hoefler. "Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis." arXiv preprint arXiv:1802.09941 (2018).

Gradient Compression for Efficient-Communication

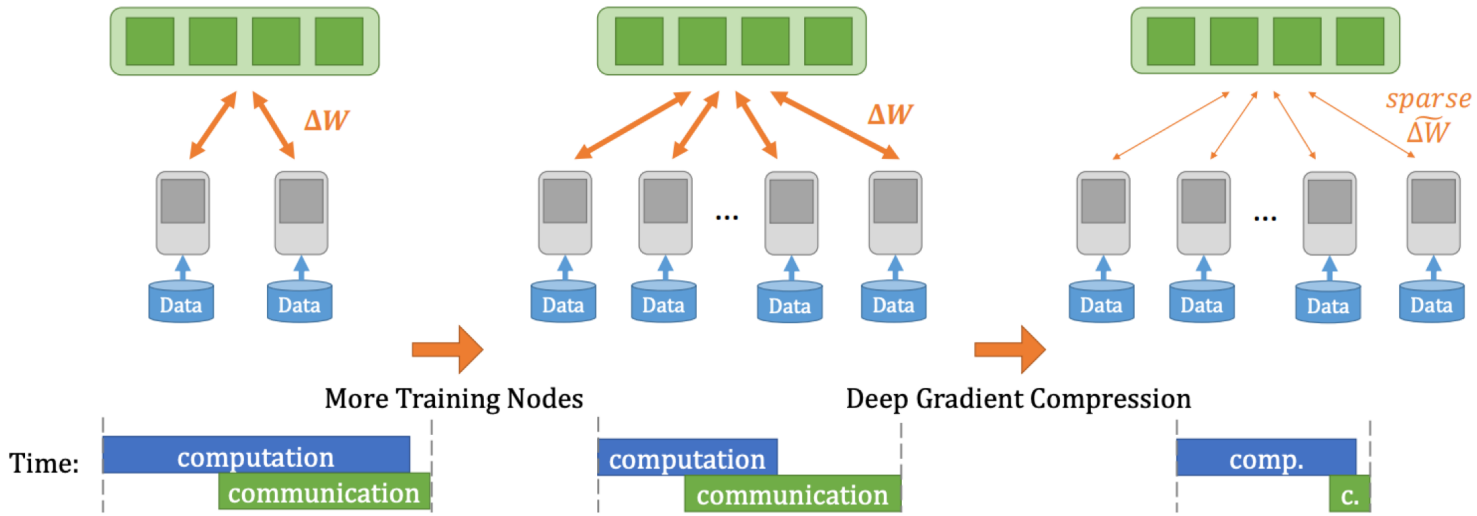


Figure: Gradient compression can reduce communication time, improve scalability, and speed up distributed training for Deep Learning models

source: Lin, Yujun, Song Han, Huizi Mao, Yu Wang, and William J. Dally. "Deep gradient compression: Reducing the communication bandwidth for distributed training." arXiv preprint arXiv:1712.01887 (2017)

Conclusions

Conclusions

The scope of the proposal is to motivate the need for a decentralized and distributed framework for large-scale ML that can be achieved by

Making training algorithms

- CHEAP in memory, computational cost, and energy consumption to run on edge devices (edge servers, if needed)
- PARALLEL to be solved among multiple edge devices (edge servers, if needed)
- SCALABLE across number of edge devices to handle bigdata
- Amenable for mapping into efficient hardware ACCELERATOR design

Thank You!

References

IPM

Sanjay Mehrotra, "On the Implementation of a Primal-Dual Interior Point Method," SIAM Journal on Optimization, vol. 2, no. 4, pp. 575-601, November 1992.

PSVM

E. Y. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui, "PSVM: Parallelizing support vector machines on distributed computers," in NIPS, 2007.

SMO

Platt, John. "Sequential minimal optimization: A fast algorithm for training support vector machines." (1998).

LIBSVM

Chih-Chung Chang and Chih-Jen Lin. (2001) "LIBSVM: a library for support vector machines". `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

SVM^{light}

Thorsten Joachims, "Learning to Classify Text Using Support Vector Machines", Kluwer Academic Publisher, 2002.

