6G FLAGSHIP UNIVERSITY OF OULU

CWC cwc.oulu.fi

CENTRE FOR WIRELESS COMMUNICATIONS
University of Oulu

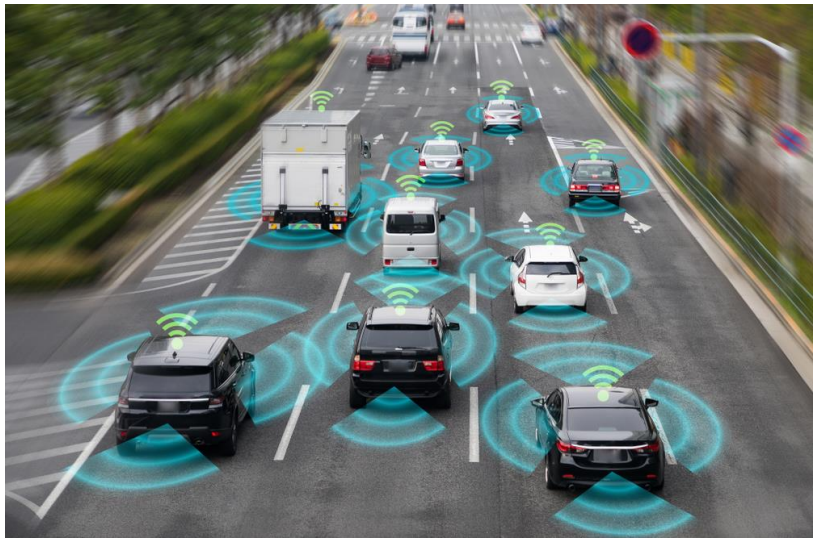
UNIVERSITY OF OULU

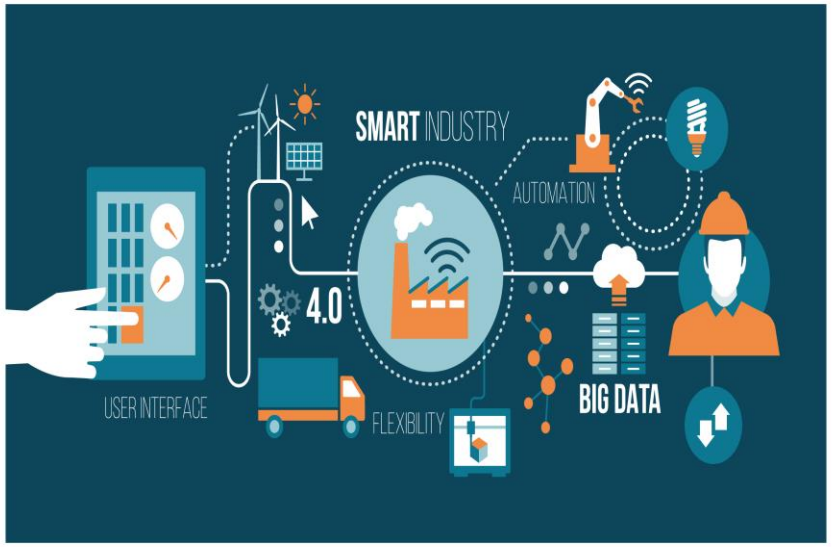
strategic Learning in Wireless Communications
Part III: Edge ML

Anis Elgabli

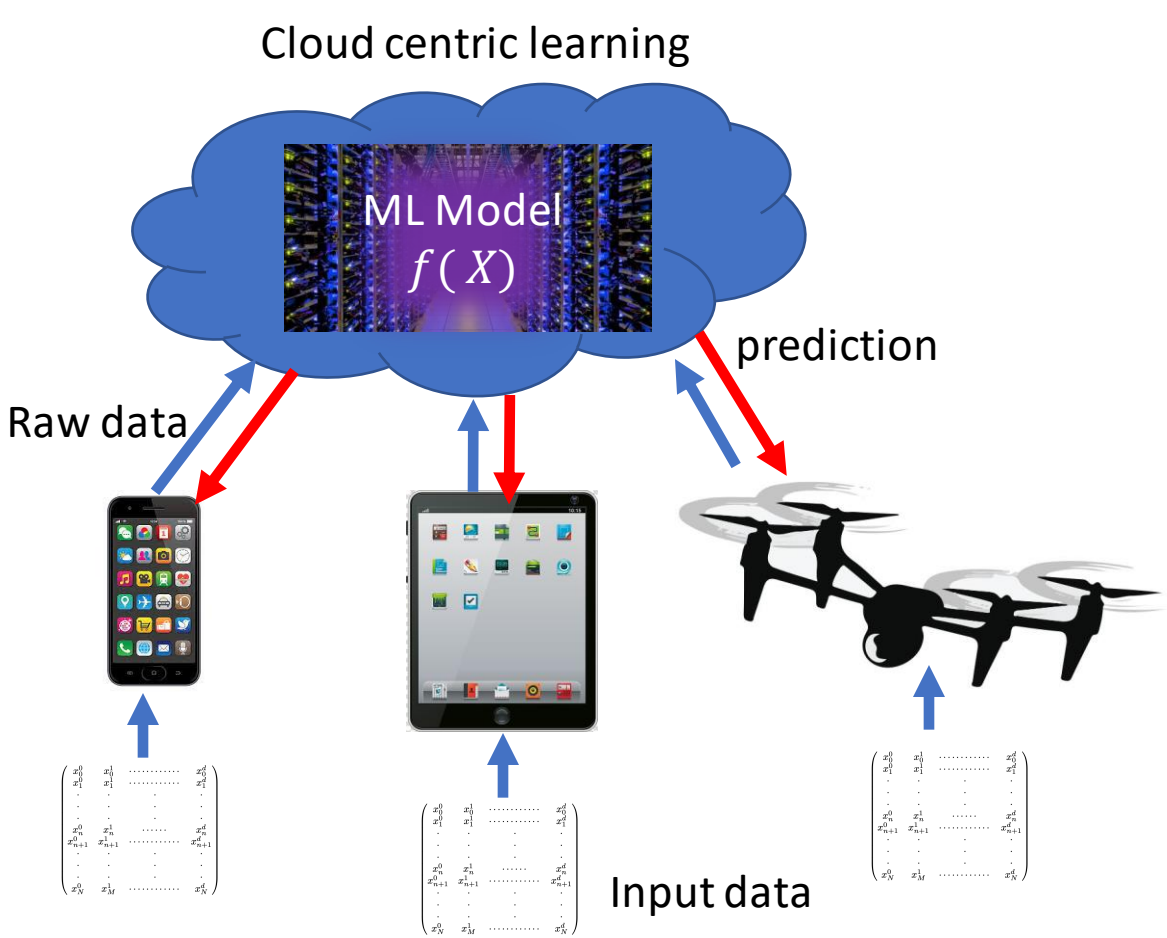
Anis.elgabli@oulu.fi

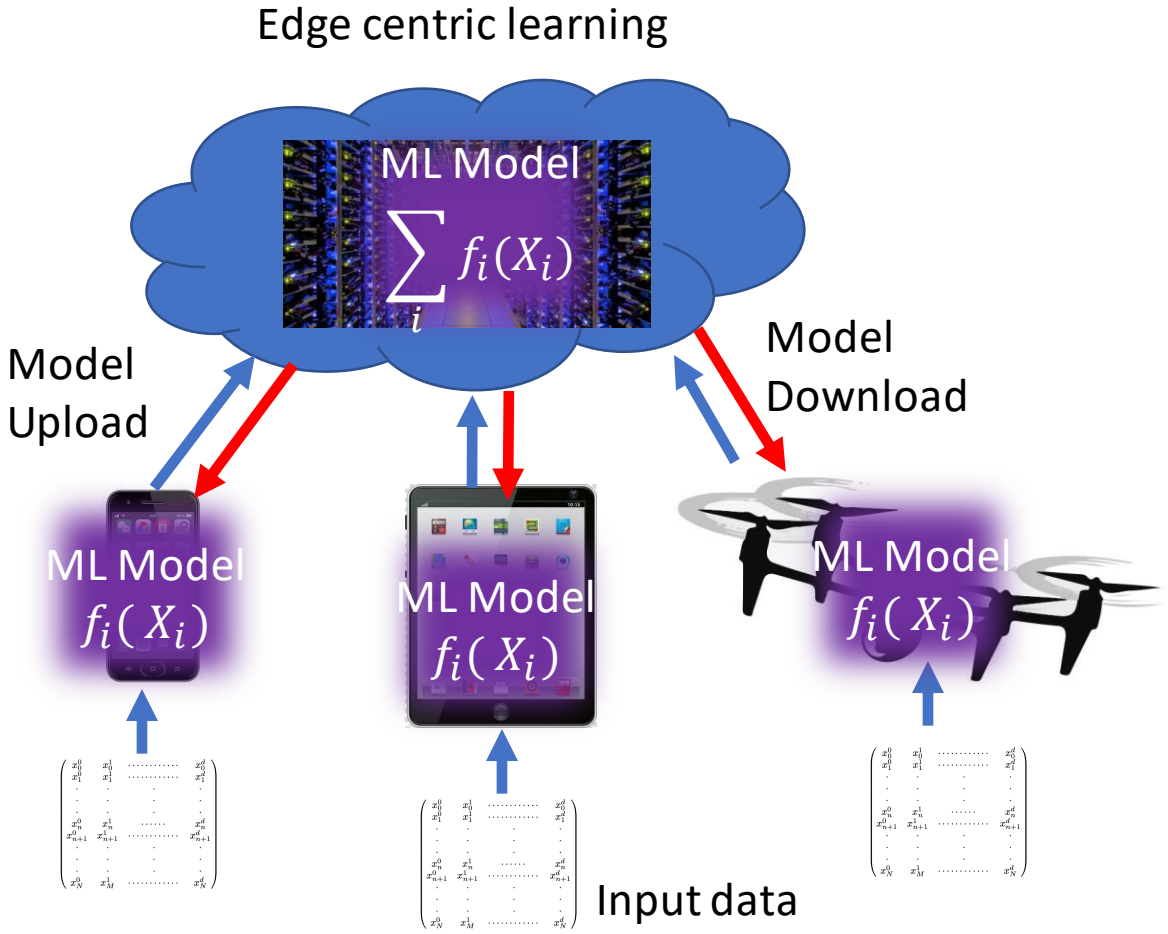
IoT and IoE paradigm: Huge amount of data is generated every where every second





Distributed ML: Moving ML computations to the Edge





High Latency: send data and receive decision
High Comm cost: Limited comm resources, transferring raw data
Privacy issue: Data leaves its origin

Low latency: infer locally and respond
Low Communication cost: transferring model parameters
Privacy: No raw data exchange
Low Computation cost: Every node process data

“Supervised” Machine Learning in a nutshell…

Training data

$$X = \begin{pmatrix} x_0^0 & x_0^1 & \cdots & x_0^d \\ x_1^0 & x_1^1 & \cdots & x_1^d \\ \vdots & \vdots & \vdots & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^d \\ x_{n+1}^0 & x_{n+1}^1 & \cdots & x_{n+1}^d \\ \vdots & \vdots & \vdots & \vdots \\ x_N^0 & x_M^1 & \cdots & x_N^d \end{pmatrix}$$

Learn the mapping between the input data and the output “label” that minimizes the loss function

ML Model
$f(X)$

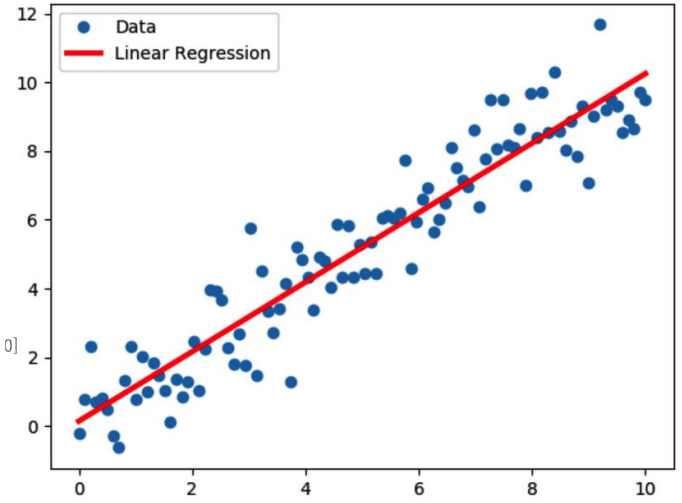
Output

$$Y' = f(X)$$

Loss function = $f(Y, Y')$
Y is the actual output “label”

e.g. Simple 1D linear regression

$$y_n' = b_0 + w_0 \cdot x_n, \; loss = \| Y - Y' \|_2^2$$



Feed data in randomly selectd mini-batches
“Stochastic Algorithm”
Feed all data at as one batch
“Deterministic Algorithm”

Why?

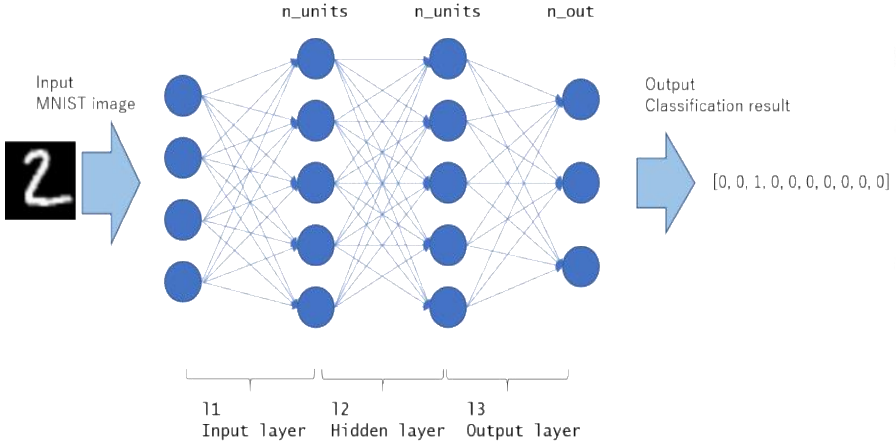
Less computation per iteration

Pay off

More iterations

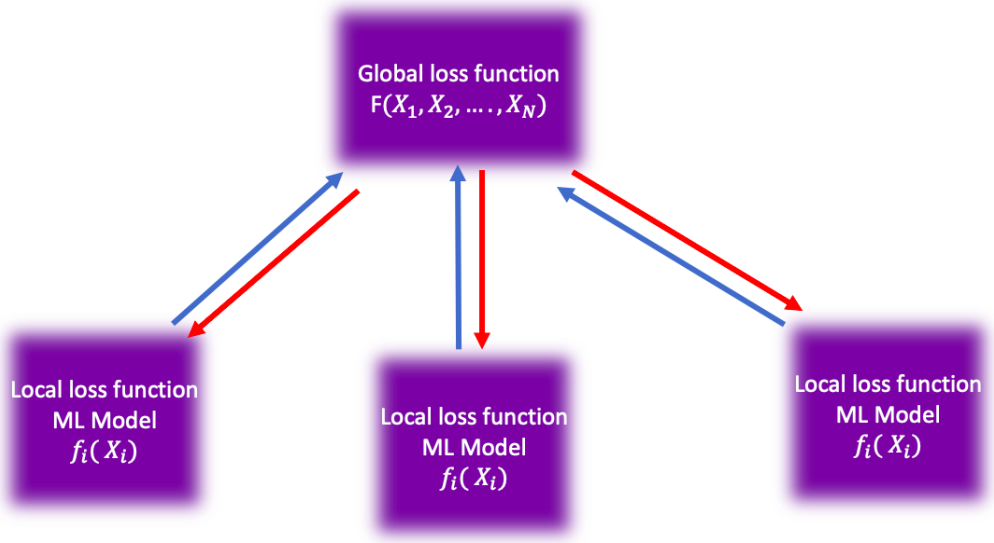
e.g., Image classification
wights and the biases

