PRESENTATION

MASSIVELY PARALLEL MACHINE LEARNING

BOTNET DETECTOR

Elise Maistre, Hugo Garde, Corentin Ibos

JANUARY 2024

INTRODUCTION

CENTRALISED

PARALLELIZED

CROSS VALIDATION

Dataset Description

x_1^1	x_2^1	...	x_{11}^1	Y^1
x_1^2	x_2^2	...	x_{11}^2	Y^2
...	...	...	...	...
$x_{11}^{1\,000\,000}$	$x_{11}^{1\,000\,000}$	...	$x_{11}^{1\,000\,000}$	$Y^{1\,000\,000}$

Centralised Version

ReadFile

Numpy Array

$$\left(\left[x_1^1 , x_2^1 , \dots , x_{11}^1 \right], Y^1 \right)$$

$$\left(\left[x_1^2 , x_2^2 , \dots , x_{11}^2 \right], Y^2 \right)$$

$$\dots$$

$$\left(\left[x_1^i , x_2^i , \dots , x_{11}^i \right], Y^i \right)$$

1 000 000

Normalisation

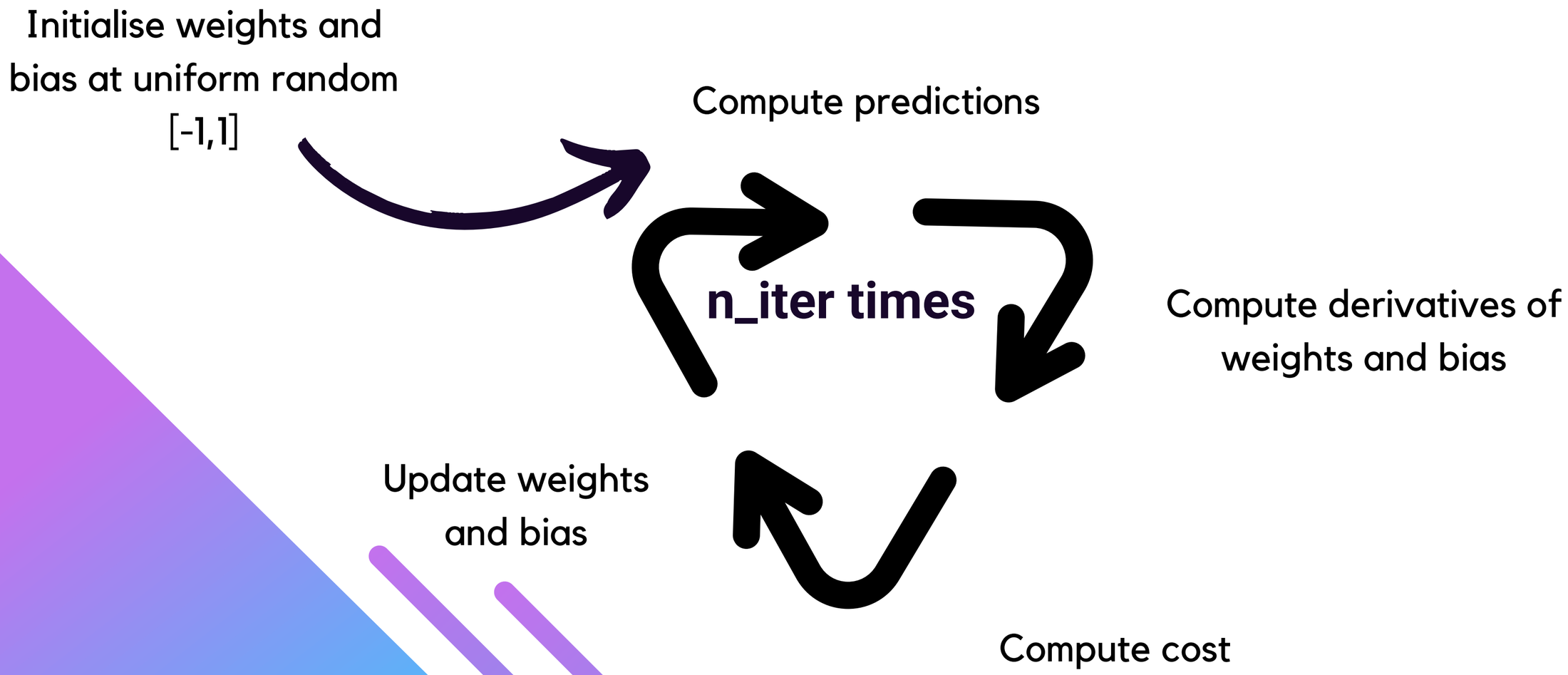
x_1^1
x_1^2
...
$x_1^{1\,000\,000}$

$$\mu_1 = \frac{1}{1\,000\,000} \sum_{k=1}^{1\,000\,000} x_1^k$$

$$\sigma_1 = \sqrt{\frac{1}{1\,000\,000} \sum_{k=1}^{1\,000\,000} (x_1^k - \mu_1)^2}$$

$$x' = \frac{x - \mu_1}{\sigma_1}$$

Training


Initialise weights and bias at uniform random [-1,1]
Compute predictions
n_iter times
Compute derivatives of weights and bias
Update weights and bias
Compute cost

Predict

$$\underbrace{x_1^1, x_2^1, \dots, x_{11}^1}_{\text{X to be evaluated}}$$

$$\underbrace{w_1, w_2, \dots, w_{11}, b}_{\text{Set of weights and bias}}$$

Compute $\hat{Y}$

$$\text{sigmoid}\,(x_1^1 . w_1 + x_2^1 . w_2 + \dots + x_{11}^1 . w_{11} + b)$$

$$0 < \hat{Y} < 1$$

assign 0 or 1 to Y depending on the threshold

Accuracy

- Use the previous function to predict
- Compare with the given label
- Divide the number of right predictions by the total number of predictions

Parallelization

ReadFile

Dataset: 11 features and label (1 000 000 rows)

x_1^1	x_2^1	...	x_{11}^1	Y^1
x_1^2	x_2^2	...	x_{11}^2	Y^2
...	...	...	...	...
$x_1^{1\,000\,000}$	$x_2^{1\,000\,000}$	...	$x_{11}^{1\,000\,000}$	$Y^{1\,000\,000}$

- **Map** : split 12 values for each line by ‘,’
- **Map** : put 11 X into a numpy of float and Y like a int for each line

Result key-value : 11 value, label

$([x_1^1 , x_2^1 , \dots , x_{11}^1], Y^1)$
$([x_1^2 , x_2^2 , \dots , x_{11}^2], Y^2)$
...
$([x_1^i , x_2^i , \dots , x_{11}^i], Y^i)$

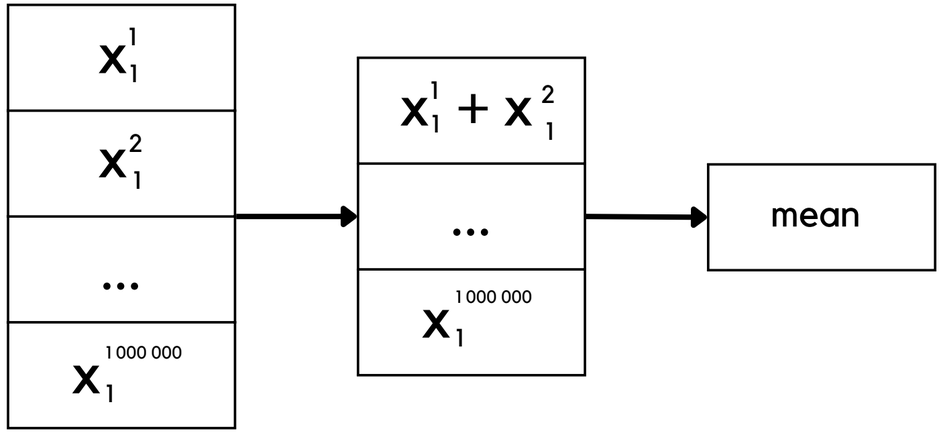
1 000 000

Normalisation

- Map : keep only X^i

x_1^1	x_2^1	...	x_{11}^1
x_1^2	x_2^2	...	x_{11}^2
...	...	...	...
$x_1^{1\,000\,000}$	$x_2^{1\,000\,000}$	...	$x_{11}^{1\,000\,000}$