Parallel SMO

Gaetano Zanghirati and Luca Zanni, "A parallel solver for large quadratic programs in training support vector machines," Parallel Computing, vol. 29, no. 4, April 2003.

PpackSVM

A. Z. Zeyuan, C. Weizhu, W. Gang, Z. Chenguang and C. Zheng, P-packSVM: "Parallel Primal grAdient desCent Kernel SVM," 2009 Ninth IEEE International Conference on Data Mining, Miami, FL, 2009, pp. 677-686. doi: 10.1109/ICDM.2009.29

References

MEKA

S. Si, C.-J. Hsieh, and I. S. Dhillon, "Memory efficient kernel approximation.," in ICML, pp. 701–709, 2014.

LSDA

K. Lee, R. Bhattacharya, **J. Dass**, V. N. S. P. Sakuru, and R. N. Mahapatra, "A relaxed synchronization approach for solving parallel quadratic programming problems with guaranteed convergence," in IPDPS, pp. 182191, May 2016

QRSVM

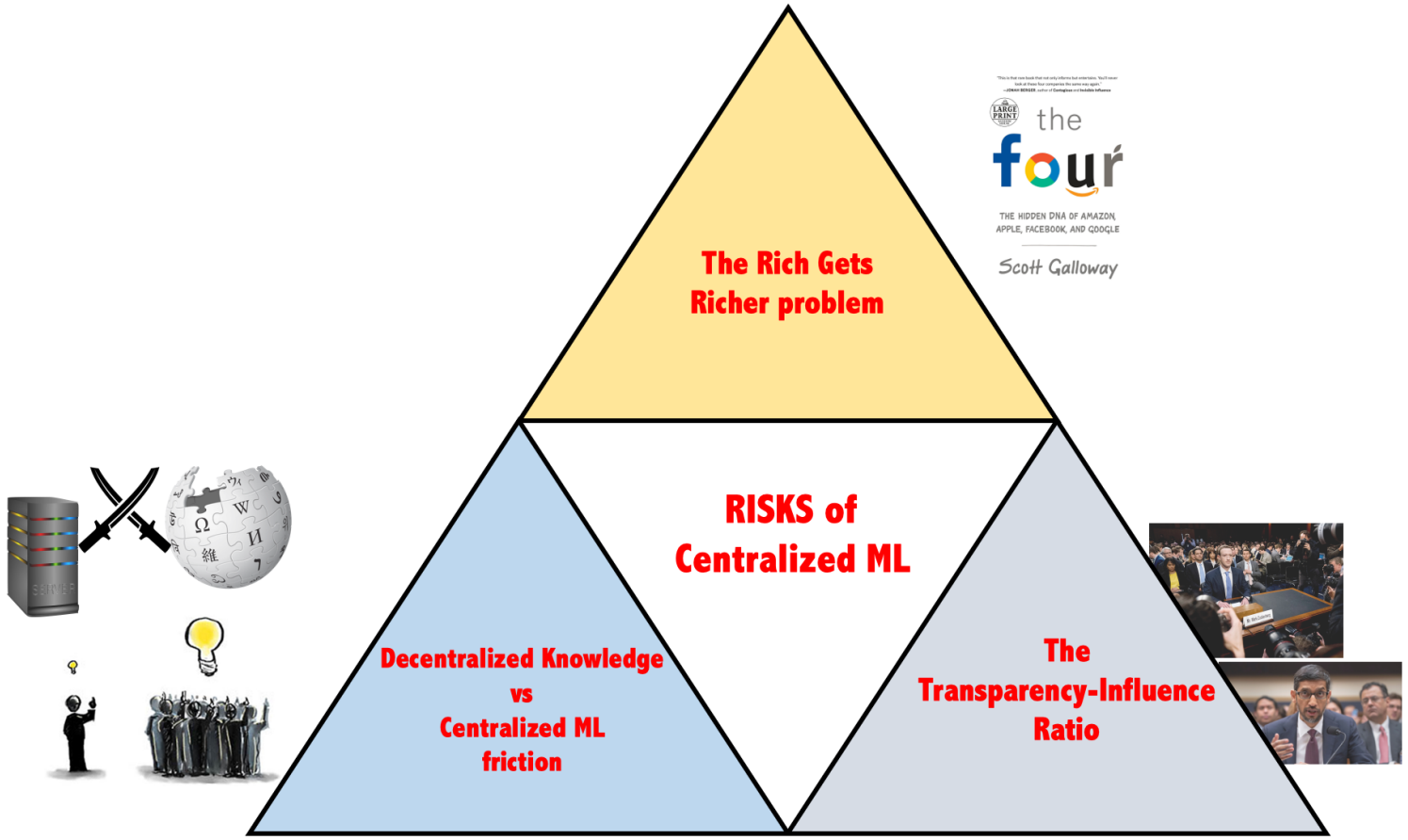
J. Dass, V. N. S. P. Sakuru, V. Sarin and R. N. Mahapatra, "Distributed QR Decomposition Framework for Training Support Vector Machines," 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS 2017), Atlanta, GA, 2017, pp. 753-763. doi: 10.1109/ICDCS.2017.222