$$loss = -\sum_n y_n' \log y_n$$



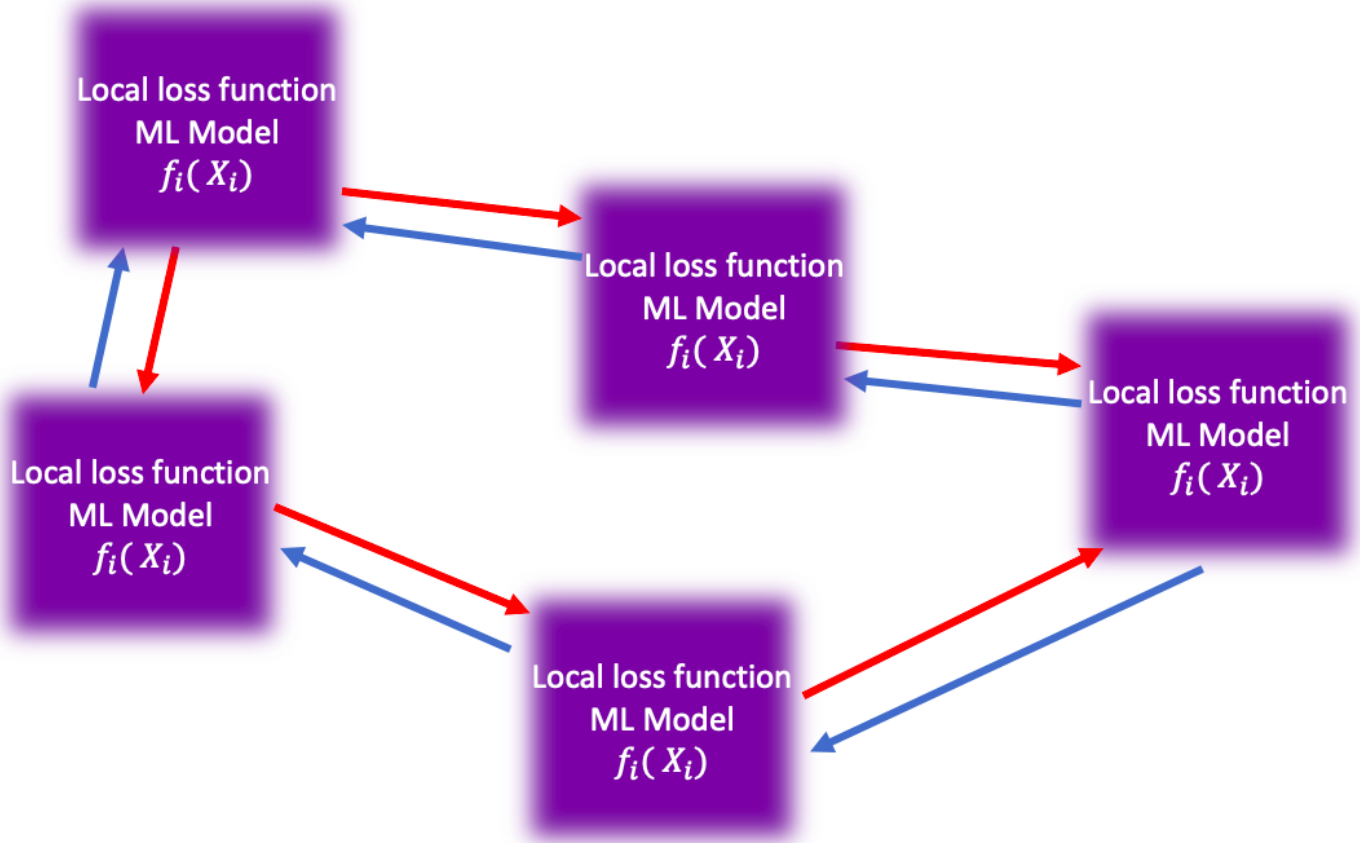
Extending to distributed (Federated) Learning.

Parameter server (PS) based learning



Exchange updates with the PS

Decentralized learning



Exchange updates with neighbors

Distributed Machine Learning, Key Points

Network topology:

- Parameter server-based topology
- No Parameter-server (Decentralized topology)

Structure of the Optimization Formulation:

- Unconstrained optimization
- Constrained Optimization
- Sum of separable functions
- Non separable function

The Optimization Problem:

- **Convex:** Linear, logistic regression, etc
- **Non-convex:** Image Classification using Deep Neural Network, DNN
- **Deterministic:**
  - Use all data at every iteration
- **Stochastic:**
  - Use a randomly selected mini-bath to update at every iteration

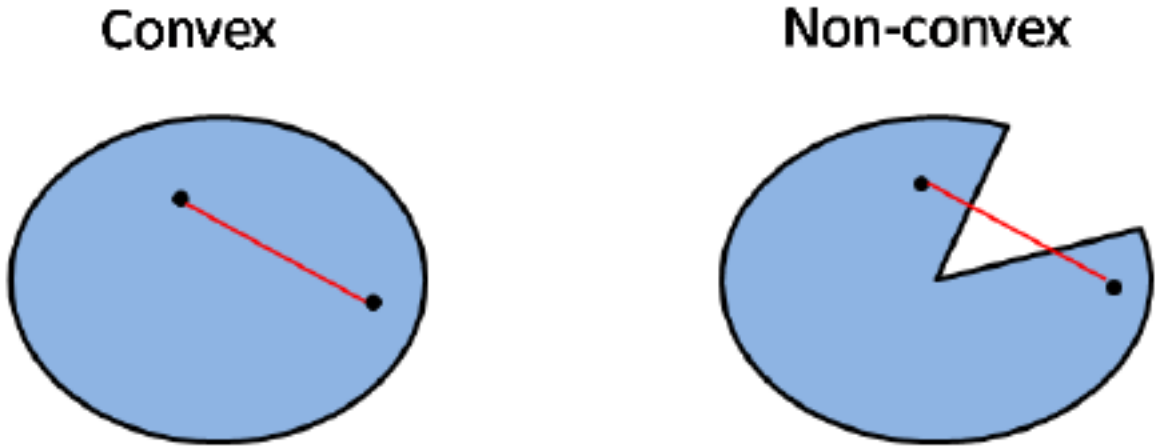
Algorithms:

- **Parameter server-based topology**
  - Unconstrained: GD, SGD
  - Constrained: Optimization Dual Decomposition, Alternating Direction Method of Multipliers (ADMM)
- **Decentralized topology**
  - Unconstrained: DGD, DSGD
  - Constrained: Dual Averaging, Group-ADMM (GADMM)

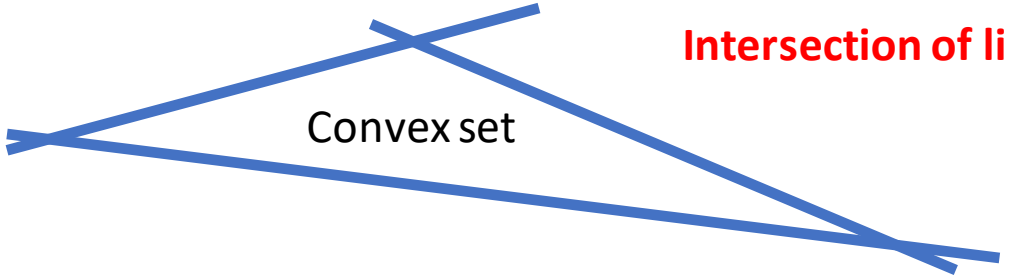
A brief recap on Convex sets, functions, and problems

Convex set

A set is convex only and only if the line between any two points in the set belongs to the set. i.e., if x and y belongs to the set, then $\lambda \cdot x + (1 - \lambda)y$ is belong to the set, $0 \leq \lambda \leq 1$.



Intersection of subspaces is a convex set:

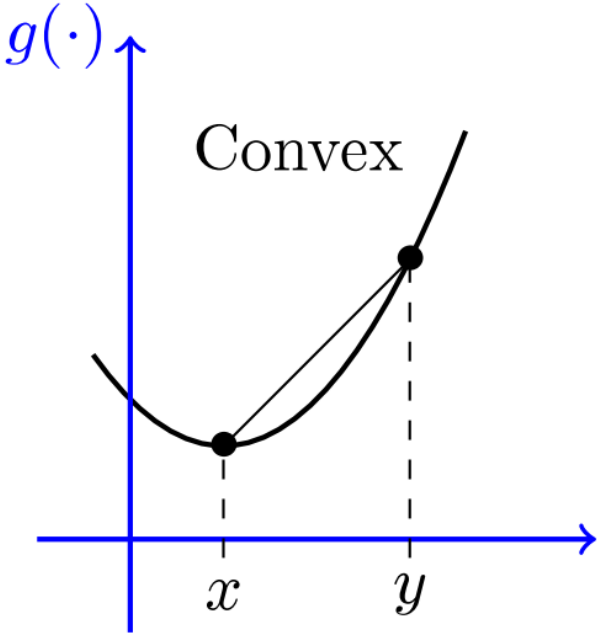


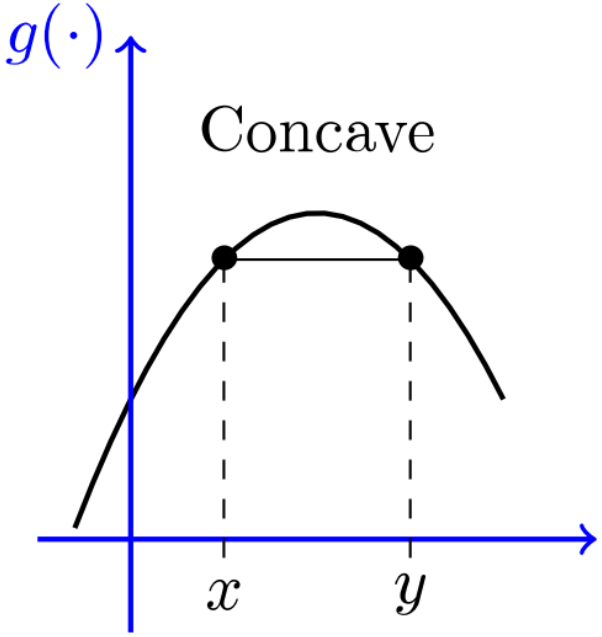
Intersection of linear constraints is a convex set

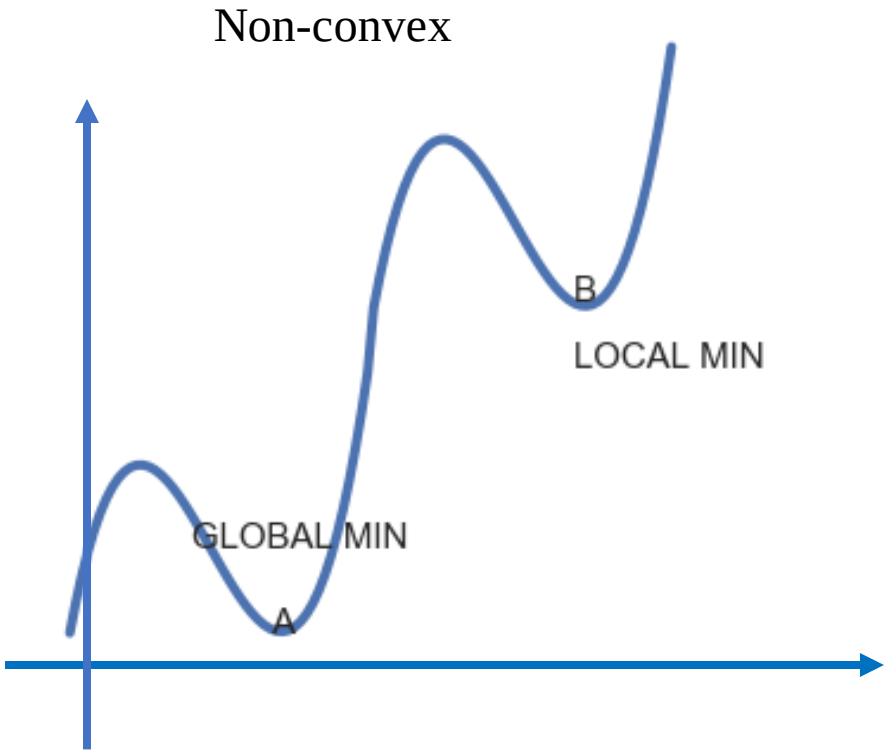
Convex, Concave, and Non-convex functions: Recap

First, the domain of the function or the feasible set of the problem must be convex:

Necessary but not sufficient

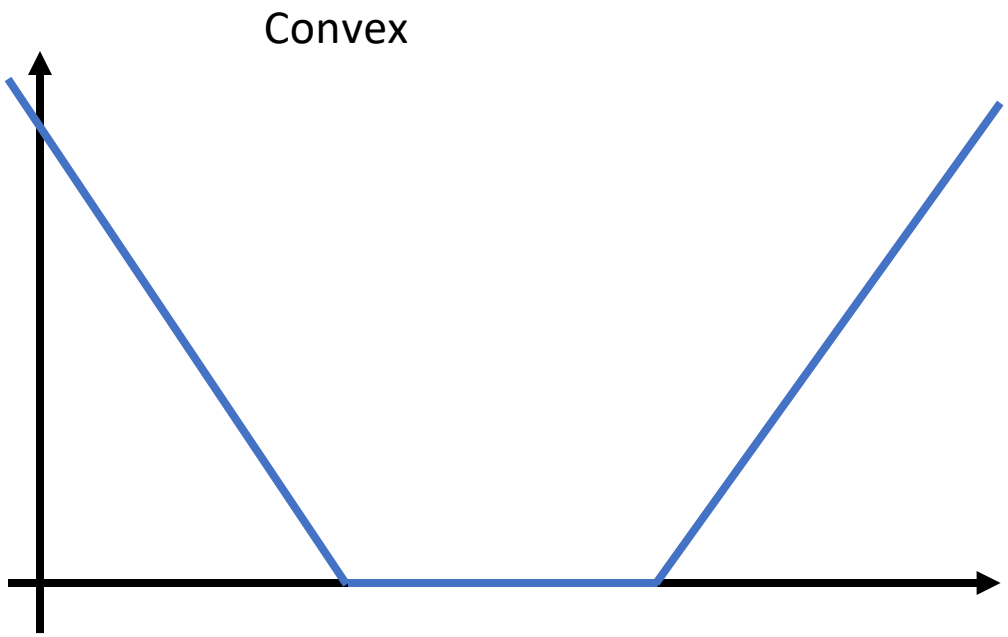


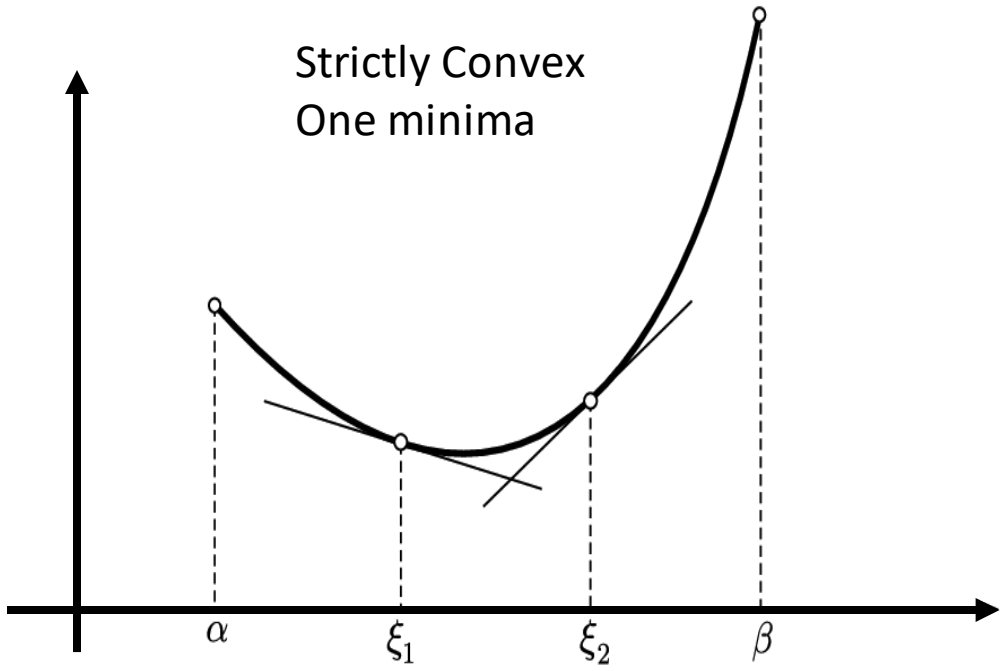




Every local minima is a global minima

Convex, Strictly convex





$\forall x, y \in C, \forall \lambda \in [0,1]: \quad f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$

$f(\lambda x + (1-\lambda)y) < \lambda f(x) + (1-\lambda) f(y)$

If its second derivative is positive at all points then the function is strictly convex

Strong Convexity

- Strong convexity means that there exists a quadratic lower bound on the growth of the function
  - So, every strong convex function is strictly convex
- A differentiable function is strongly convex with parameter $m > 0$ if:

$$(\nabla f(x) - \nabla f(y))^T (x - y) \geq m\|x - y\|_2^2$$

Equivelantly:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{m}{2}\|y - x\|_2^2$$

A non-diffrentiable function is strongly convex if:

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y) - \frac{1}{2}mt(1-t)\|x - y\|_2^2$$

As $m \to 0$, strong convexity approach the definition of strict convexity.
When $m = 0$, is identical to the definition of a convex function

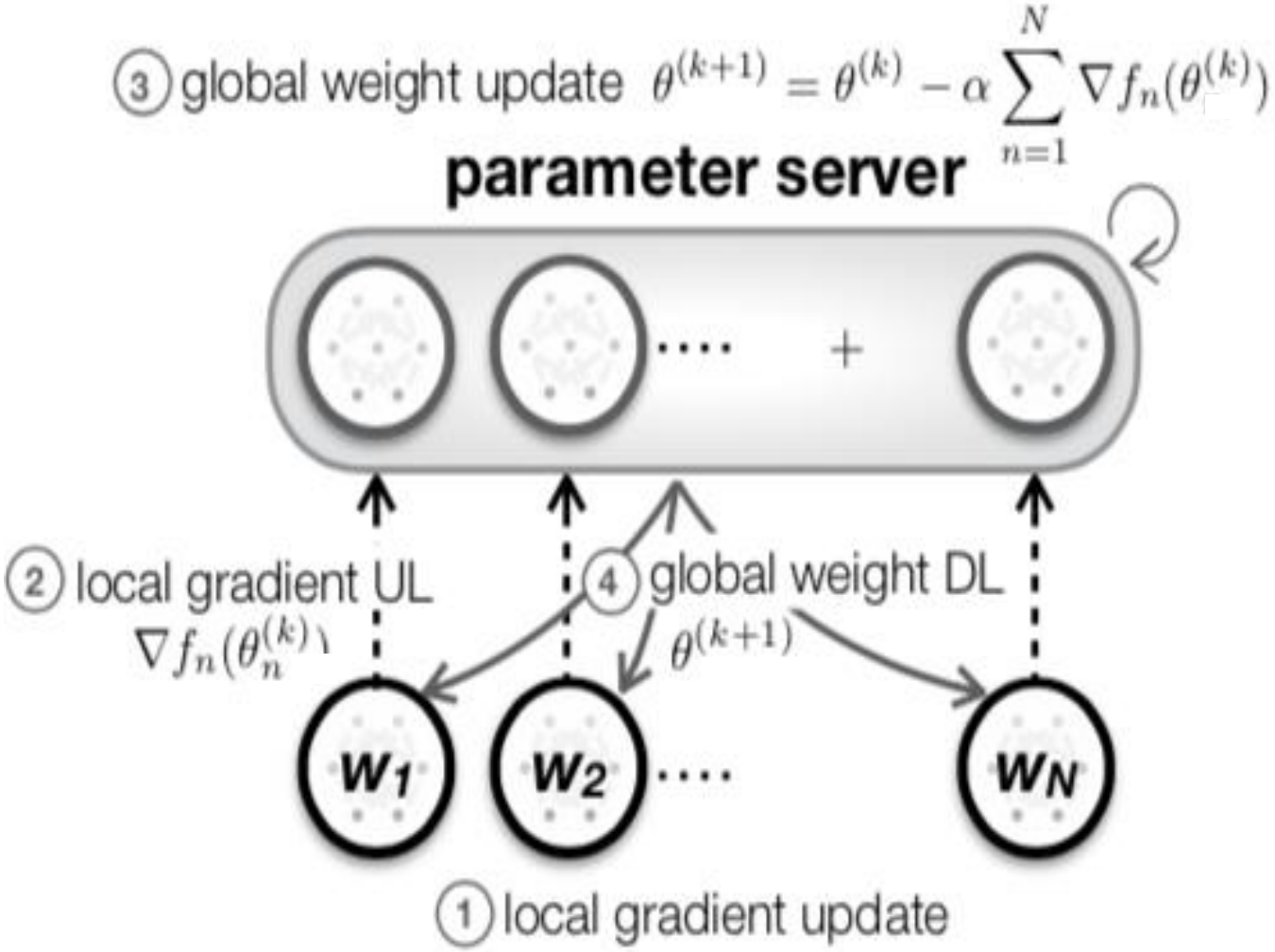
Distributed (Federated) Learning problem:

Unconstrained formulation:

$$\min_{\Theta} F(\Theta), \quad F(\Theta) := \sum_{n=1}^{N} f_n(\Theta)$$

N is the number of workers
$\Theta \in R^d$ is the global model parameter
$f_n(\Theta)$ is the n-th worker's objective function

Aggregated Gradient Descent



When every $f_n(\Theta)$ is convex. GD converges to the optimal solution

Stochastic Gradient Descent (Mini-batch SGD)

Local Step at each worker:

Gradient at the i-th worker and k-th iteration using a mini-batch os size $|\xi_k^{(i)}|$

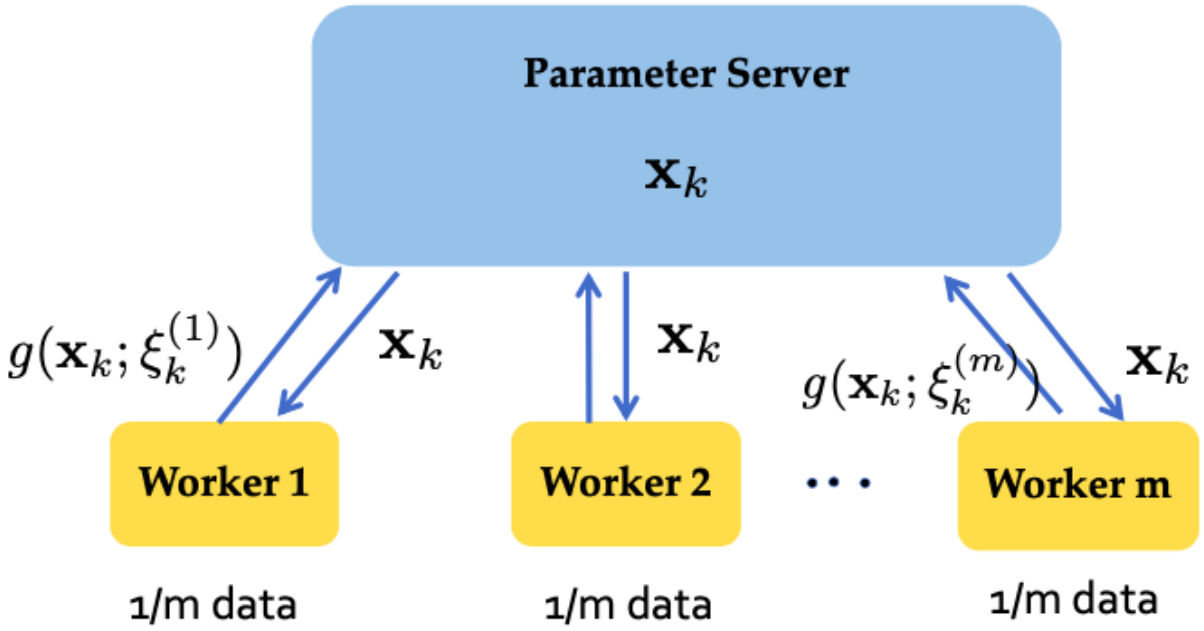
$$g(\mathbf{x}_k;\xi_k^{(i)}) = \frac{1}{|\xi_k^{(i)}|}\sum_{j\in\xi_k^{(i)}} \nabla f_j(\mathbf{x}_k)$$

Global Step at the PS:

Averaging local models across all workers at the PS

$$\mathbf{x}_{k+1} = \frac{1}{N}\sum_{i=1}^{N}\left[\mathbf{x}_k - \eta g(\mathbf{x}_k;\xi_k^{(i)})\right]$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\eta}{m}\sum_{i=1}^{m} g(\mathbf{x}_k;\xi_k^{(i)})$$

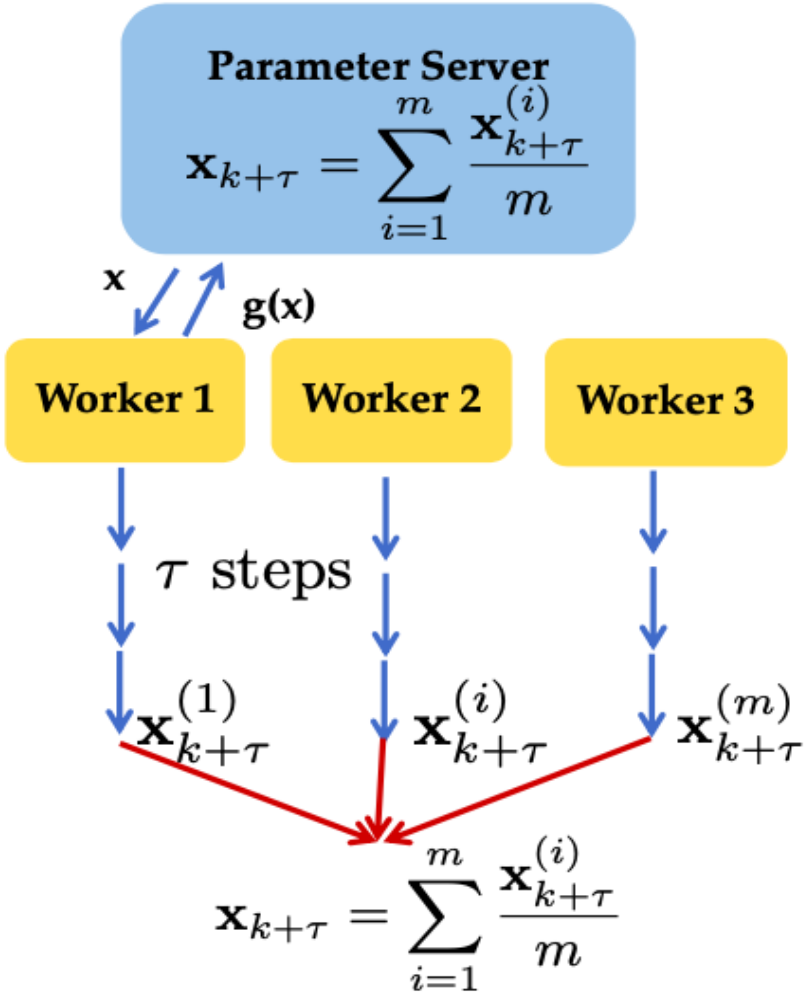


Federated Averaging: (local SGD)

Workers perform τ local updates → Local Computation

Local models are averaged after every τ local steps → Communication

May not converge.

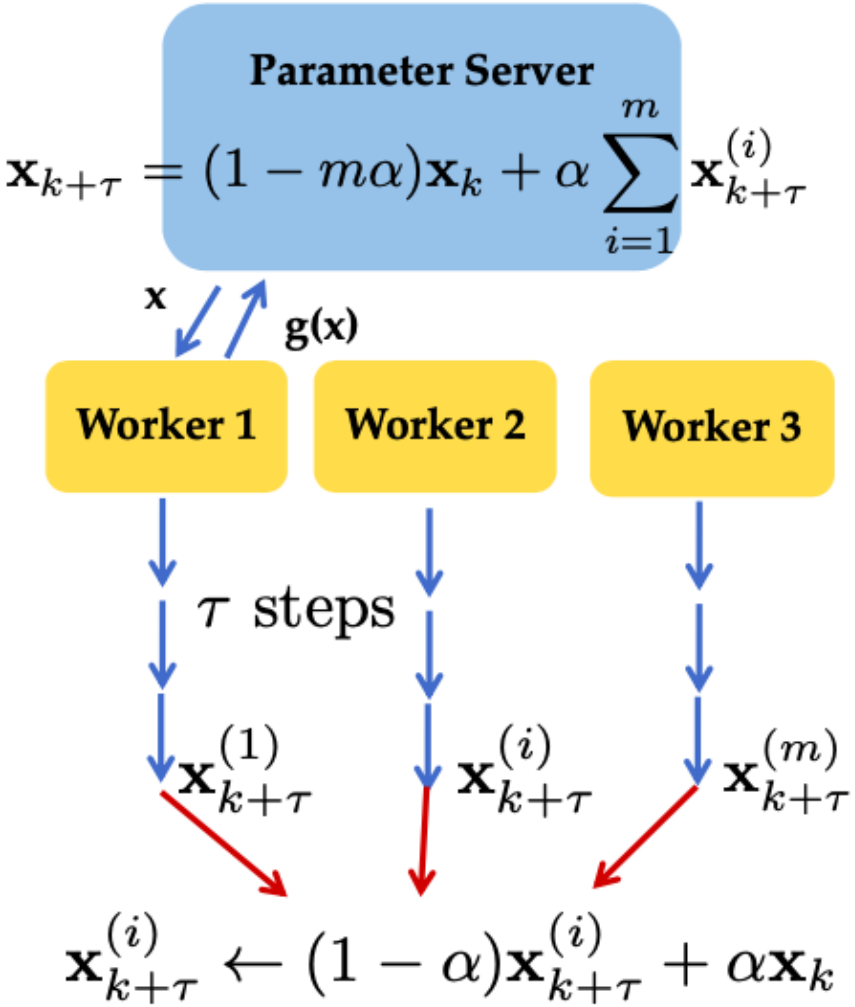


Wang, Jianyu, and Gauri Joshi. "Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms." *arXiv preprint arXiv:1808.07576* (2018).

Elastic Averaging:

Workers perform τ local updates→ Local Computation

To avoid diverging, worker i updates according to the following rule:



Wang, Jianyu, and Gauri Joshi. "Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms." *arXiv preprint arXiv:1808.07576* (2018).

Communication-Efficient Distributed Machine Learning

- **Exchange minimum number of variables per iteration.**
  e.g. Exchange gradient vector, model parameter vector, model output.

- **Minimum number of workers transmit per iteration**
  - e.g. LAG (Lazily Aggregated Gradient)

- **Minimum number of workers transmit plus Quantization:**
  e.g. LAQ

- **Minimum number of neighbors to communicate with (Decentralization)**
  e.g. GADMM (Group ADMM)

- **Minimum number of neighbors plus model Quantization.**
  e.g. Q-GADMM

Communication-Efficient Learning: Temporal sparsity through Lazily Aggregated Gradient Descent

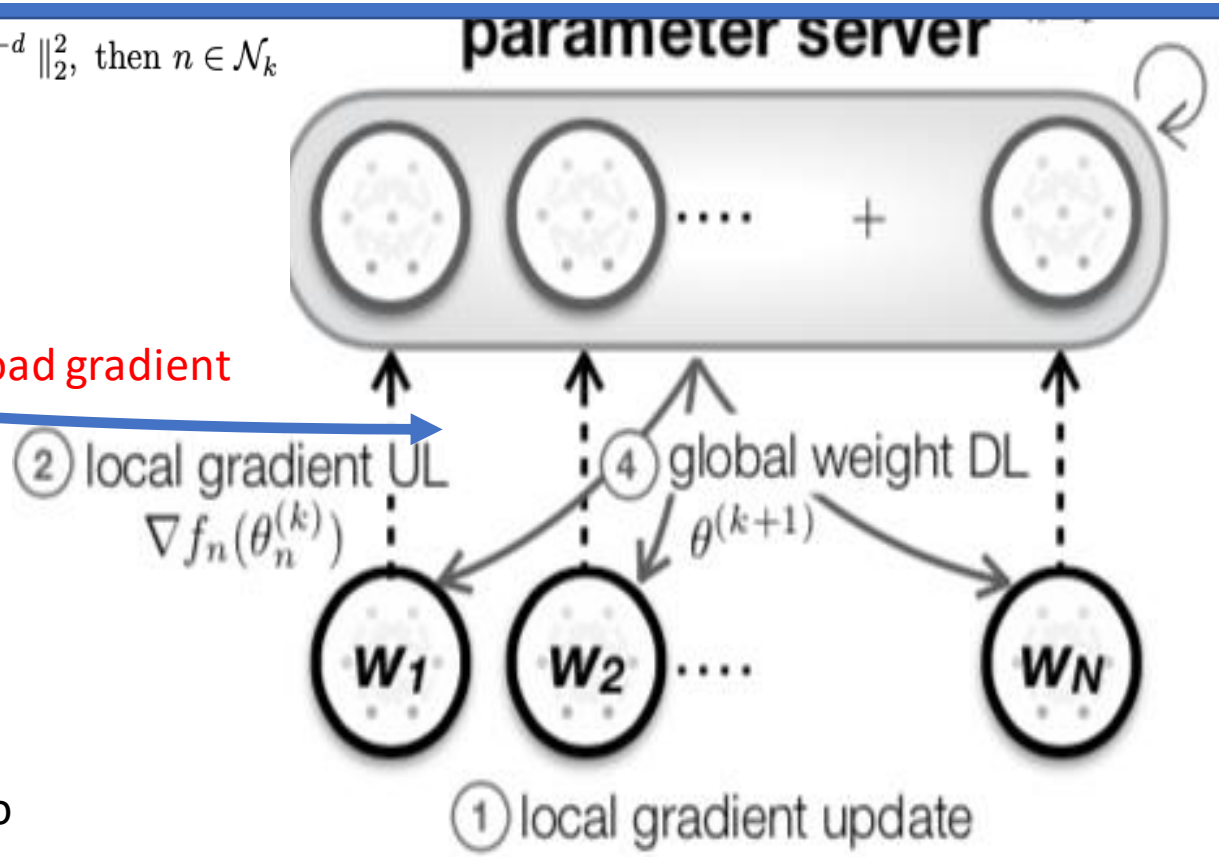
Unconstrained formulation:

$$\min_{\Theta} F(\Theta), \quad F(\Theta) := \sum_{n=1}^{N} f_n(\Theta)$$

LAG WK, if : $\| \nabla f_n(\theta^{(K)}) - \nabla f_n(\theta^{(K+1)}) \|_2^2 > \frac{1}{\alpha^2 M^2} \sum_{d=1}^{D} \zeta_d \| \theta^{k+2-d} - \theta^{k+1-d} \|_2^2$, then $n \in \mathcal{N}_k$