- Reduce : mean for each column



$$\mu_1 = \frac{1}{1\,000\,000} \sum_{k=1}^{1\,000\,000} x_1^k$$

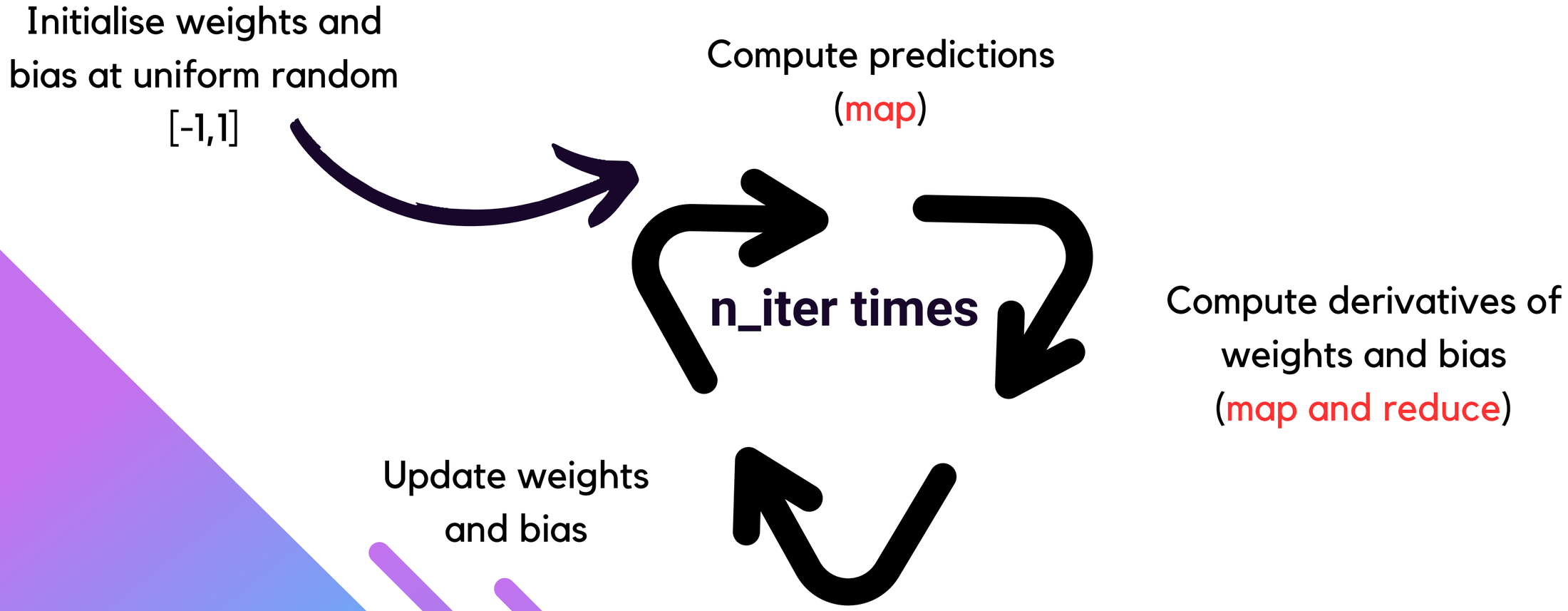
- Map : for each line, each colum separately, do the formula $(x_1^k - \mu_1)^2$
- Reduce : like the mean but with the good map to have the variance

$$V_1 = \frac{1}{1\,000\,000} \sum_{k=1}^{1\,000\,000} (x_1^k - \mu_1)^2$$

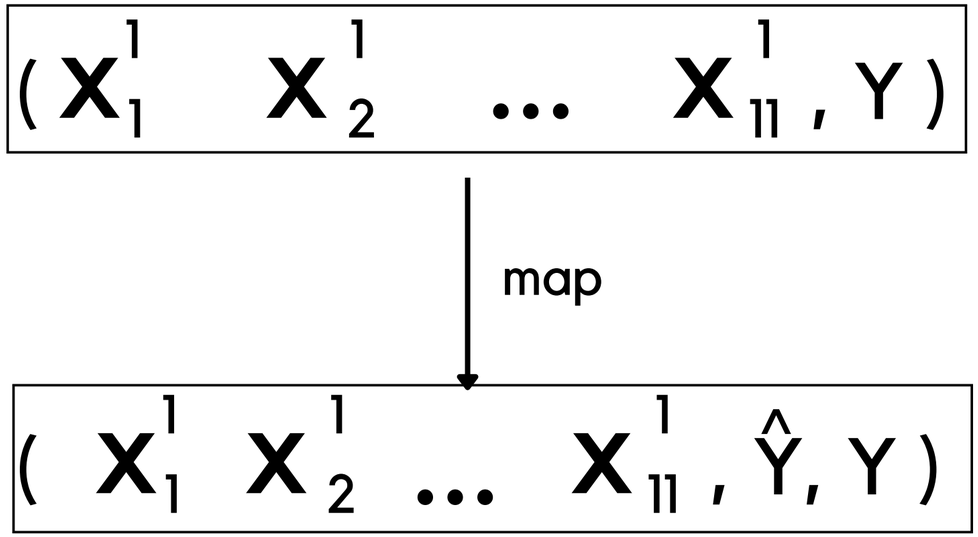
- Map : Normalize X by using mean and variance

$$x' = \frac{x - \mu_1}{\sigma_1}$$

Training


Initialise weights and bias at uniform random [-1,1]
Compute predictions (map)
n_iter times
Compute derivatives of weights and bias (map and reduce)
Update weights and bias
Compute cost (map and reduce)

Training



Compute predictions

$$\hat{Y} = \text{sigmoid}\,(x_1^1 . w_1 + x_2^1 . w_2 + \dots + x_{11}^1 . w_{11} + b)$$

Training

map

$$dw_1 = \frac{1}{m}\sum_{j=1}^{m}\boxed{(\hat{y}^{(j)} - y^{(j)}) * x_1^{(j)}} + \frac{\lambda}{m}w_1$$

$$dw_2 = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_2^{(j)} + \frac{\lambda}{m}w_2$$

$$\dots$$

$$dw_k = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_k^{(j)} + \frac{\lambda}{m}w_k$$

$$db = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)})$$

Compute derivatives

$$(X_1^1 \quad X_2^1 \quad \dots \quad X_{11}^1, \hat{Y}, Y)$$

map

$$[\hat{y}-y*x_1^1, \hat{y}-y*x_2^1, \dots \hat{y}-y*x_{11}^1]$$

Training

reduce

$$dw_1 = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_1^{(j)} + \frac{\lambda}{m}w_1$$

$$dw_2 = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_2^{(j)} + \frac{\lambda}{m}w_2$$

$$\dots$$

$$dw_k = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_k^{(j)} + \frac{\lambda}{m}w_k$$

$$db = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)})$$

Compute derivatives

$(X_1^1 \; X_2^1 \; \dots \; X_{11}^1, \hat{Y}, Y)$

map

$[\hat{y}-y*x_1^1, \hat{y}-y*x_2^1, \dots \hat{y}-y*x_{11}^1]$

reduce

sum

Training

map

$$J(W) = -\frac{1}{m}\sum_{i=1}^{m}\boxed{(y^{(i)}log(\hat{y}^{(i)}) + (1-y^{(i)})log(1-\hat{y}^{(i)}))} + \frac{\lambda}{2m}\sum_{i=1}^{k} w_i^2$$