communication efficient QRSVM

J. Dass, V. Sarin and R. N. Mahapatra "Fast and Communication-Efficient Algorithm for Distributed Support Vector Machine Training," in 2018 IEEE Transactions on Parallel and Distributed Systems (TPDS). doi: 10.1109/TPDS.2018.2879950

Appendix

Risks in a Centralized ML Framework



source:
https://towardsdatascience.com/everything-you-need-to-know-about-decentralized-ai-3abdb052324b

Stability of LSDA

From Step 1, for $t = kP$,

$$\boldsymbol{x}_i^{kP+1} = -\boldsymbol{K}_i^{-1}(\boldsymbol{A}_i^T \beta_i^{kp} + \boldsymbol{c}_i)$$

For $t = (k+1)P - 1$,

Stability of LSDA

From Step 1, for $t = kP$,

$$\boldsymbol{x}_i^{kP+1} = -\boldsymbol{K}_i^{-1}(\boldsymbol{A}_i^T \boldsymbol{\beta}_i^{kp} + \boldsymbol{c}_i)$$

For $t = (k+1)P - 1$,

LSDA as Discrete-time Dynamical system

$$\boldsymbol{\beta}^{(k+1)P} = \boldsymbol{A}(P)\boldsymbol{\beta}^{kP} + \boldsymbol{b}(P)$$

where,
$\boldsymbol{A}(P) = \left(\boldsymbol{I}_n - \eta P \sum_{i=1}^{p} \boldsymbol{A}_i \boldsymbol{K}_i^{-1} \boldsymbol{A}_i^T\right)$, $\boldsymbol{b}(P) = -\eta P \sum_{i=1}^{p} \left(\boldsymbol{A}_i \boldsymbol{K}_i^{-1} \boldsymbol{c}_i + \frac{1}{p}\boldsymbol{b}\right)$

Stability of LSDA

From Step 1, for $t = kP$,

$$\boldsymbol{x}_i^{kP+1} = -\boldsymbol{K}_i^{-1}(\boldsymbol{A}_i^T \boldsymbol{\beta}_i^{kp} + \boldsymbol{c}_i)$$

For $t = (k+1)P - 1$,

LSDA as Discrete-time Dynamical system

$$\boldsymbol{\beta}^{(k+1)P} = \boldsymbol{A}(P)\boldsymbol{\beta}^{kP} + \boldsymbol{b}(P)$$

where,
$\boldsymbol{A}(P) = \left(\boldsymbol{I}_n - \eta P \sum_{i=1}^{p} \boldsymbol{A}_i \boldsymbol{K}_i^{-1} \boldsymbol{A}_i^T\right)$, $\boldsymbol{b}(P) = -\eta P \sum_{i=1}^{p} \left(\boldsymbol{A}_i \boldsymbol{K}_i^{-1} \boldsymbol{c}_i + \frac{1}{p}\boldsymbol{b}\right)$