Lazily Aggregated Gradient Descent

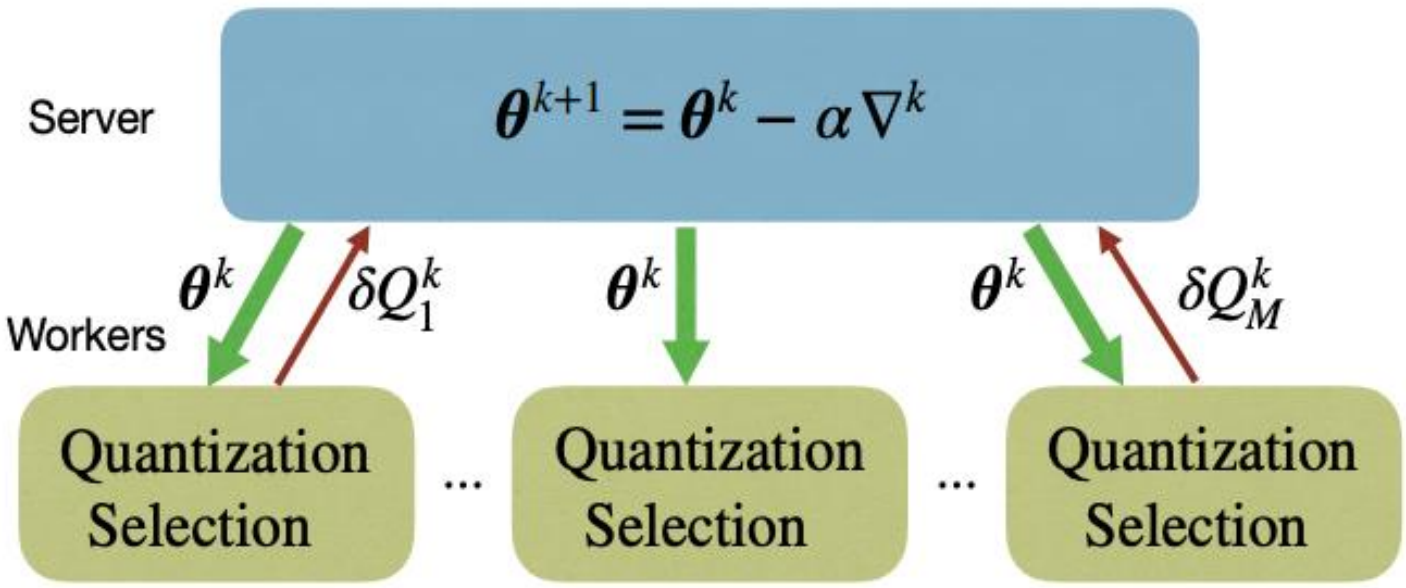
$$\theta^{(k+1)} = \theta^{(k)} - \alpha \sum_{n} \nabla f_n(\theta_n^{(K-1)}) - \alpha \sum_{n \in \mathcal{N}_k} \delta \nabla f_n(\theta^{(K)})$$



- **Exchange gradients only**
- **Total Communication cost (TC) per iteration (LAG-WK):**
  - $TC = \sum_{t=1}^{T_a} (L_{BC,t}^c + \sum_{n=1}^{N} 1_{\{n \in N_k\}} \cdot L_{n,t}^c)$

- $L_{BC,t}^c$: The cost of using the downlink broadcast channel
- $L_{n,t}^c$: The cost of using the uplink channel from worker n to the PS.
- T_a: Is the total number of iterations to reach accuracy a.

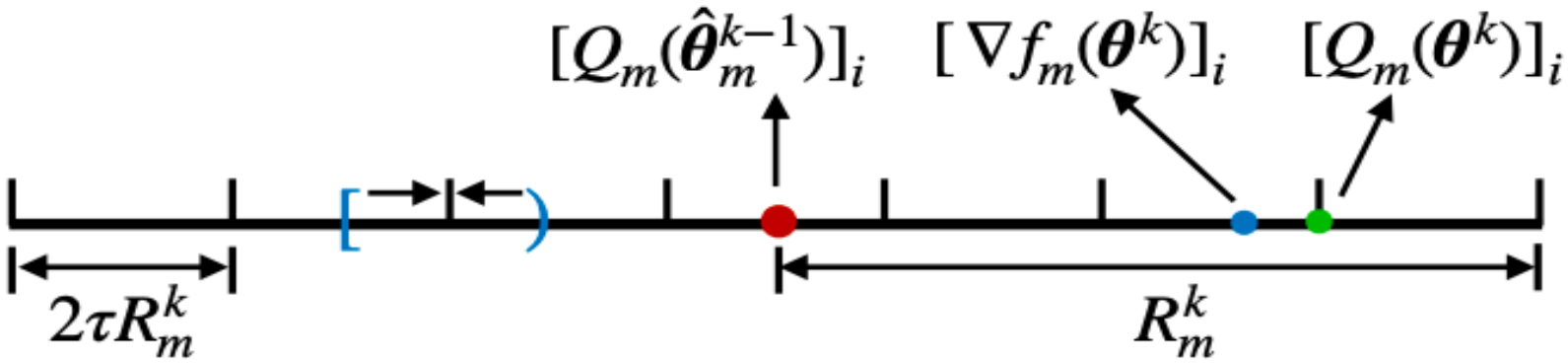
LAQ: Low payload size per exchange through Lazily Aggregated Quantized Gradient



LAQ iteration $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \nabla^k$ with $\nabla^k = \nabla^{k-1} + \sum_{m \in \mathcal{M}^k} \delta Q_m^k$

$$\boldsymbol{\varepsilon}^k := \nabla f(\boldsymbol{\theta}^k) - Q(\boldsymbol{\theta}^k) = \sum_{m=1}^{M} \boldsymbol{\varepsilon}_m^k; \text{ that is, } Q(\boldsymbol{\theta}^k) = \nabla f(\boldsymbol{\theta}^k) - \boldsymbol{\varepsilon}^k$$

LAQ: Lazily Aggregated Quantized Gradient



Quantization example ($b = 3$)

$$Q_m(\boldsymbol{\theta}^k) = \mathcal{Q}(\nabla f_m(\boldsymbol{\theta}^k), Q_m(\hat{\boldsymbol{\theta}}_m^{k-1}))$$

$$R_m^k = \|\nabla f_m(\boldsymbol{\theta}^k) - Q_m(\hat{\boldsymbol{\theta}}_m^{k-1})\|_\infty$$

$$\tau := 1/(2^b - 1)$$

$$[q_m(\boldsymbol{\theta}^k)]_i = \left\lfloor \frac{[\nabla f_m(\boldsymbol{\theta}^k)]_i - [Q_m(\hat{\boldsymbol{\theta}}_m^{k-1})]_i + R_m^k}{2\tau R_m^k} + \frac{1}{2} \right\rfloor, \quad i = 1, \cdots, p$$

$$\delta Q_m^k = Q_m(\boldsymbol{\theta}^k) - Q_m(\hat{\boldsymbol{\theta}}_m^{k-1}) = 2\tau R_m^k q_m(\boldsymbol{\theta}^k) - R_m^k \mathbf{1}$$

transmit R_m^k and $q_m(\boldsymbol{\theta}^k)$

which can be transmitted by $32 + bp$ bits

32 bits for R_m^k and bp bits for $q_m(\boldsymbol{\theta}^k)$

Instead of $32p$ bits

Distributed constrained optimization: Recap

Dual decomposition:

$$\text{minimize} \quad \sum_{i=1}^{N} f_i(x_i)$$

$$\text{s.t} \quad \sum_{i} A_i x_i = b$$

The Lagrangian can be written as:

$$L(x\,,y) = \sum_{i=1}^{N} f_i(x_i) \; + \; y^T \sum_{i} (A_i x_i - b)$$

$$L(x,y) = \sum_{i=1}^{N} L_i(x_i,y) = \sum_{i=1}^{N} \left(f_i(x_i) + y^T A_i x_i - (1/N) y^T b\right)$$

With that the x minimization problem splits into N separate Problems that can be solved in parallel:

Primal update: $$x_i^{k+1} := \underset{x_i}{\operatorname{argmin}}\, L_i(x_i, y^k)$$

Dual update: $$y^{k+1} := y^k + \alpha^k (Ax^{k+1} - b).$$

Where:

$$Ax = \sum_{i=1}^{N} A_i x_i$$

dual ascent often cannot be used.

- Example, if f is a nonzero affine function of any component of x, then the x-update fails, since L is unbounded below in x for most y.
- What is the alternative?

Alternating direction method of multipliers: ADMM

ADMM was initally introduced to solve the following optimization problem:

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c \end{array}$$

The Augmented Lagrangian can be written as:

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|_2^2$$

ADMM consists of the iterations

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_x L_\rho(x, z^k, y^k) \\ z^{k+1} &:= \operatorname*{argmin}_z L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} &:= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \end{aligned}$$

Also called Gauss seidel ADMM

Multiblock-ADMM

$$\begin{aligned} \text{minimize} \quad & f_1(\mathbf{x}_1) + \cdots + f_N(\mathbf{x}_N) \\ \text{subject to} \quad & A_1\mathbf{x}_1 + \cdots + A_N\mathbf{x}_N = c \end{aligned}$$

$$\mathcal{L}_\rho(\mathbf{x}_1, \ldots, \mathbf{x}_N, \lambda) = \sum_{i=1}^{N} f_i(\mathbf{x}_i) - \lambda^\top \left(\sum_{i=1}^{N} A_i\mathbf{x}_i - c \right) + \frac{\rho}{2} \left\| \sum_{i=1}^{N} A_i\mathbf{x}_i - c \right\|_2^2$$

Algorithm 2: Gauss-Seidel ADMM

Initialize $\mathbf{x}^0,\ \lambda^0,\ \rho > 0$;

for $k = 0,\ 1, \ldots$ **do**

Update $\mathbf{x}_i$ for $i = 1, \ldots, N$ *sequentially* by:

$\mathbf{x}_i^{k+1} = \min_{\mathbf{x}_i} f_i(\mathbf{x}_i) + \frac{\rho}{2} \left\| \sum_{j<i} A_j\mathbf{x}_j^{k+1} + A_i\mathbf{x}_i + \sum_{j>i} A_j\mathbf{x}_j^k - c - \frac{\lambda^k}{\rho} \right\|_2^2$;

Update $\lambda^{k+1} = \lambda^k - \rho \left(\sum_{i=1}^{N} A_i\mathbf{x}_i^{k+1} - c \right)$.

Algorithm 3: Jacobian ADMM

Initialize $\mathbf{x}^0,\ \lambda^0,\ \rho > 0$;

for $k = 0,\ 1, \ldots$ **do**

Update $\mathbf{x}_i$ for $i = 1, \ldots, N$ *in parallel* by:

$\mathbf{x}_i^{k+1} = \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}_i) + \frac{\rho}{2} \left\| A_i\mathbf{x}_i + \sum_{j \neq i} A_j\mathbf{x}_j^k - c - \frac{\lambda^k}{\rho} \right\|_2^2$;

Update $\lambda^{k+1} = \lambda^k - \rho \left(\sum_{i=1}^{N} A_i\mathbf{x}_i^{k+1} - c \right)$.

Algorithm 4: Proximal Jacobian ADMM

Initialize: $\mathbf{x}_i^0$ $(i = 1, 2, \ldots, N)$ and λ^0;

for $k = 0, 1, \ldots$ **do**

Update $\mathbf{x}_i$ for $i = 1, \ldots, N$ *in parallel* by:

$\mathbf{x}_i^{k+1} = \arg\min_{\mathbf{x}_i} f_i(\mathbf{x}_i) + \frac{\rho}{2} \left\| A_i\mathbf{x}_i + \sum_{j \neq i} A_j\mathbf{x}_j^k - c - \frac{\lambda^k}{\rho} \right\|_2^2 + \frac{1}{2} \left\| \mathbf{x}_i - \mathbf{x}_i^k \right\|_{P_i}^2$;

Update $\lambda^{k+1} = \lambda^k - \gamma\rho(\sum_{i=1}^{N} A_i\mathbf{x}_i^{k+1} - c)$.

The Parameter server (PS) based Learning problem:

Constrained formulation:

$$\min_{\Theta,\{\theta_n\}_{n=1}^N} \sum_{n=1}^N f_n(\theta_n)$$
$$\text{s.t.} \quad \theta_n = \Theta, \quad \forall\, n$$

ADMM:

The update steps of the primal and dual variables at iteration k+1:

Augmented Lagrangian: $\mathcal{L}_\rho(\Theta, \{\theta_n\}_{n=1}^N, \lambda) = \sum_{n=1}^N f_n(\theta_n) + \sum_{n=1}^N \langle \lambda_n, \theta_n - \Theta\rangle + \frac{\rho}{2}\sum_{n=1}^N \|\theta_n - \Theta\|^2$

Primal variable, Worker n:

$$\theta_n^{k+1} = \arg\min_{\theta_n} \left[f_n(\theta_n) + \langle \lambda_n^k, \theta_n - \Theta^k\rangle + \frac{\rho}{2}\|\theta_n - \Theta^k\|^2\right]$$

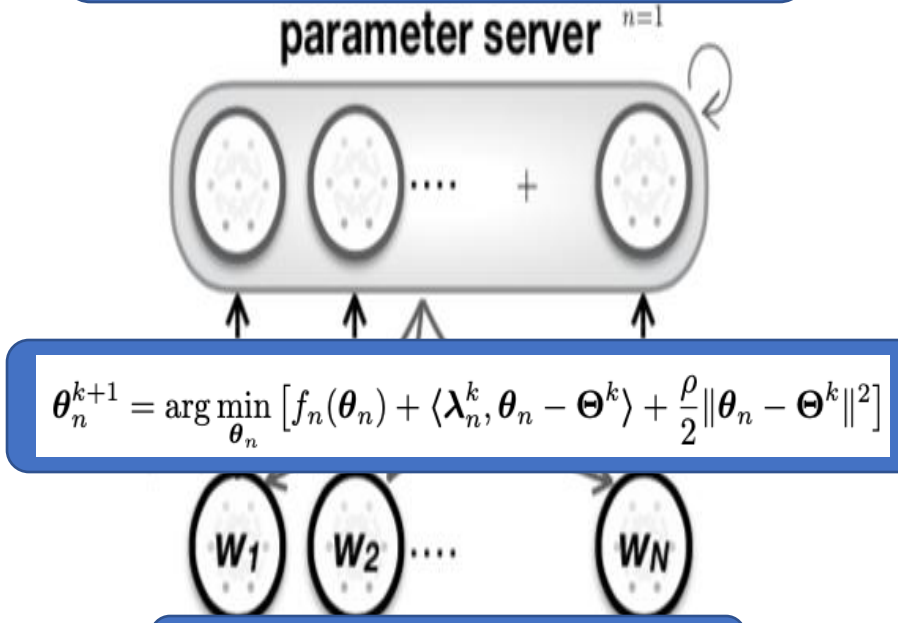
Primal variable, Parameter Server:

$$\Theta^{k+1} = \frac{1}{N}\sum_{n=1}^N \left(\theta_n^{k+1} + \frac{1}{\rho}\lambda_n^k\right)$$

Dual Variables:

$$\lambda_n^{k+1} = \lambda_n^k + \rho(\theta_n^{k+1} - \Theta^{k+1}), n = \{1, \cdots, N\}$$

$$\Theta^{k+1} = \frac{1}{N}\sum_{n=1}^N \left(\theta_n^{k+1} + \frac{1}{\rho}\lambda_n^k\right)$$



$$\theta_n^{k+1} = \arg\min_{\theta_n} \left[f_n(\theta_n) + \langle \lambda_n^k, \theta_n - \Theta^k\rangle + \frac{\rho}{2}\|\theta_n - \Theta^k\|^2\right]$$

$$\lambda_n^{k+1} = \lambda_n^k + \rho(\theta_n^{k+1} - \Theta^{k+1})$$

Exchange model parameters only

Total Communication cost (TC) per iteration:

- $TC = \sum_{t=1}^{T_a}(L_{BC,t}^c + \sum_{n=1}^N L_{n,t}^c)$

Building and training ML models using Tensorflow

- Tensorflow:
  - Open source library to help you develop and train ML models
  - Based on Python

- Our task:
  - Build and train a Deep Neural Network (DNN) model to perform image classification
  - Will start with a simple model that runs in a single machine, and then we extend it to distributed (federated learning) setting.
  - For the single machine-based model, we use the slides from this tutorial on youtube https://www.youtube.com/watch?v=vq2nnJ4g6N0, so for detailed steps, you may watch This video.