Compute cost

$$\boxed{(X_1^1 \quad X_2^1 \quad \dots \quad X_{11}^1, \hat{Y}, Y)}$$

map

$$\boxed{[\hat{y}\text{-}y^{*}x_1^1,\ \hat{y}\text{-}y^{*}x_2^1, \dots \hat{y}\text{-}y^{*}x_{11}^1]}$$

Training

reduce

$$J(W) = -\frac{1}{m}\boxed{\sum_{i=1}^{m}(y^{(i)}log(\hat{y}^{(i)}) + (1-y^{(i)})log(1-\hat{y}^{(i)}))} + \frac{\lambda}{2m}\sum_{i=1}^{k} w_i^2$$

Compute cost

$$(X_1^1 \quad X_2^1 \quad \dots \quad X_{11}^1, \hat{Y}, Y)$$

map

$$Y*log(\hat{Y})+(1-Y)*log(1-\hat{Y})$$

reduce

sum

Cross validation

Cross validation

Transform

Pre-processing data => indexing data by adding a key to have K uniformly spread packets of data

random_assigned_rdd =
data.map(lambda x: (rd.randint(0, 10), x))

0	x_1^1	x_2^1	...	x_{784}^1
2	x_1^2	x_2^2	...	x_{784}^2
2	x_1^3	x_2^3	...	x_{784}^3
...	...	...	...	...
8	$x_1^{70\,000}$	$x_1^{70\,000}$	...	$x_{784}^{70\,000}$

Cross validation

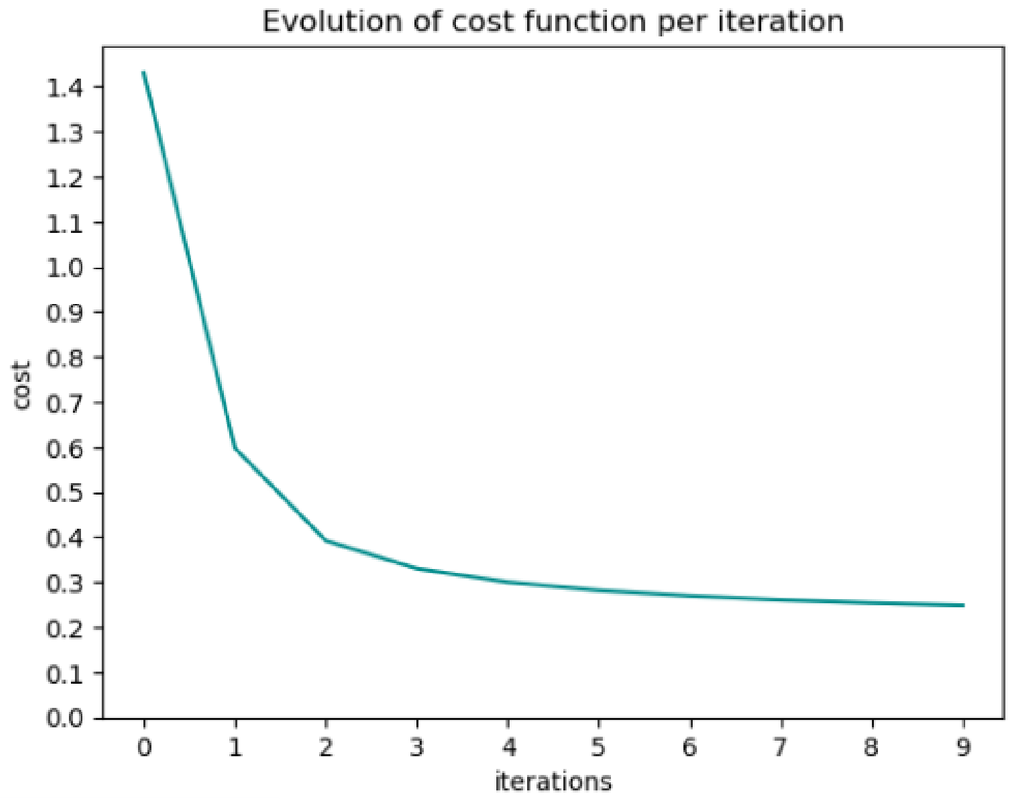
Get block data

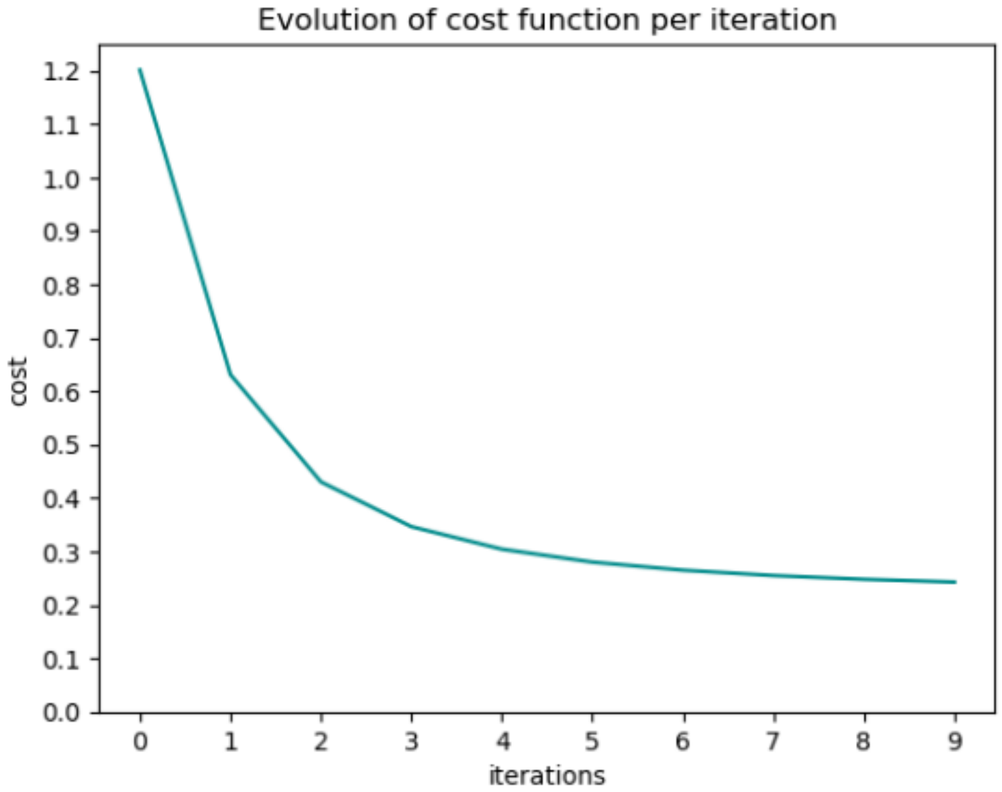
Filter data into 2 RDDS =>
training and test set

```
train_data = data_cv.flatMap(lambda row: [row[1]] if row[0] != index else [])
test_data = data_cv.flatMap(lambda row: [row[1]] if row[0] == index else [])
```