Dual variable in LSDA is stable [Lee, Dass, et al 2016], if and only if

$$\rho\Big(\boldsymbol{A}(P)\Big) < 1, \quad \text{i.e.} \boldsymbol{A}(P) \text{is Schur stable}$$

where, $\rho(.)$ denotes spectral radius of the given matrix, i.e. largest magnitude of eigen value

Optimal step size: Proof

Let, $f_1(P) = |1 - \lambda_{min}(\boldsymbol{M})P|$ and $f_2(P) = |1 - \lambda_{max}(\boldsymbol{M})P|$

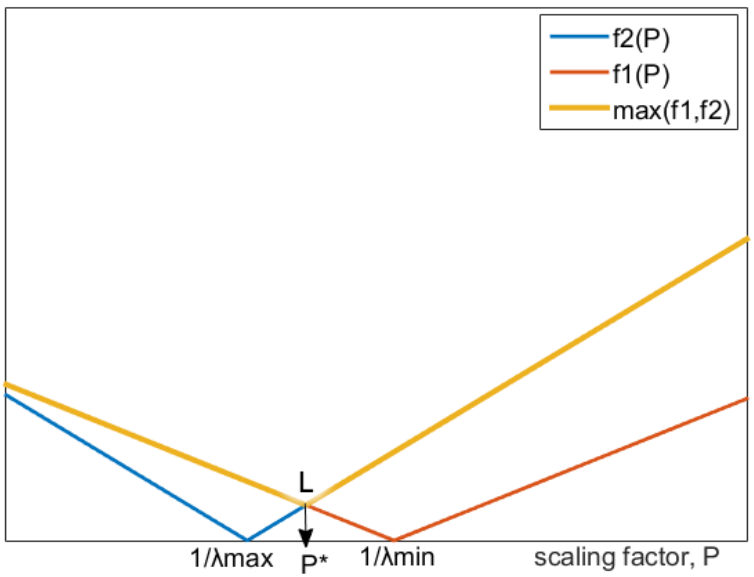


Figure: Plotting optimal scaling factor $P^{\star}$

Intersection point $(\bar{\lambda}^{-1}, \lambda_{max}\bar{\lambda}^{-1} - 1))$ where $\bar{\lambda} = \frac{\lambda_{max}(\mathbf{M}) + \lambda_{min}(\boldsymbol{M})}{2}$

SVM formulation

Specifically, we focus on SVM with following properties

- **Binary-class** (two class classification)
- **Soft margin** (to handle outliers and noise)
- l_2-**regularization** (for differentiability, analytical solution)
- l_2-**loss** (for differentiability, stable solution)

Optimal step size: Proof

$$\max\{f_1(P), f_2(P)\} = \begin{cases} 1 - \lambda_{min}(\mathbf{M})P & \text{if } 0 < P \leq \bar{\lambda}^{-1} \\ \lambda_{max}(\mathbf{M})P - 1 & \text{if } P > \bar{\lambda}^{-1} \end{cases}$$

$$\underset{P}{\text{argmin}} \max\{f_1(P), f_2(P)\} = \bar{\lambda}^{-1} = \frac{2}{\lambda_{max}(\mathbf{M}) + \lambda_{min}(\mathbf{M})}$$

Since, optimal scaling factor $P^\star \in \mathbb{N}$

$$P^\star = \begin{cases} 1 & \text{if} \quad 0 < \bar{\lambda}^{-1} < 2 \\ \lfloor \bar{\lambda}^{-1} \rfloor & \text{if} \quad \bar{\lambda}^{-1} \geq 2 \end{cases}$$

Distributed QRSVM: Parameter Discussions [ICDCS 17]

Parameters	**a9a**	**covtype**
rank, k	40	64
C	2^{-1}	2^{-1}
γ	2^{-3}	2^{3}
approx. K_{error}	0.51	0.58
#processors, p	16	16
stopping threshold	10^{-3}	10^{-3}
optimal step size, η^*	1.9	1.9
#iterations, t	166	512