MNIST dataset

MNIST database contains 60,000 training images and 10,000 testing images.

- Handwritten digits 0-9.
- 28x28 pixels

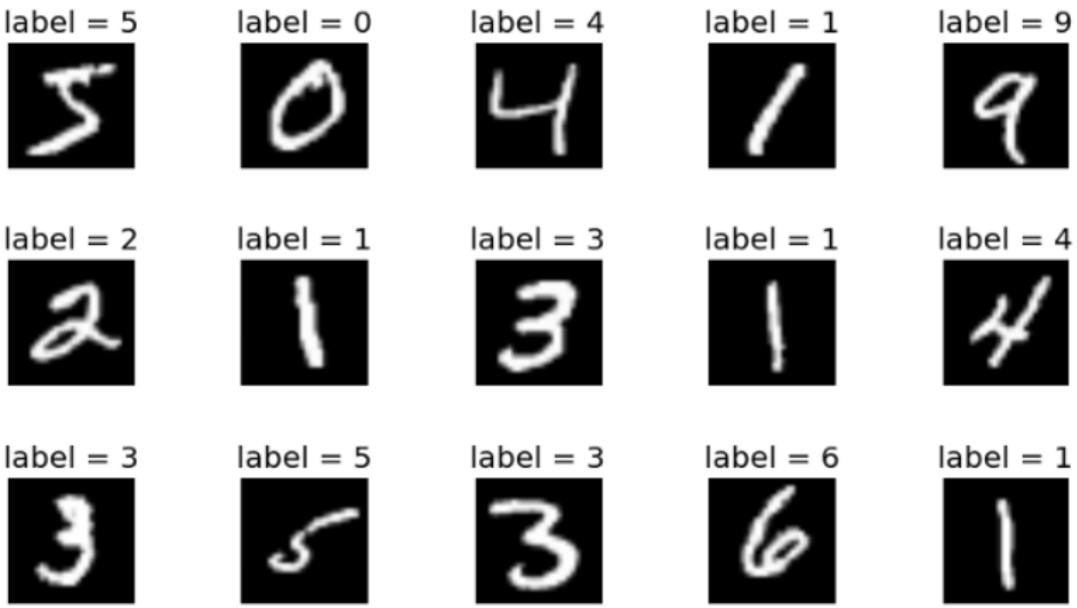
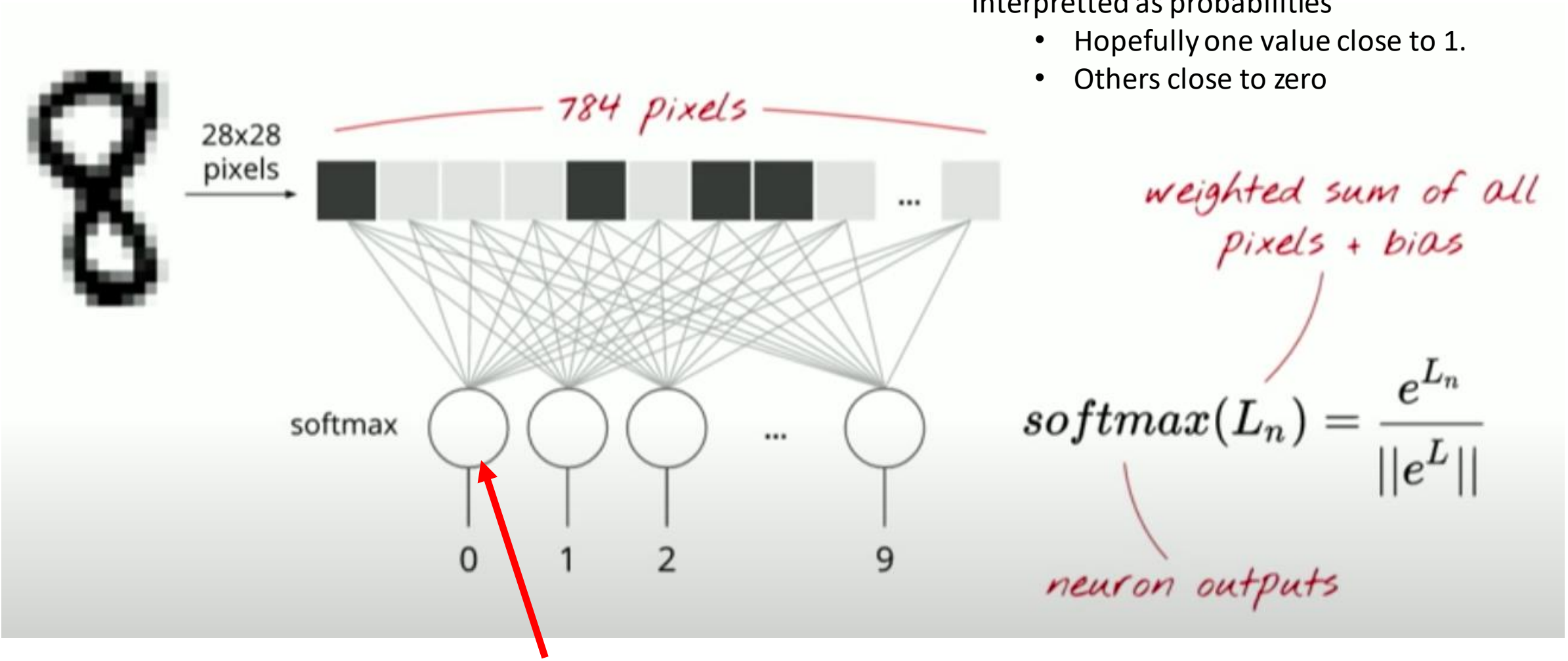


Image calssification using NN.

Simple approach: 1 layer, 10 neurons, Why 10?
Falttening all pixels into one vector: 28x28pixels → 784 pixels

Exponential normalized: increase the differences
Produces 10 values between 0 and 1
Interpretted as probabilities

- Hopefully one value close to 1.
- Others close to zero



take a weighted sum of all pixels, add a bias, and feed through a non-linear activation function

Softmax in a batch of images

Feeding one batch of 100 images at a time

Predictions
Y[100, 10]

Images
X[100, 784]

Weights
W[784,10]

Biases
b[10]

$$Y = softmax(X.W + b)$$

applied line by line

matrix multiply

broadcast on all lines

tensor shapes in [ ]

In tensorflow

```
Y = tf.nn.softmax(tf.matmul(X, W) + b)
```

Loss function

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	0	0	0

actual probabilities, "one-hot" encoded

Cross entropy: $$-\sum Y_i'.log(Y_i)$$

this is a "6"

computed probabilities

0	1	2	3	4	5	6	7	8	9
0.1	0.2	0.1	0.3	0.2	0.1	0.9	0.2	0.1	0.1

Tensorflow: Initialization

```
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

init = tf.initialize_all_variables()
```

this will become the batch size, 100

Tensorflow

```
# model
Y = tf.nn.softmax(tf.matmul(tf.reshape(X, [-1, 784]), W) + b)
# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])
```

"one-hot" encoded

```
# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))
```

"one-hot" decoding

```
# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

```
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)
```

Learning rate

Tensorflow

```
sess = tf.Session()
sess.run(init)

for i in range(1000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ?
    a,c = sess.run([accuracy, cross_entropy], feed_dict=train_data)

    # success on test data ?
    test_data={X: mnist.test.images, Y_: mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy], feed=test_data)
```

running a Tensorflow computation, feeding placeholders

Tip: do this every 100 iterations

Tensorflow

initialisation

```python
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
init = tf.initialize_all_variables()
```

model

```python
# model
Y=tf.nn.softmax(tf.matmul(tf.reshape(X,[-1, 784]), W) + b)

# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])
```

success metrics

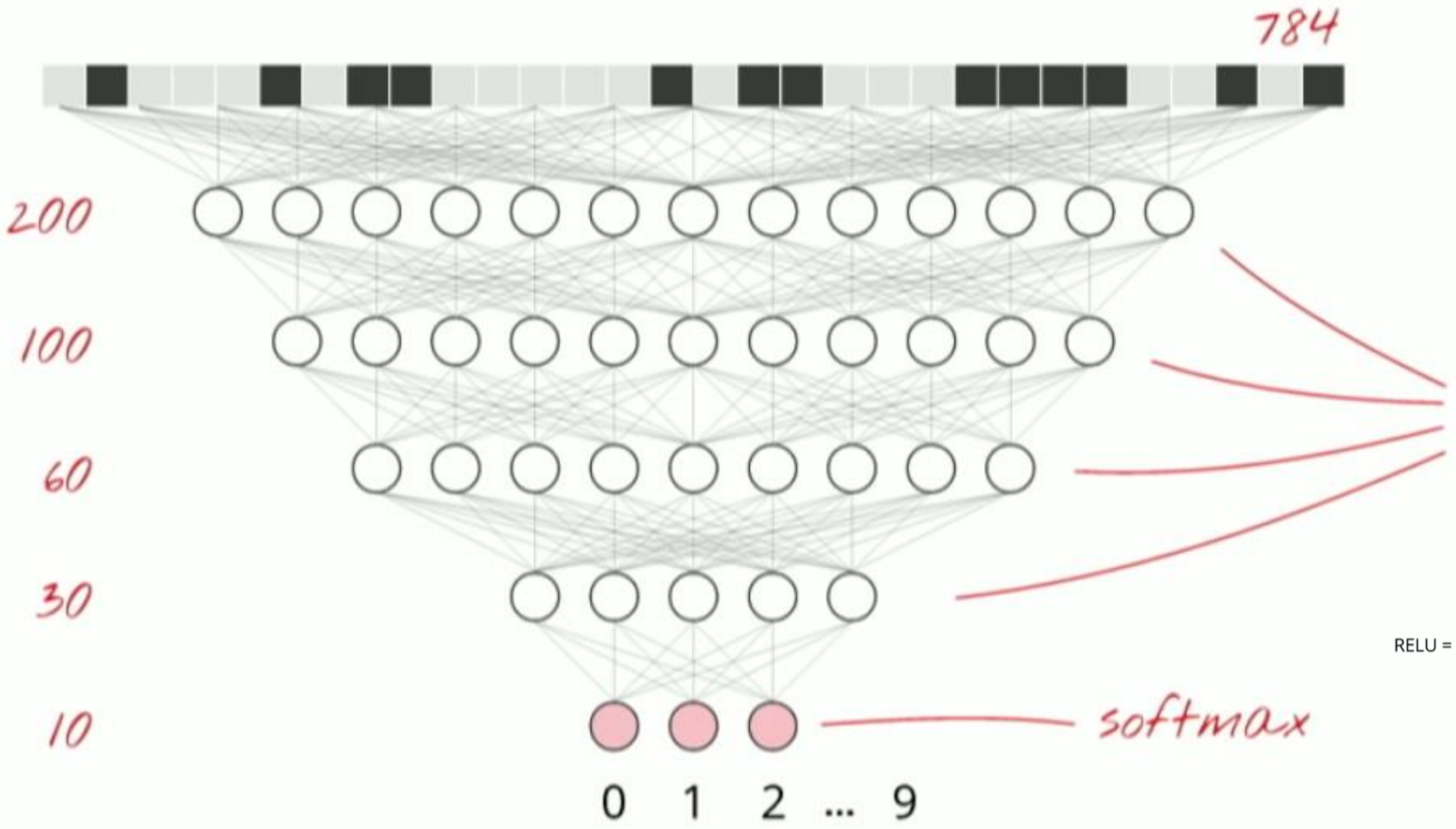
```python
# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))

# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
accuracy = tf.reduce_mean(tf.cast(is_correct,tf.float32))
```

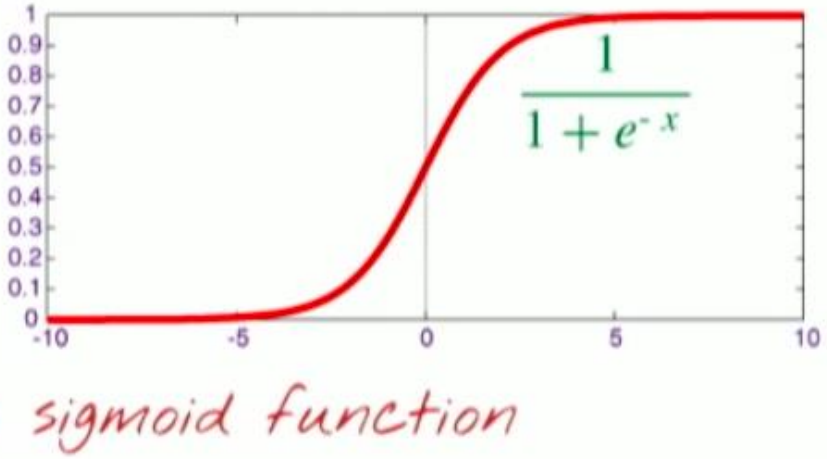
training step

```python
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)

sess = tf.Session()
sess.run(init)

for i in range(10000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ? add code to print it
    a,c = sess.run([accuracy, cross_entropy], feed=train_data)

    # success on test data ?
    test_data={X:mnist.test.images, Y_:mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy], feed=test_data)
```

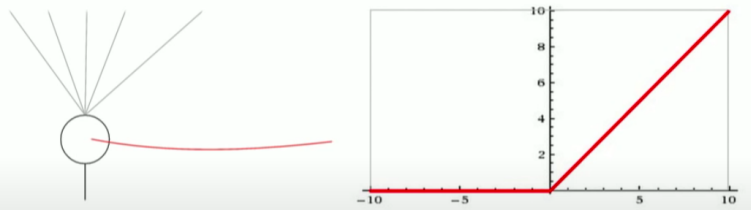
Run

You want better accuracy: Go deeper





RELU = Rectified Linear Unit



```
Y = tf.nn.relu(tf.matmul(X, W) + b)
```

You want better accuracy: Go deeper

```
K = 200
L = 100
M = 60
N = 30

W1 = tf.Variable(tf.truncated_normal([28*28, K] ,stddev=0.1))
B1 = tf.Variable(tf.zeros([K]))

W2 = tf.Variable(tf.truncated_normal([K, L], stddev=0.1))
B2 = tf.Variable(tf.zeros([L]))