Centralised vs Parallelized

Evolution of cost function

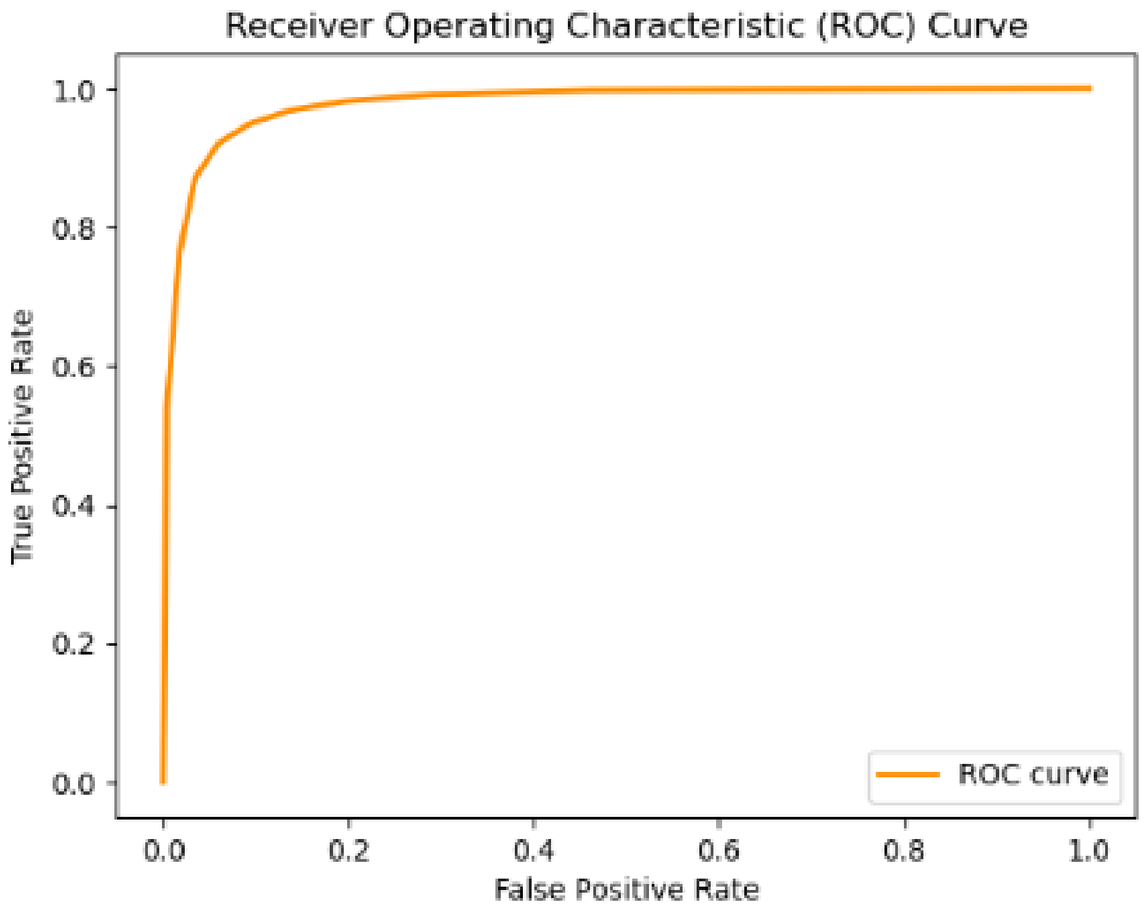


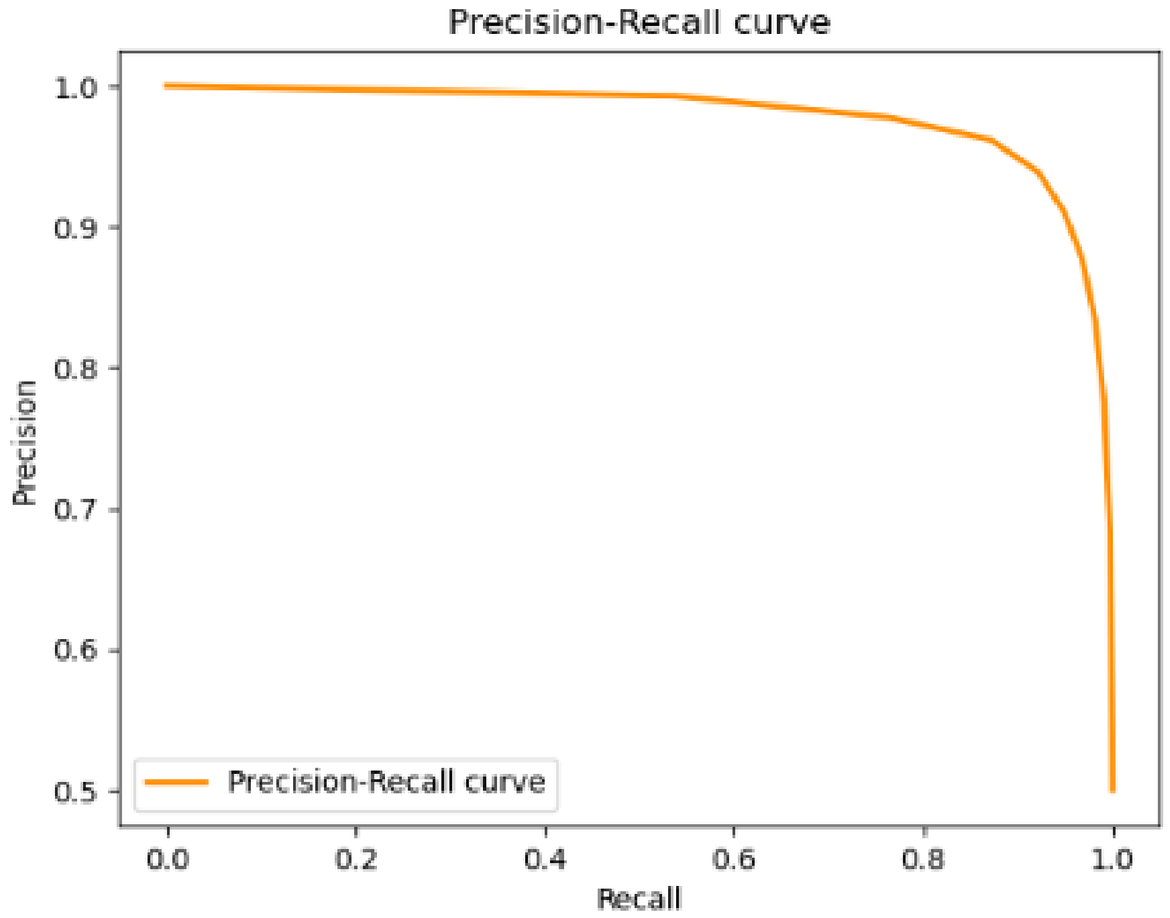


Centralised

Parallelized

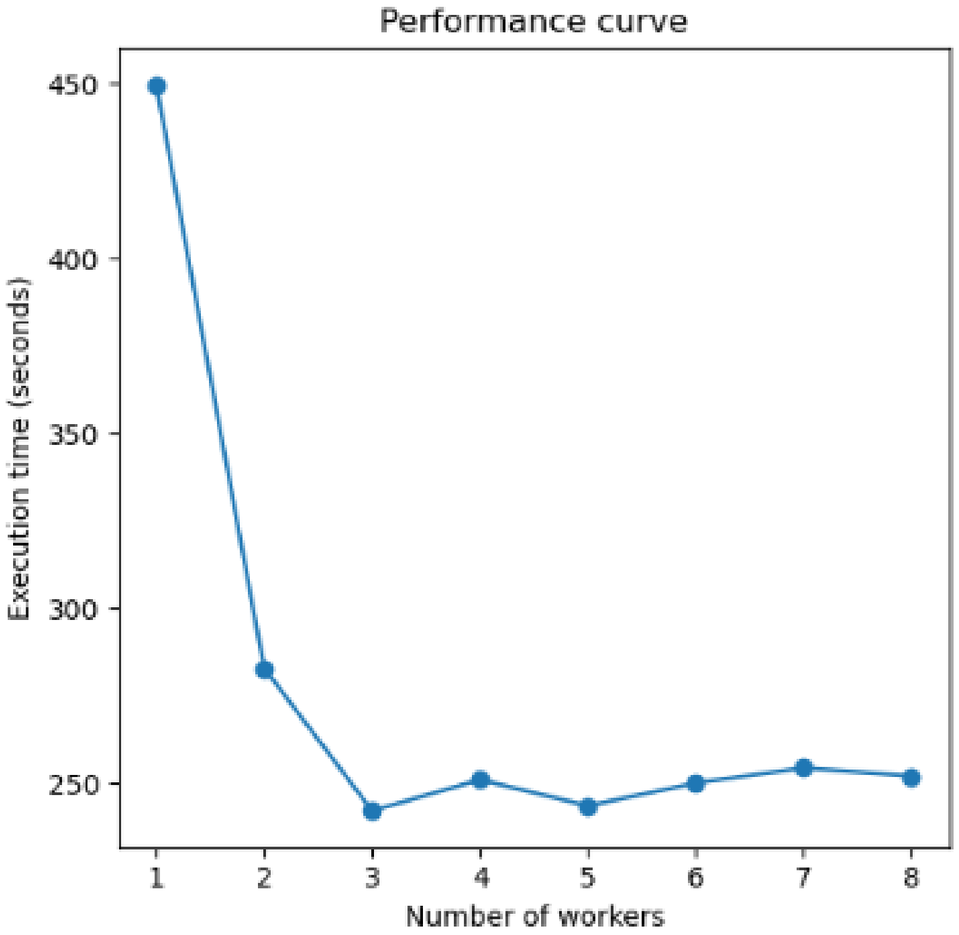
Precision-Recall and ROC curves

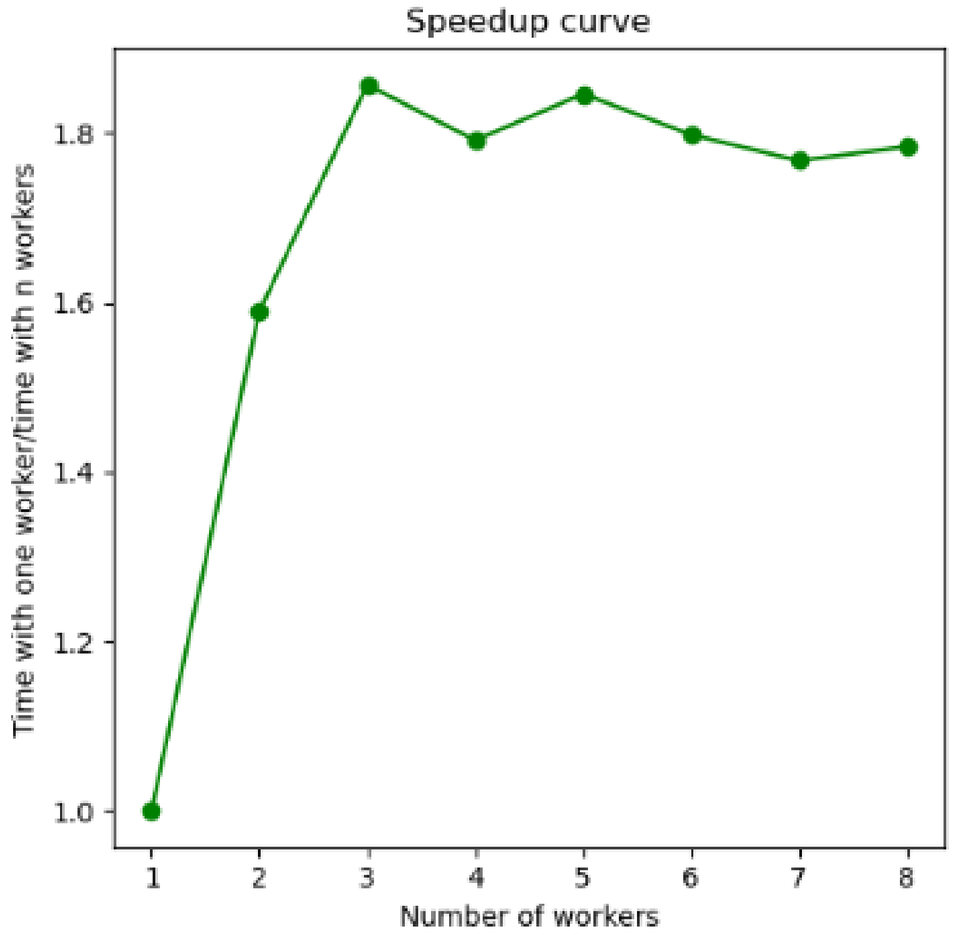




Performance and speedup curve

4 cores





THANK YOU

MNIST DATASET

MASSIVELY PARALLEL MACHINE LEARNING

K-MEANS

Elise Maistre, Hugo Garde, Corentin Ibos