W3 = tf.Variable(tf.truncated_normal([L, M], stddev=0.1))
B3 = tf.Variable(tf.zeros([M]))
W4 = tf.Variable(tf.truncated_normal([M, N], stddev=0.1))
B4 = tf.Variable(tf.zeros([N]))
W5 = tf.Variable(tf.truncated_normal([N, 10], stddev=0.1))
B5 = tf.Variable(tf.zeros([10]))
```

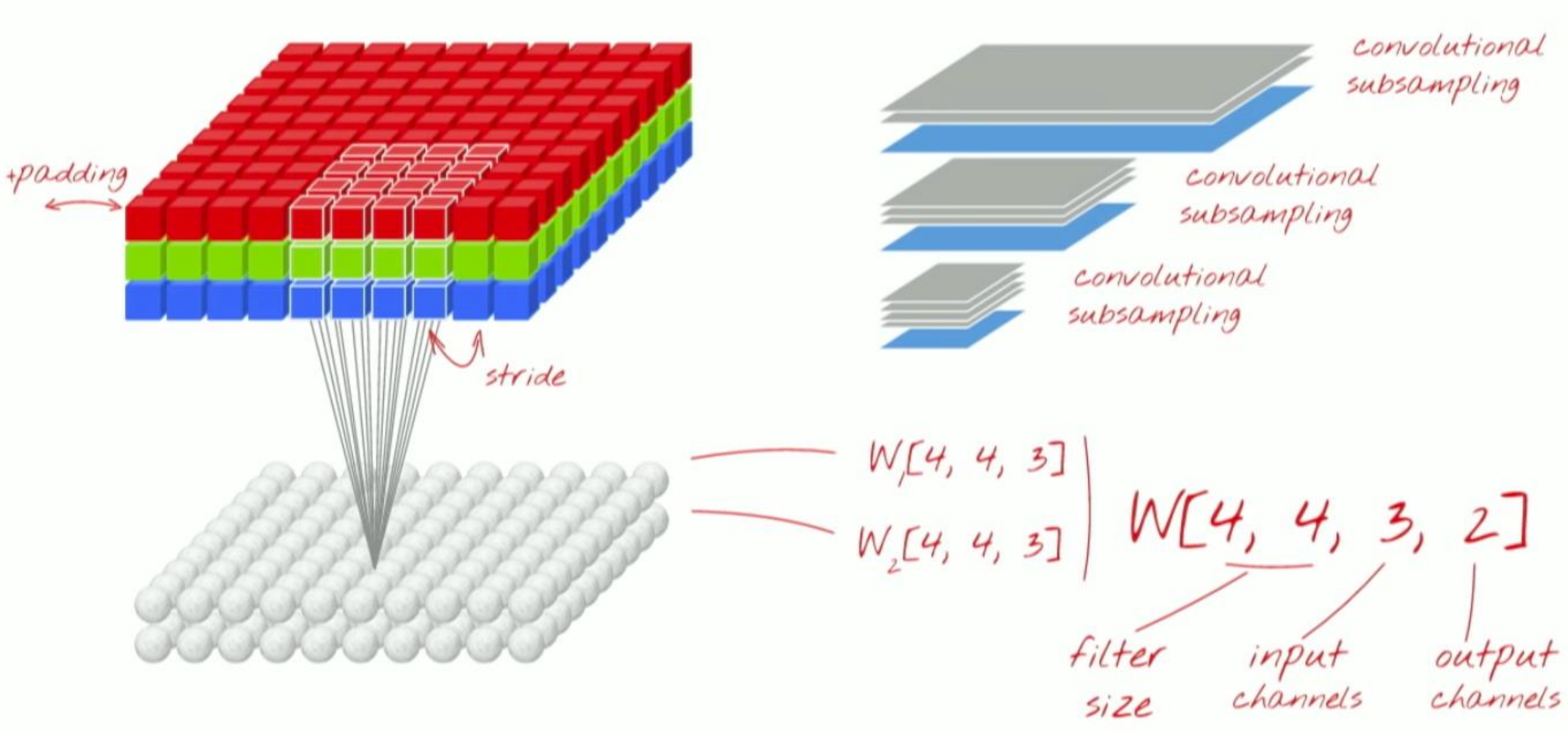
weights initialised
with random values

You want better accuracy: Go deeper

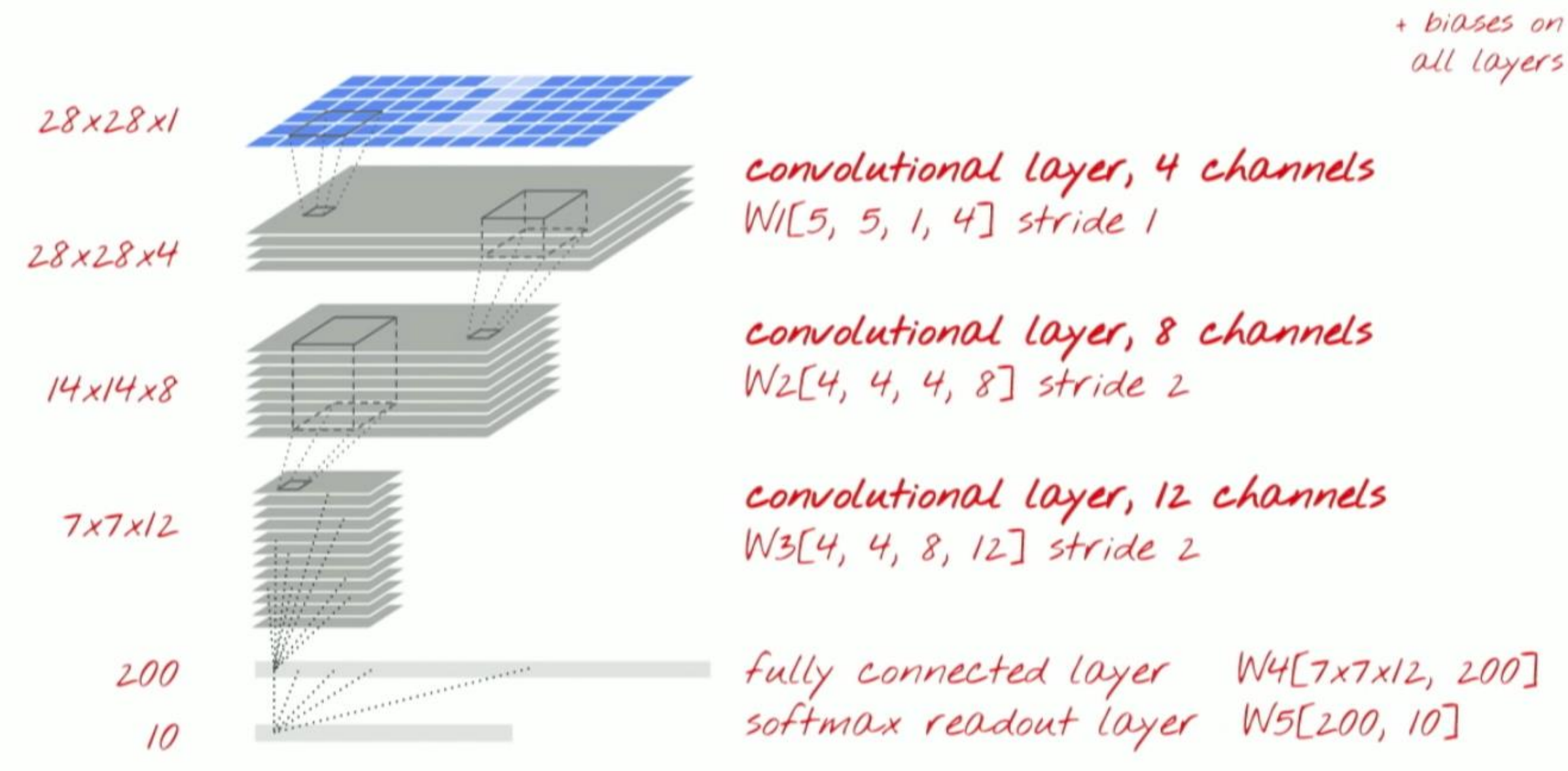
weights and biases

```
X = tf.reshape(X, [-1, 28*28])

Y1 = tf.nn.sigmoid(tf.matmul(X, W1) + B1)
Y2 = tf.nn.sigmoid(tf.matmul(Y1, W2) + B2)
Y3 = tf.nn.sigmoid(tf.matmul(Y2, W3) + B3)
Y4 = tf.nn.sigmoid(tf.matmul(Y3, W4) + B4)
Y = tf.nn.softmax(tf.matmul(Y4, W5) + B5)
```

You want better accuracy: Convolutional Neural Networks (CNN)



You want better accuracy: Convolutional Neural Networks (CNN)



Tensorflow: Convolutional Neural Networks (CNN)

filter size — input channels — output channels

```
K=4
L=8
M=12

W1 = tf.Variable(tf.truncated_normal([5, 5, 1, K] ,stddev=0.1))
B1 = tf.Variable(tf.ones([K])/10)
W2 = tf.Variable(tf.truncated_normal([5, 5, K, L] ,stddev=0.1))
B2 = tf.Variable(tf.ones([L])/10)
W3 = tf.Variable(tf.truncated_normal([4, 4, L, M] ,stddev=0.1))
B3 = tf.Variable(tf.ones([M])/10)

N=200

W4 = tf.Variable(tf.truncated_normal([7*7*M, N] ,stddev=0.1))
B4 = tf.Variable(tf.ones([N])/10)
W5 = tf.Variable(tf.truncated_normal([N, 10] ,stddev=0.1))
B5 = tf.Variable(tf.zeros([10])/10)
```

weights initialised with random values

Tensorflow: Convolutional Neural Networks (CNN)

input image batch
X[100, 28, 28, 1]

weights

stride

biases

```
Y1 = tf.nn.relu(tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME') + B1)
Y2 = tf.nn.relu(tf.nn.conv2d(Y1, W2, strides=[1, 2, 2, 1], padding='SAME') + B2)
Y3 = tf.nn.relu(tf.nn.conv2d(Y2, W3, strides=[1, 2, 2, 1], padding='SAME') + B3)

YY = tf.reshape(Y3, shape=[-1, 7 * 7 * M])

Y4 = tf.nn.relu(tf.matmul(YY, W4) + B4)
Y  = tf.nn.softmax(tf.matmul(Y4, W5) + B5)
```

flatten all values for fully connected layer

Y3 [100, 7, 7, 12]

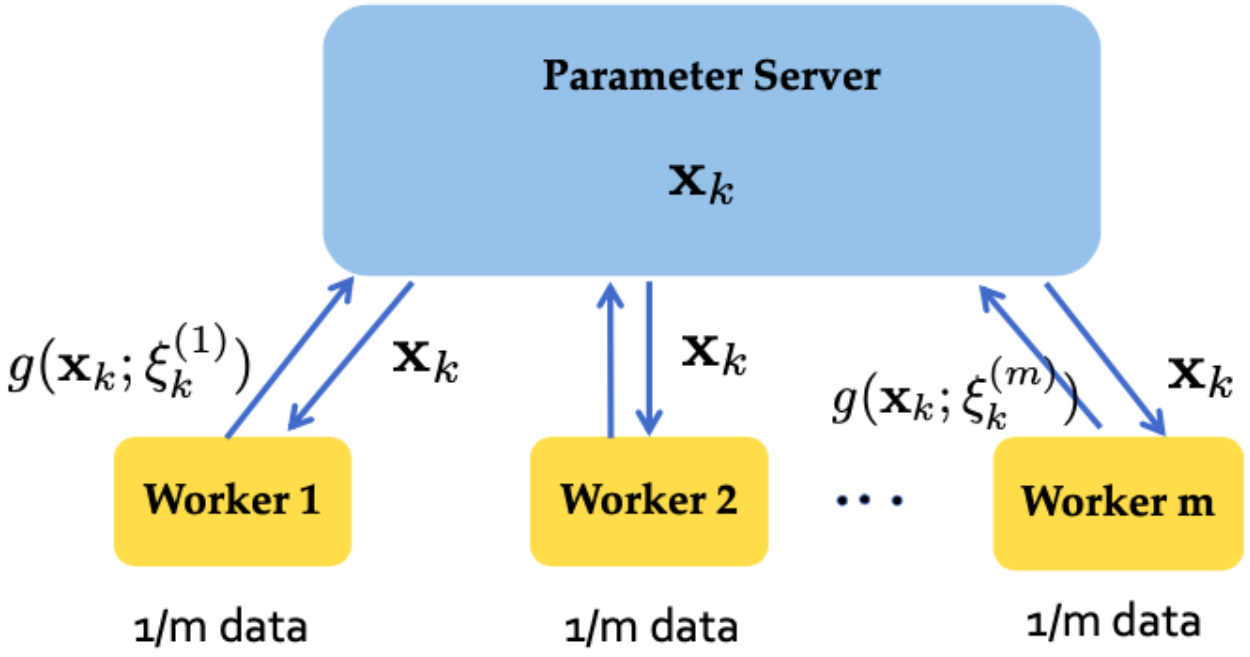
YY [100, 7x7x12]

Tensorflow implementation of DNN for MNIST classification

Each worker just computes the gradient using a randomly selected minibatch with respect to the glaobal model

- Minibatch SGD

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\eta}{m}\sum_{i=1}^{m} g(\mathbf{x}_k; \xi_k^{(i)})$$



Parameter Server Framework

- Gradients are transferred between server and workers
- All worker nodes have the same model parameters

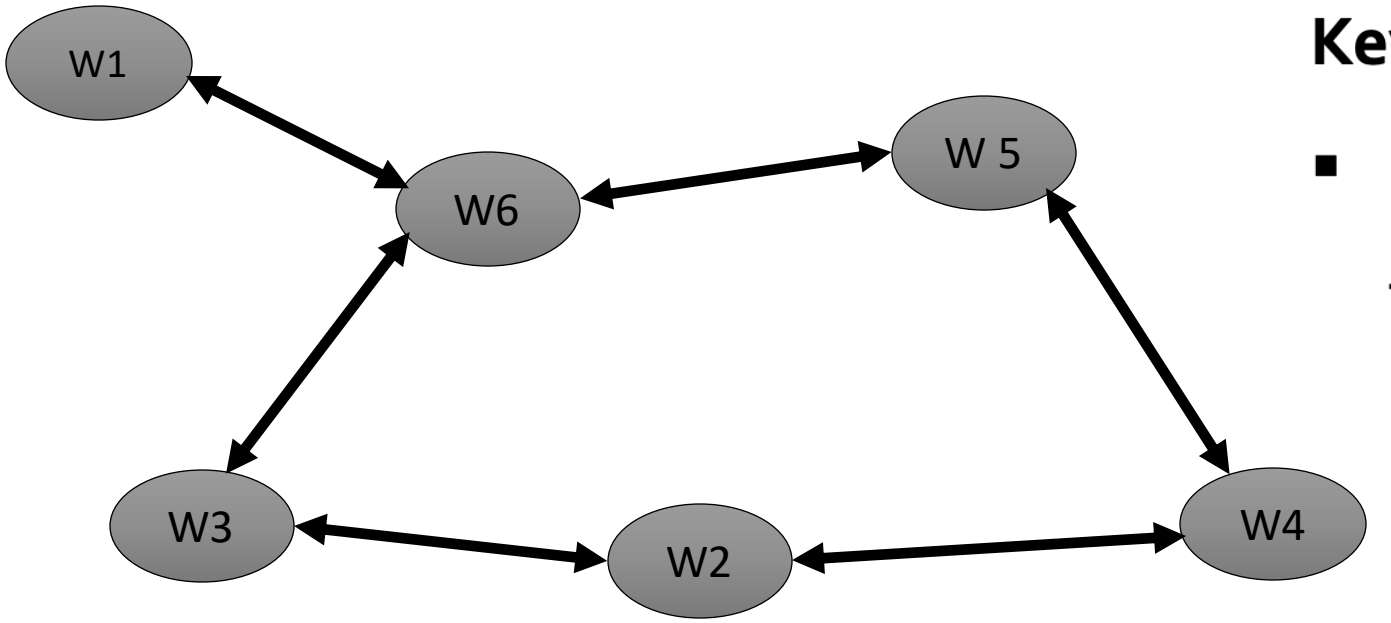
Homework 1 in part 3 of the course

- You will be modifying the code that uses minibatch-SGD to use local SGD (Federated averaging)
  - In federated averaging:
    - the parameter server shares a global model with the workers
    - each worker will start with that model and perform number of local SGD iterations
    - Each worker shares its new model with the parameter server
    - The parameter server will average all local models to produce the new global model

Decentralized-based Learning problem:

Unconstrained formulation:

$$\min_{\Theta} F(\Theta), \quad F(\Theta) := \sum_{n=1}^{N} f_n(\Theta)$$



Decentralized SGD:

$$\mathbf{x}_{k+1}^{(i)} = \sum_{j \in \mathcal{N}_i} W_{ji} \left[\mathbf{x}_k^{(j)} - \eta g(\mathbf{x}_k^{(j)}; \xi_k^{(j)}) \right]$$

Key Idea:

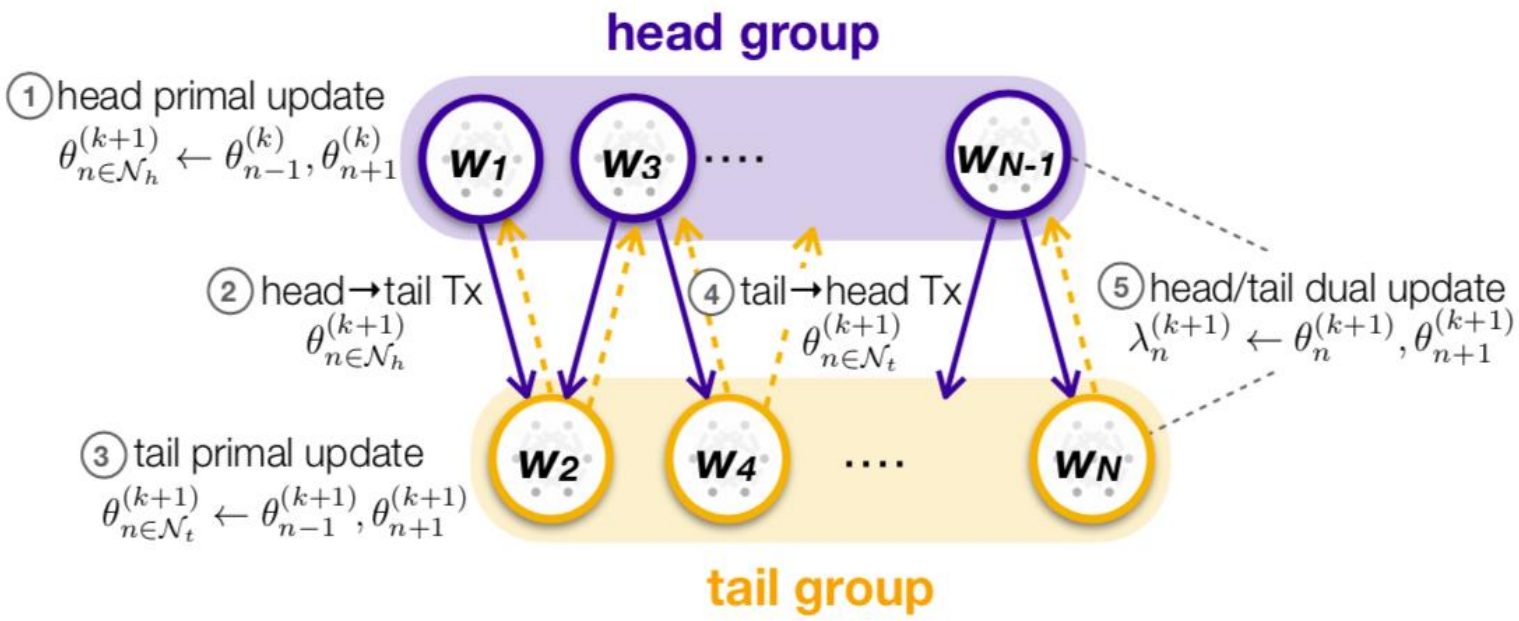
- Each worker requires information from **very few neighbors**

Spatial sparsity

Decentralized-based Learning problem: Our solution

Constrained formulation:

$$\boldsymbol{\theta}^\star := \arg \min_{\{\boldsymbol{\theta}_n\}_{n=1}^N} \sum_{n=1}^N f_n(\boldsymbol{\theta}_n)$$
$$\text{s.t. } \boldsymbol{\theta}_n = \boldsymbol{\theta}_{n+1}, \quad n = 1, \cdots, N-1.$$



Group ADMM (GADMM)

- **Divide workers into two groups**
- **Workers belong to the same group update their models in parallel**
- **The two groups alternate in their updates**
- **Each worker communicates with at most two neighbors**

GADMM

Augmented Lagrangian:

$$\mathcal{L}_\rho(\{\boldsymbol{\theta}_n\}_{n=1}^N, \boldsymbol{\lambda}) = \sum_{n=1}^{N} f_n(\boldsymbol{\theta}_n) + \sum_{n=1}^{N-1} \langle \boldsymbol{\lambda}_n, \boldsymbol{\theta}_n - \boldsymbol{\theta}_{n+1} \rangle + \frac{\rho}{2} \sum_{n=1}^{N-1} \|\boldsymbol{\theta}_n - \boldsymbol{\theta}_{n+1}\|^2$$

Primal variables of head workers:

$$\boldsymbol{\theta}_n^{k+1} = \arg\min_{\boldsymbol{\theta}_n} \Big[f_n(\boldsymbol{\theta}_n) + \langle \boldsymbol{\lambda}_{n-1}^k, \boldsymbol{\theta}_{n-1}^k - \boldsymbol{\theta}_n \rangle + \langle \boldsymbol{\lambda}_n^k, \boldsymbol{\theta}_n - \boldsymbol{\theta}_{n+1}^k \rangle + \frac{\rho}{2} \|\boldsymbol{\theta}_{n-1}^k - \boldsymbol{\theta}_n\|^2 + \frac{\rho}{2} \|\boldsymbol{\theta}_n - \boldsymbol{\theta}_{n+1}^k\|^2 \Big], n \in \mathcal{N}_h \setminus \{1\}$$

$$\boldsymbol{\theta}_n^{k+1} = \arg\min_{\boldsymbol{\theta}_n} \Big[f_n(\boldsymbol{\theta}_n) + \langle \boldsymbol{\lambda}_n^k, \boldsymbol{\theta}_n - \boldsymbol{\theta}_{n+1}^k \rangle + \frac{\rho}{2} \|\boldsymbol{\theta}_n - \boldsymbol{\theta}_{n+1}^k\|^2 \Big], n = 1$$

Primal variables of tail workers:

$$\boldsymbol{\theta}_n^{k+1} = \arg\min_{\boldsymbol{\theta}_n} \Big[f_n(\boldsymbol{\theta}_n) + \langle \boldsymbol{\lambda}_{n-1}^k, \boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_n \rangle + \langle \boldsymbol{\lambda}_n^k, \boldsymbol{\theta}_n - \boldsymbol{\theta}_{n+1}^{k+1} \rangle + \frac{\rho}{2} \|\boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_n\|^2 + \frac{\rho}{2} \|\boldsymbol{\theta}_n - \boldsymbol{\theta}_{n+1}^{k+1}\|^2 \Big], n \in \mathcal{N}_t \setminus \{N\}.$$

$$\boldsymbol{\theta}_n^{k+1} = \arg\min_{\boldsymbol{\theta}_n} \Big[f_n(\boldsymbol{\theta}_n) + \langle \boldsymbol{\lambda}_{n-1}^k, \boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_n \rangle + \frac{\rho}{2} \|\boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_n\|^2 \Big], n = N.$$

Dual variables (updated locally, zero communication overhead):

$$\boldsymbol{\lambda}_n^{k+1} = \boldsymbol{\lambda}_n^k + \rho(\boldsymbol{\theta}_n^{k+1} - \boldsymbol{\theta}_{n+1}^{k+1}), n = \{1, \cdots, N-1\}$$

Optimality of GADMM for convex loss functions

Necessary and sufficient optimality conditions (Boyd et al., 2011):

$$\boldsymbol{\theta}_n^\star = \boldsymbol{\theta}_{n-1}^\star, n \in \{2, \cdots, N\} \qquad \text{(primal feasibility)}$$

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^\star) - \boldsymbol{\lambda}_{n-1}^\star + \boldsymbol{\lambda}_n^\star, n \in \{2, \cdots, N-1\}$$
$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^\star) + \boldsymbol{\lambda}_n^\star, n = 1 \qquad \text{(dual feasibility)}$$
$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^\star) + \boldsymbol{\lambda}_{n-1}^\star, n = N$$

Tail workers: At iteration k+1, from the first order optimality condition it holds that:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \boldsymbol{\lambda}_{n-1}^k + \boldsymbol{\lambda}_n^k + \rho(\boldsymbol{\theta}_n^{k+1} - \boldsymbol{\theta}_{n-1}^{k+1}) + \rho(\boldsymbol{\theta}_n^{k+1} - \boldsymbol{\theta}_{n+1}^{k+1}) \qquad n \in \mathcal{N}_t \setminus \{N\}$$

Can be rewritten as:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \left(\boldsymbol{\lambda}_{n-1}^k + \rho(\boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_n^{k+1})\right) + \left(\boldsymbol{\lambda}_n^k + \rho(\boldsymbol{\theta}_n^{k+1} - \boldsymbol{\theta}_{n+1}^{k+1})\right)$$

Can be rewritten as:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \boldsymbol{\lambda}_{n-1}^{k+1} + \boldsymbol{\lambda}_n^{k+1}, n \in \mathcal{N}_t \setminus \{N\}$$

Following the same steps for the edge worker (worker N) considering the left neighbo

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \boldsymbol{\lambda}_{n-1}^{k+1}, n = N.$$

Dual feasibility condition for tail workers is satisfied at every iteration

Optimality of GADMM for convex loss functions

Head workers: At iteration k+1, from the first order optimality condition it holds that:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \boldsymbol{\lambda}_{n-1}^k + \boldsymbol{\lambda}_n^k + \rho(\boldsymbol{\theta}_n^{k+1} - \boldsymbol{\theta}_{n-1}^k) + \rho(\boldsymbol{\theta}_n^{k+1} - \boldsymbol{\theta}_{n+1}^k) \qquad n \in \mathcal{N}_h \setminus \{1\}$$

Can be rewritten as:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \left(\boldsymbol{\lambda}_{n-1}^k + \rho(\boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_n^{k+1})\right) + \left(\boldsymbol{\lambda}_n^k + \rho(\boldsymbol{\theta}_n^{k+1} - \boldsymbol{\theta}_{n+1}^{k+1})\right) + \rho(\boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_{n-1}^k) + \rho(\boldsymbol{\theta}_{n+1}^{k+1} - \boldsymbol{\theta}_{n+1}^k)$$

Can be rewritten as:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \boldsymbol{\lambda}_{n-1}^{k+1} + \boldsymbol{\lambda}_n^{k+1} + \rho(\boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_{n-1}^k) + \rho(\boldsymbol{\theta}_{n+1}^{k+1} - \boldsymbol{\theta}_{n+1}^k)$$

Following the same steps for the edge worker (worker 1) considering the right neighbor only, gives

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) + \boldsymbol{\lambda}_n^{k+1} + \rho(\boldsymbol{\theta}_{n+1}^{k+1} - \boldsymbol{\theta}_{n+1}^k).$$

Dual and Primal residual at iteration k+1 are defined as follows:

Dual residual

$$\mathbf{s}_n^{k+1} = \begin{cases} \rho(\boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_{n-1}^k) + \rho(\boldsymbol{\theta}_{n+1}^{k+1} - \boldsymbol{\theta}_{n+1}^k), & \text{for } n \in \mathcal{N}_h \setminus \{1\} \\ \rho(\boldsymbol{\theta}_{n+1}^{k+1} - \boldsymbol{\theta}_{n+1}^k), & \text{for } n = 1. \end{cases}$$

Primal residual

$$\mathbf{r}_{n,n+1}^{k+1} = \boldsymbol{\theta}_n^{k+1} - \boldsymbol{\theta}_{n+1}^{k+1}$$

Optimality of GADMM for convex loss functions

To prove the convergence and the optimality of GADMM, we need to show that:

$\mathbf{r}_{n,n+1}^k$, and $\boldsymbol{s}_n^k$ converge to 0, and $\sum_{n=1}^N f_n(\boldsymbol{\theta}_n^k)$ converges to $\sum_{n=1}^N f_n(\boldsymbol{\theta}^\star)$as $k \to \infty$

Lemma 1 *For the iterates $\boldsymbol{\theta}_n^{k+1}$ generated by Algorithm 1, we have*

(i) Upper bound on the optimality gap

$$\sum_{n=1}^{N}[f_n(\boldsymbol{\theta}_n^{k+1}) - f_n(\boldsymbol{\theta}^\star)] \leq -\sum_{n=1}^{N-1}\langle \boldsymbol{\lambda}_n^{k+1}, \mathbf{r}_{n,n+1}^{k+1}\rangle + \sum_{n\in\mathcal{N}_h}\langle \mathbf{s}_n^{k+1}, \boldsymbol{\theta}_n^\star - \boldsymbol{\theta}_n^{k+1}\rangle.$$

(ii) Lower bound on the optimality gap

$$\sum_{n=1}^{N}[f_n(\boldsymbol{\theta}_n^{k+1}) - f_n(\boldsymbol{\theta}^\star)] \geq -\sum_{n=1}^{N-1}\langle \boldsymbol{\lambda}_n^\star, \mathbf{r}_{n,n+1}^{k+1}\rangle.$$

Optimality of GADMM for convex loss functions

Theorem 2 *When $f_n(\boldsymbol{\theta}_n)$ is closed, proper, and convex for all n, and the Lagrangian $\mathcal{L}_0$ has a saddle point, for GADMM iterates, it holds that*

*(i) the primal residual converges to zero as $k \to \infty$.*i.e.,

$$\lim_{k\to\infty} \mathbf{r}_{n,n+1}^k = \mathbf{0}, n \in \{1, \cdots, N-1\}.$$

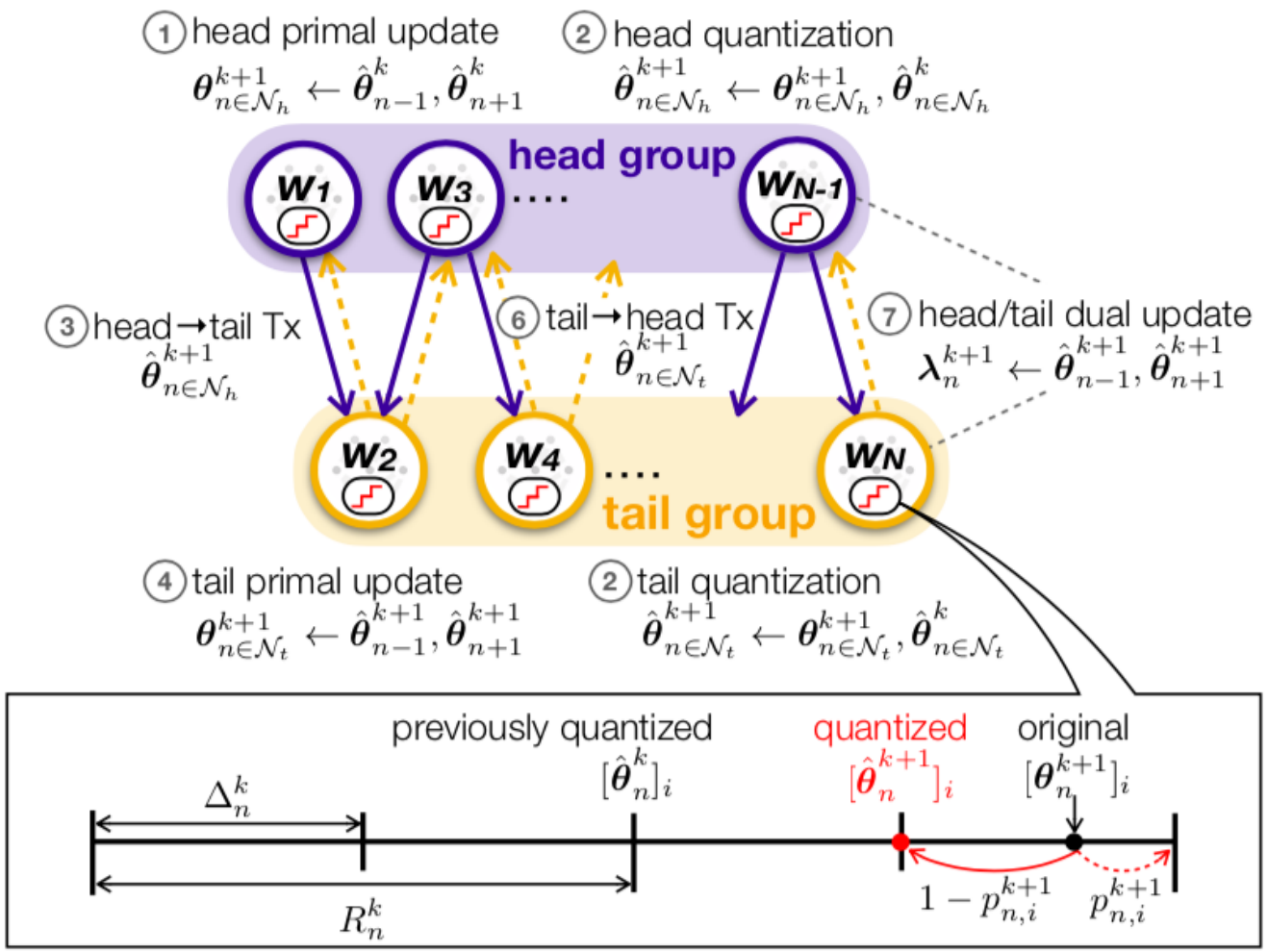
*(ii) the dual residual converges to zero as $k \to \infty$.*i.e.,

$$\lim_{k\to\infty} \mathbf{s}_n^k = \mathbf{0}, n \in \mathcal{N}_h.$$

*(iii) the optimality gap converges to zero as $k \to \infty$.*i.e.,

$$\lim_{k\to\infty} \sum_{n=1}^{N} f_n(\boldsymbol{\theta}_n^k) = \sum_{n=1}^{N} f_n(\boldsymbol{\theta}^\star).$$

Q-GADMM: Quantized Group ADMM



Q-GADMM: Stochastic Quantization

$$\hat{\boldsymbol{\theta}}_n^k = Q_n(\boldsymbol{\theta}_n^k, \hat{\boldsymbol{\theta}}_n^{k-1}) \quad \Delta_n^k = 2R_n^k/(2^{b_n^k} - 1)$$