JANUARY 2024

INTRODUCTION

DATASET

CENTRALISED

PARALLELIZED

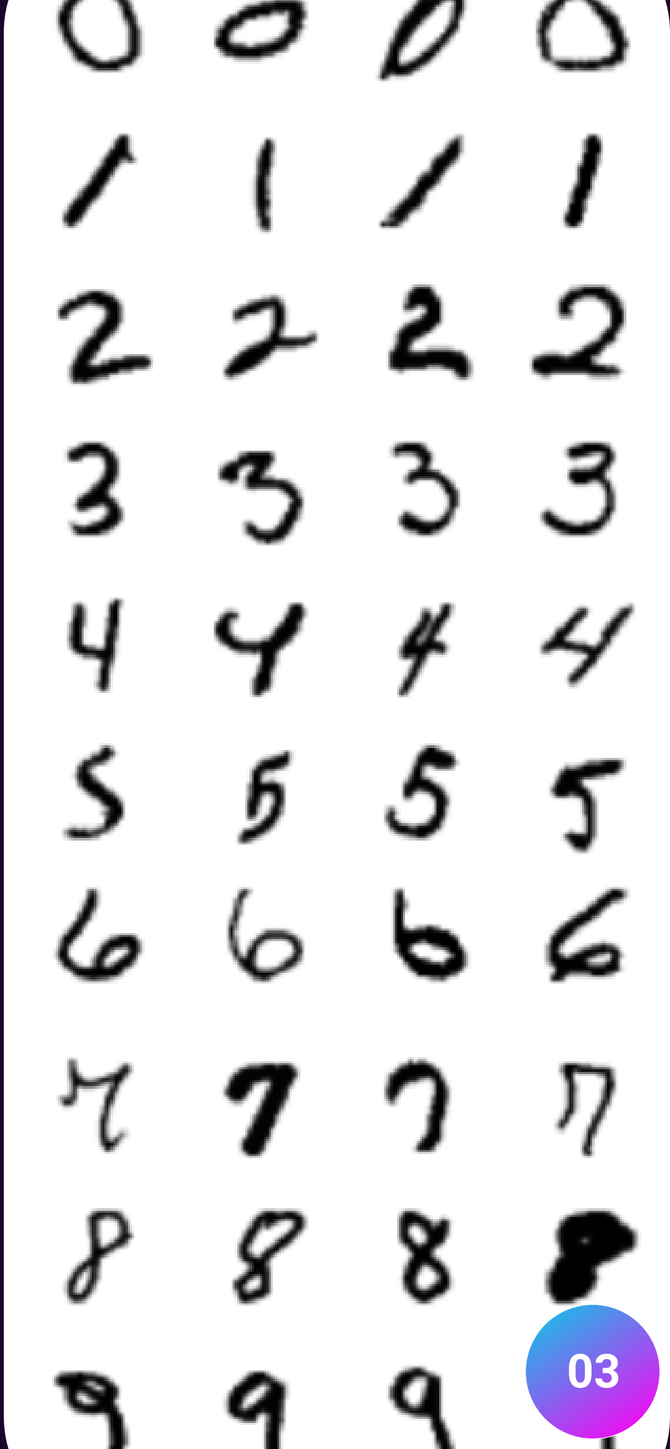
The data

Raw data

70,000 labeled grayscale images of handwritten digits

Formating

28 × 28 matrix for each image flattened into a vector of length 784 (28 × 28 = 784)



Centralised Version

Centralised - ReadFile

- Read a CSV file and create a numpy array out of it
- Useless data: first row and first column

READFILE

LABEL	1x1	1x2	...	28x28
Y^{1}	x_{1}^{1}	x_{2}^{1}	...	x_{784}^{1}
Y^{2}	x_{1}^{2}	x_{2}^{2}	...	x_{784}^{2}
...	...	...	...	...
$Y^{70\,000}$	$x_{1}^{70\,000}$	$x_{1}^{70\,000}$	...	$x_{784}^{70\,000}$

Centralised - ReadFile

- Read a CSV file and create a numpy array out of it
- Useless data: first row and first column

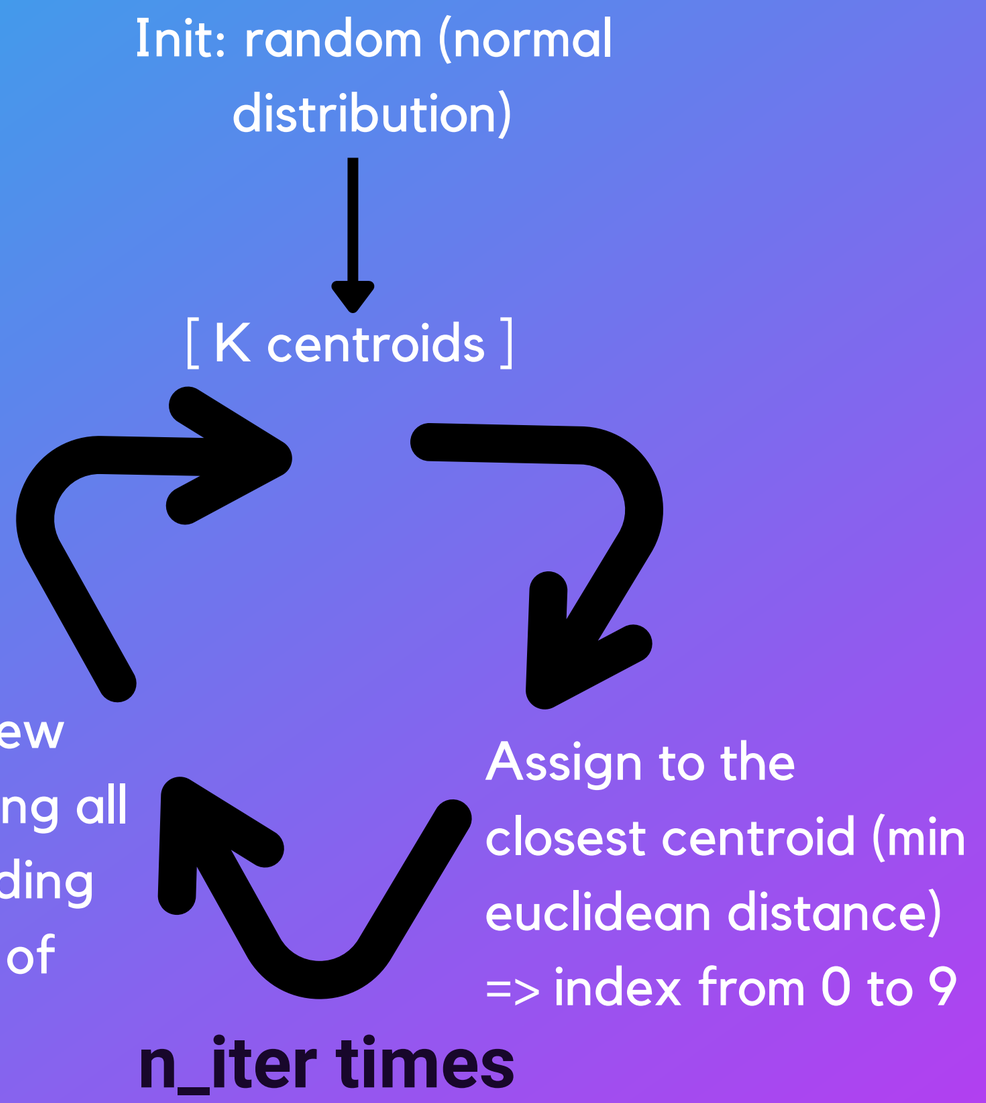
READFILE

LABEL	1x1	1x2	...	28x28
Y^{1}	x_{1}^{1}	x_{2}^{1}	...	x_{784}^{1}
Y^{2}	x_{1}^{2}	x_{2}^{2}	...	x_{784}^{2}
...	...	...	...	...
$Y^{70\,000}$	$x_{1}^{70\,000}$	$x_{1}^{70\,000}$	...	$x_{784}^{70\,000}$

Centralised - Train

- Standard normal random initialization of the centroids
- Assign each point to a centroid
- Compute the new centroids
- Repeat that several times

K-means



Parallelization

ReadFile

textFile →

Label, 1x1, 1x2, ..., 28x28
$Y^1, x_1^1, x_2^1, \dots, x_{784}^1$
$Y^2, x_1^2, x_2^2, \dots, x_{784}^2$
..., ..., ..., ..., ...
$Y^{70\,000}, x_1^{70\,000}, x_2^{70\,000}, \dots, x_{784}^{70\,000}$

ReadFile

flatMap

Label, 1x1, 1x2, ..., 28x28
$Y^1, x_1^1, x_2^1, \dots, x_{784}^1$
$Y^2, x_1^2, x_2^2, \dots, x_{784}^2$
..., ..., ..., ..., ...
$Y^{70\,000}, x_1^{70\,000}, x_2^{70\,000}, \dots, x_{784}^{70\,000}$