$$[c_n(\boldsymbol{\theta}_n^k)]_i = \frac{1}{\Delta_n^k}\left([\boldsymbol{\theta}_n^k]_i - [\hat{\boldsymbol{\theta}}_n^{k-1}]_i + R_n^k\right)$$

$$[q_n(\boldsymbol{\theta}_n^k)]_i = \begin{cases} \lceil [c_n(\boldsymbol{\theta}_n^k)]_i \rceil & \text{with probability } p_{n,i}^k \\ \lfloor [c_n(\boldsymbol{\theta}_n^k)]_i \rfloor & \text{otherwise,} \end{cases}$$

Q-GADMM: Stochastic Quantization

To maintain unbiased error with mean zero:

$$p_{n,i}^k \left([c_n(\boldsymbol{\theta}_n^k)]_i - \lceil [c_n(\boldsymbol{\theta}_n^k)]_i \rceil\right) + (1-p_{n,i}^k)\left([c_n(\boldsymbol{\theta}_n^k)]_i - \lfloor [c_n(\boldsymbol{\theta}_n^k)]_i \rfloor\right) = 0$$

Hence:

$$p_{n,i}^k = \left([c_n(\boldsymbol{\theta}_n^k)]_i - \lfloor [c_n(\boldsymbol{\theta}_n^k)]_i \rfloor\right)$$

To guarantee convergence to the optimal solution as k → ∞, the following inequality should be satisfied:

$$\Delta_n^k \leq \Delta_n^{k-1}$$

Therefore, b_n^k, should satisfy:

$$b_n^k \geq \left\lceil \log_2\left(1 + (2^{b_n^{k-1}} - 1) R_n^k / R_n^{k-1}\right)\right\rceil$$

The quantized model is:

$$\hat{\boldsymbol{\theta}}_n^k = \hat{\boldsymbol{\theta}}_n^{k-1} + \Delta_n^k q_n(\boldsymbol{\theta}_n^k) - R_n^k \mathbf{1}$$

Q-GADMM

Augmented Lagrangian:

$$\mathcal{L}_\rho = \sum_{n=1}^{N} f_n(\boldsymbol{\theta}_n) + \sum_{n=1}^{N-1} \langle \boldsymbol{\lambda}_n, \boldsymbol{\theta}_n - \hat{\boldsymbol{\theta}}_{n+1} \rangle + \frac{\rho}{2} \sum_{n=1}^{N-1} \|\boldsymbol{\theta}_n - \hat{\boldsymbol{\theta}}_{n+1}\|_2^2$$

The primal variables of the head workers:

$$\boldsymbol{\theta}_{n\in\mathcal{N}_h}^{k+1} = \underset{\theta_n}{\operatorname{argmin}} \{ f_n(\boldsymbol{\theta}_n) + \langle \boldsymbol{\lambda}_{n-1}^k, \hat{\boldsymbol{\theta}}_{n-1}^k - \boldsymbol{\theta}_n \rangle + \langle \boldsymbol{\lambda}_n^k, \boldsymbol{\theta}_n - \hat{\boldsymbol{\theta}}_{n+1}^k \rangle$$

$$+ \frac{\rho}{2} \|\hat{\boldsymbol{\theta}}_{n-1}^k - \boldsymbol{\theta}_n\|_2^2 + \frac{\rho}{2} \|\boldsymbol{\theta}_n - \hat{\boldsymbol{\theta}}_{n+1}^k\|_2^2 \}, n \in \mathcal{N}_h \setminus \{1\}.$$

The primal variables of the tail workers:

$$\boldsymbol{\theta}_{n\in\mathcal{N}_t}^{k+1} = \underset{\boldsymbol{\theta}_n}{\operatorname{argmin}} \{ f_n(\boldsymbol{\theta}_n) + \langle \boldsymbol{\lambda}_{n-1}^k, \hat{\boldsymbol{\theta}}_{n-1}^{k+1} - \boldsymbol{\theta}_n \rangle + \langle \boldsymbol{\lambda}_n^k, \boldsymbol{\theta}_n - \hat{\boldsymbol{\theta}}_{n+1}^{k+1} \rangle$$

$$+ \frac{\rho}{2} \|\hat{\boldsymbol{\theta}}_{n-1}^{k+1} - \boldsymbol{\theta}_n\|_2^2 + \frac{\rho}{2} \|\boldsymbol{\theta}_n - \hat{\boldsymbol{\theta}}_{n+1}^{k+1}\|_2^2 \}, n \in \mathcal{N}_h \setminus \{N\}.$$

The dual variables:

$$\boldsymbol{\lambda}_n^{k+1} = \boldsymbol{\lambda}_n^k + \rho(\hat{\boldsymbol{\theta}}_n^{k+1} - \hat{\boldsymbol{\theta}}_{n+1}^{k+1}),\ n = 1, \ldots, N-1$$

Q-GADMM: Convergence Analysis

Primal and dual visibility:

$$\boldsymbol{\theta}_n^* = \boldsymbol{\theta}_{n-1}^*, \forall n > 1, \quad \mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^*) - \boldsymbol{\lambda}_{n-1}^* + \boldsymbol{\lambda}_n^*$$

Tail workers: At iteration k+1, from the first order optimality condition it holds that:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \boldsymbol{\lambda}_{n-1}^k + \boldsymbol{\lambda}_n^k + \rho(\boldsymbol{\theta}_n^{k+1} - \hat{\boldsymbol{\theta}}_{n-1}^{k+1}) + \rho(\boldsymbol{\theta}_n^{k+1} - \hat{\boldsymbol{\theta}}_{n+1}^{k+1})$$

Given:

$$\boldsymbol{\epsilon}_n^{k+1} = \boldsymbol{\theta}_n^{k+1} - \hat{\boldsymbol{\theta}}_n^{k+1} \quad \textbf{and} \quad \boldsymbol{\lambda}_n^{k+1} = \boldsymbol{\lambda}_n^k + \rho(\hat{\boldsymbol{\theta}}_n^{k+1} - \hat{\boldsymbol{\theta}}_{n+1}^{k+1})$$

The above equation can be rewritten as:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \boldsymbol{\lambda}_{n-1}^{k+1} + \boldsymbol{\lambda}_n^{k+1} + 2\rho\boldsymbol{\epsilon}_n^{k+1}, n \in \mathcal{N}_t \setminus \{N\}$$

Head workers: At iteration k+1, from the first order optimality condition it holds that:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \boldsymbol{\lambda}_{n-1}^k + \boldsymbol{\lambda}_n^k + \rho(\boldsymbol{\theta}_n^{k+1} - \hat{\boldsymbol{\theta}}_{n-1}^k) + \rho(\boldsymbol{\theta}_n^{k+1} - \hat{\boldsymbol{\theta}}_{n+1}^k) \quad n \in \mathcal{N}_h \setminus \{1\}$$

Given:

$$\boldsymbol{\epsilon}_n^{k+1} = \boldsymbol{\theta}_n^{k+1} - \hat{\boldsymbol{\theta}}_n^{k+1}$$

and

$$\boldsymbol{\lambda}_n^{k+1} = \boldsymbol{\lambda}_n^k + \rho(\hat{\boldsymbol{\theta}}_n^{k+1} - \hat{\boldsymbol{\theta}}_{n+1}^{k+1})$$

Can be rewritten as:

$$\mathbf{0} \in \partial f_n(\boldsymbol{\theta}_n^{k+1}) - \boldsymbol{\lambda}_{n-1}^{k+1} + \boldsymbol{\lambda}_n^{k+1} + 2\rho\boldsymbol{\epsilon}_n^{k+1}$$
$$+ \rho(\hat{\boldsymbol{\theta}}_{n-1}^{k+1} - \hat{\boldsymbol{\theta}}_{n-1}^k) + \rho(\hat{\boldsymbol{\theta}}_{n+1}^{k+1} - \hat{\boldsymbol{\theta}}_{n+1}^k), n \in \mathcal{N}_h \setminus \{1\}$$

Dual and Primal residual at iteration k+1 are defined as follows:

Dual residual

$$\boldsymbol{s}_n^{k+1} = \begin{cases} \rho(\hat{\boldsymbol{\theta}}_{n-1}^{k+1} - \hat{\boldsymbol{\theta}}_{n-1}^k) + \rho(\hat{\boldsymbol{\theta}}_{n+1}^{k+1} - \hat{\boldsymbol{\theta}}_{n+1}^k), & \text{if } n \in \mathcal{N}_h \setminus \{1\} \\ \rho(\hat{\boldsymbol{\theta}}_{n+1}^{k+1} - \hat{\boldsymbol{\theta}}_{n+1}^k), & \text{if } n = 1. \end{cases}$$

Primal residual

$$\mathbf{r}_{n,n+1}^{k+1} = \boldsymbol{\theta}_n^{k+1} - \boldsymbol{\theta}_{n+1}^{k+1}$$

Lemma 1 At $k+1$ iteration of Q-GADMM, the optimality gap satisfies $\mathsf{LB}_1 \leq \mathbb{E}\left[\left\{\sum_{n=1}^{N} f_n(\boldsymbol{\theta}_n^{k+1}) - \sum_{n=1}^{N} f_n(\boldsymbol{\theta}_n^{*})\right\}\right] \leq \mathsf{UB}_1$

$$\mathsf{LB}_1 = \mathbb{E}\left[\sum_{n=1}^{N-1} \langle \boldsymbol{\lambda}_n^{*}, \boldsymbol{r}_{n,n+1}^{k+1} \rangle\right]$$

$$\mathsf{UB}_1 = \mathbb{E}\left[-\sum_{n=1}^{N-1} \langle \boldsymbol{\lambda}_n^{k+1}, \boldsymbol{r}_{n,n+1}^{k+1} \rangle + \sum_{n=1}^{N-1} 2\rho \langle \boldsymbol{\epsilon}_n^{k+1}, \boldsymbol{\theta}_n^{*} - \boldsymbol{\theta}_n^{k+1} \rangle + \sum_{n \in \mathcal{N}_h} \langle \boldsymbol{s}_n^{k+1}, \boldsymbol{\theta}_n^{*} - \boldsymbol{\theta}_n^{k+1} \rangle\right]$$

$$V_k = 1/\rho \sum_{n=1}^{N-1} \left\| \boldsymbol{\lambda}_n^k - \boldsymbol{\lambda}_n^\star \right\|^2 + \rho \sum_{n \in \mathcal{N}_h \setminus \{1\}} \left\| \boldsymbol{\theta}_{n-1}^k - \boldsymbol{\theta}^\star \right\|^2 + \rho \sum_{n \in \mathcal{N}_h} \left\| \boldsymbol{\theta}_{n+1}^k - \boldsymbol{\theta}^\star \right\|^2$$

Lemma 2 When $f_n(\boldsymbol{\theta}_n)$ is closed, proper, and convex, and the Lagrangian $\boldsymbol{\mathcal{L}}_0$ has a saddle point, then the following inequality holds true at the $(k+1)$-th iteration of Q-GADMM:

$$\mathbb{E}\left[V^k - V^{k+1}\right] \geq \rho \sum_n \mathbb{E}\left[\| \boldsymbol{\epsilon}_n^{k+1} + \boldsymbol{\epsilon}_{n+1}^{k+1} \|_2^2\right] - \frac{\rho d}{2} \sum_{n \in \mathcal{N}_t} (\Delta_n^{k+1})^2 + H_v$$

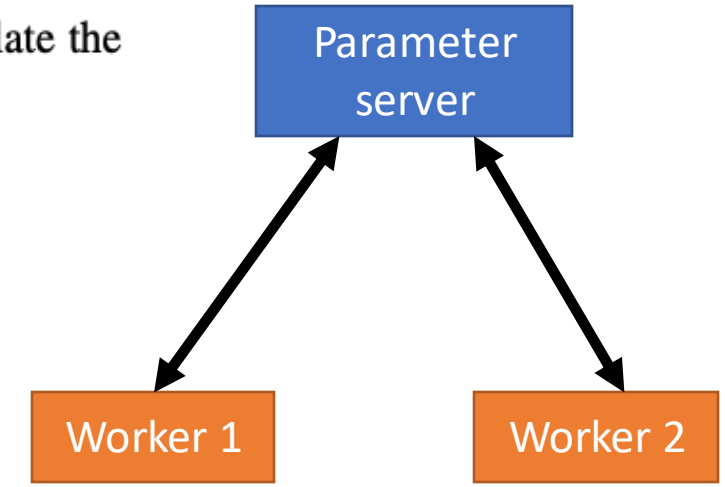
$$H_v = \rho \sum_{n \in \mathcal{N}_h \setminus \{1\}} \mathbb{E}\left[\| (\boldsymbol{\theta}_{n-1}^{k+1} - \boldsymbol{\theta}_{n-1}^k) - (\boldsymbol{\epsilon}_{n-1}^{k+1} - \boldsymbol{\epsilon}_{n-1}^k) - \boldsymbol{r}_{n-1,n}^{k+1} \|_2^2\right] + \rho \sum_{n \in \mathcal{N}_h} \mathbb{E}\left[\| (\boldsymbol{\theta}_{n+1}^{k+1} - \boldsymbol{\theta}_{n+1}^k) - (\boldsymbol{\epsilon}_{n+1}^{k+1} - \boldsymbol{\epsilon}_{n+1}^k) + \boldsymbol{r}_{n,n+1}^{k+1} \|_2^2\right]$$
$$+ 2\rho \left(\mathbb{E}\left[\| \boldsymbol{\epsilon}_1^{k+1} \|_2^2\right] + \mathbb{E}\left[\| \boldsymbol{\epsilon}_N^{k+1} \|_2^2\right]\right) + 2\rho \sum_{n \in \mathcal{N}_t} \mathbb{E}\left[\| \boldsymbol{\epsilon}_n^k \|_2^2\right]$$

Theorem 1 For non-increasing quantization step sizes, *i.e.*, $\Delta_n^k \leq \Delta_n^{k-1}$ $\forall k$, and under the assumption of Lemma 2, as $k \to \infty$, the primal and dual residual converges to 0 with probability 1, *i.e.*, $\lim_{k\to\infty} \boldsymbol{r}_{n,n+1}^k \overset{a.s.}{=} \mathbf{0}$ and $\lim_{k\to\infty} \boldsymbol{s}_n^k \overset{a.s.}{=} \mathbf{0}$. Furthermore, the optimality gap converges to 0 with probability 1, *i.e.*, $\lim_{k\to\infty} \sum_{n=1}^N f_n(\boldsymbol{\theta}_n^k) = \sum_{n=1}^N f_n(\boldsymbol{\theta}^*)$.

Intuitively, when Δ_n^k is non-increasing, the RHS of (23) is always positive. For such positive $\mathbb{E}\left[V^k - V^{k+1}\right]$, iteration $k+1$ is one more step towards the optimal solution. Therefore, following the same proof of theorem 1 for GADMM [17], as $k \to \infty$, Q-GADMM converges to the optimal solution.

Homework 2

Assume a system consists of a parameter server (PS) and two workers. Each worker i is exposed to its local batch of training data $H_i \in R^{mXd}$ and corresponding label vector $Y_i \in R^m$. Workers need to learn a global model $\Theta \in R^d$ that solves a distributed linear regression problem. To this end, we formulate the linear regression learning problem as the following constrained optimization problem.



$$\text{Minimize} \left\{ \frac{1}{2} \parallel Y_1 - H_1\theta_1 \parallel_2^2 + \frac{1}{2} \parallel Y_2 - H_2\theta_2 \parallel_2^2 \right\}$$

$$\text{s.}t$$

$$\theta_1 = \Theta$$

$$\theta_2 = \Theta$$

Assume that we want to solve the formulated problem using distributed ADMM. With that, you are required to:

(i) Write the equation of the augmented Lagrangian

(ii) Find the closed form expression of the update of θ_1, θ_2, and Θ at iteration $k+1$.

(iii) write the updating equation of the dual variables.