ReadFile

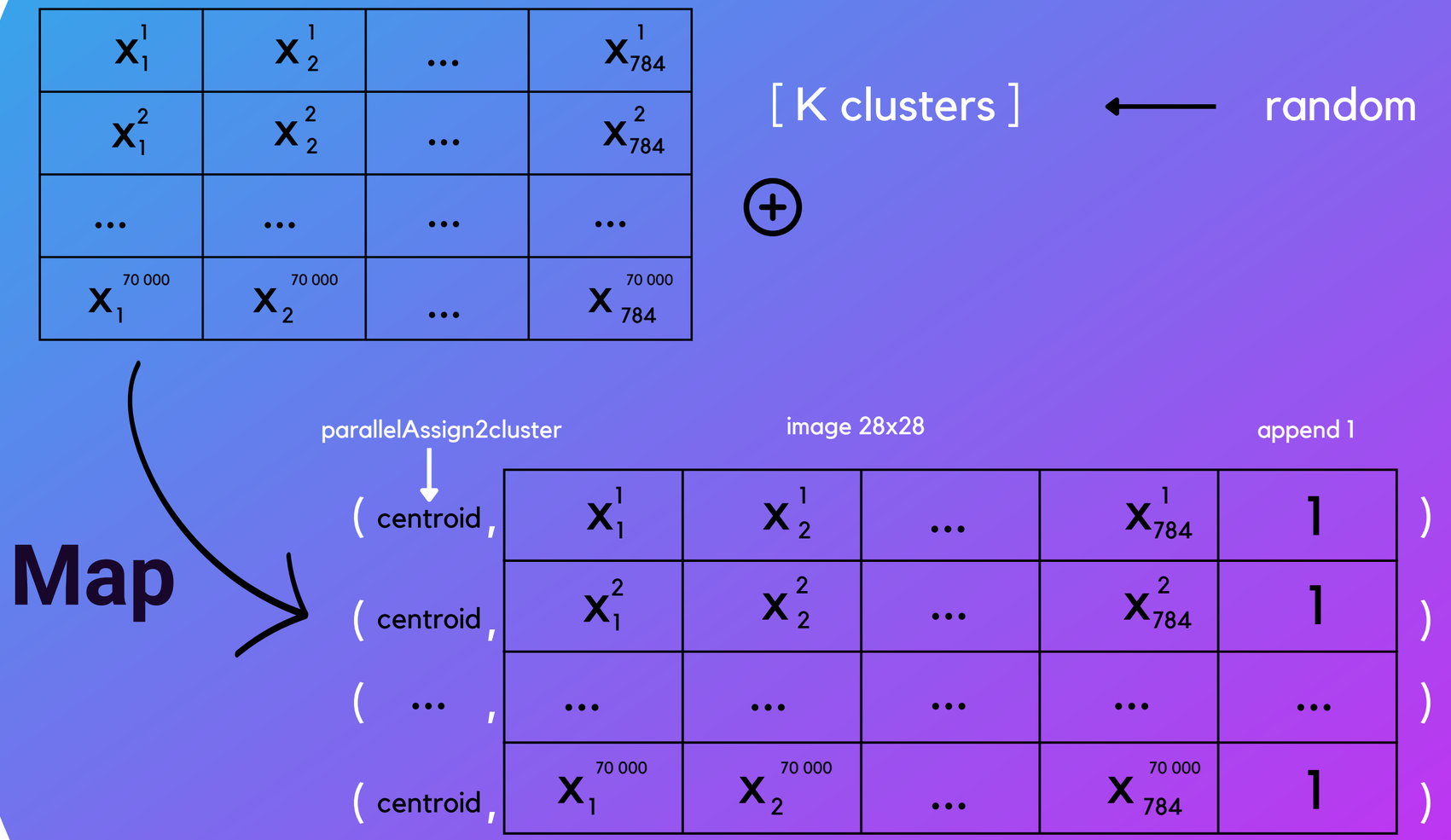
map

$[x_1^1, x_2^1, \dots, x_{784}^1]$
$[x_1^2, x_2^2, \dots, x_{784}^2]$
$[\dots, \dots, \dots, \dots]$
$[x_1^{70\,000}, x_2^{70\,000}, \dots, x_{784}^{70\,000}]$

Parallel - Train

- To parallelize the allocation of data to each cluster and to obtain a new cluster corresponding to the center of mass of all the data in the previous cluster

- Carried out iteratively

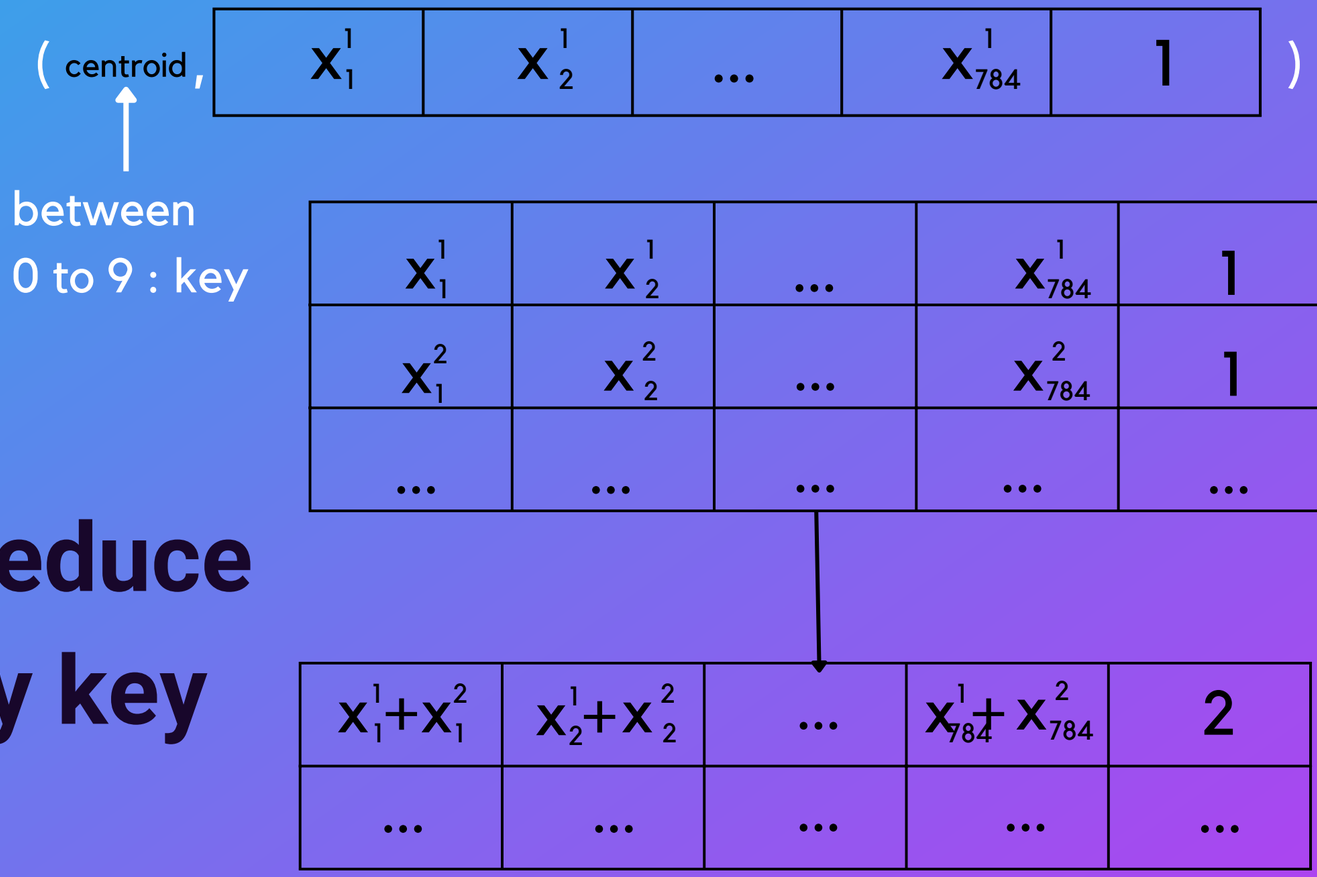


A map to transform each x into a key value with the cluster and values of x with a 1 at the end

Parallel - Train

- To parallelize the allocation of data to each cluster and to obtain a new cluster corresponding to the center of mass of all the data in the previous cluster

- Carried out iteratively



Doing that for each line and obtain 10 different key-value and a table of 785 values which are sum of all with same key

Parallel - Train

- To parallelize the allocation of data to each cluster and to obtain a new cluster corresponding to the center of mass of all the data in the previous cluster
- Carried out iteratively

Map and collect

x_1^1	x_2^1	...	x_{784}^1	count 0
x_1^2	x_2^2	...	x_{784}^2	count 1
...	...	...	...	...
x_1^{10}	x_2^{10}	...	x_{784}^{10}	count 9

x_1^1 / count 0	x_2^1 / count 0	...	x_{784}^1 / count 0
x_1^2 / count 1	x_2^2 / count 1	...	x_{784}^2 / count 1
...	...	...	...
x_1^{10} / count 9	x_2^{10} / count 9	...	x_{784}^{10} / count 9

Obtain 10 tables of 784 values all divide by the number of x entries into each cluster which are the new centroids

Centralised vs Parallelized

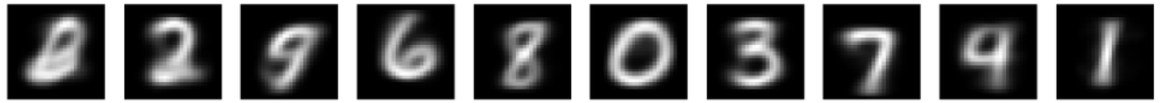
Centralised vs Parallelized

Results for centralised version :

Execution time for 10 iterations : 1607.58 s

It 1

It 3



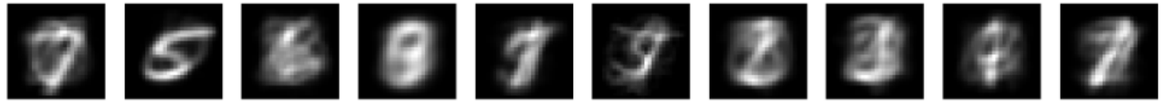
It 5

It 10

Results for parallelised version:

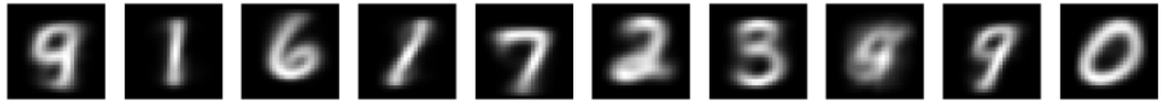
Execution time for 10 iterations : 1124.60 s

It 1

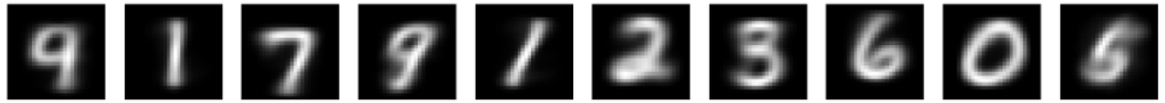


It 3

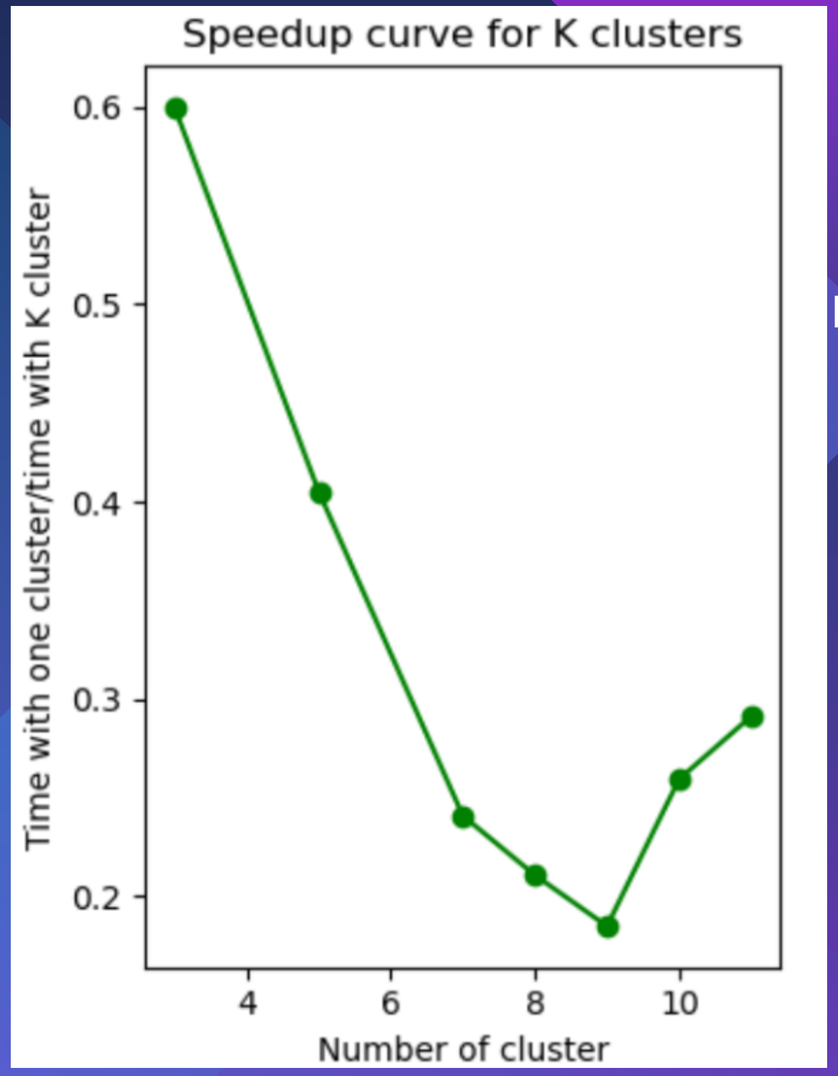
It 5



It 10

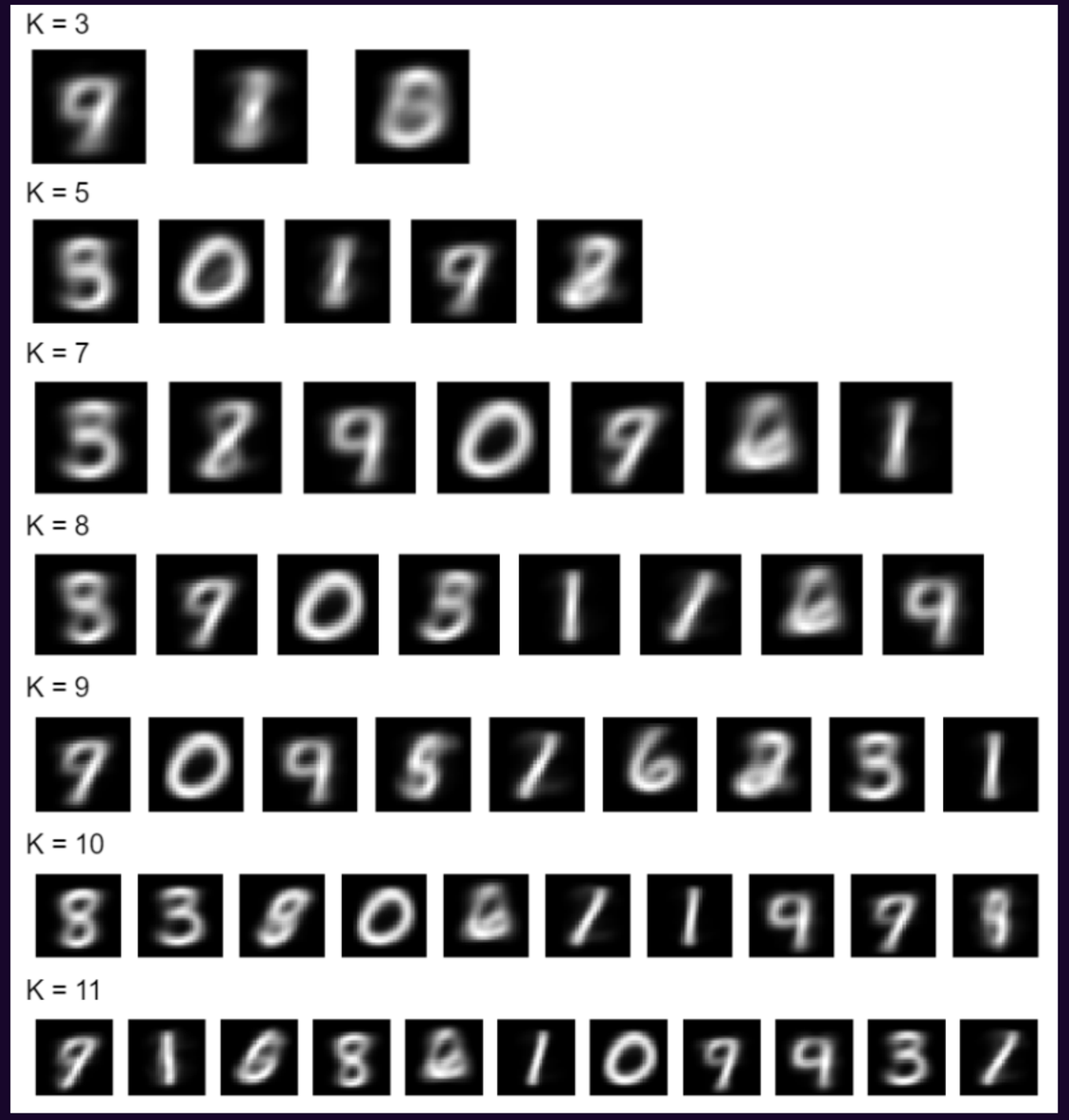


TRAINING ON K



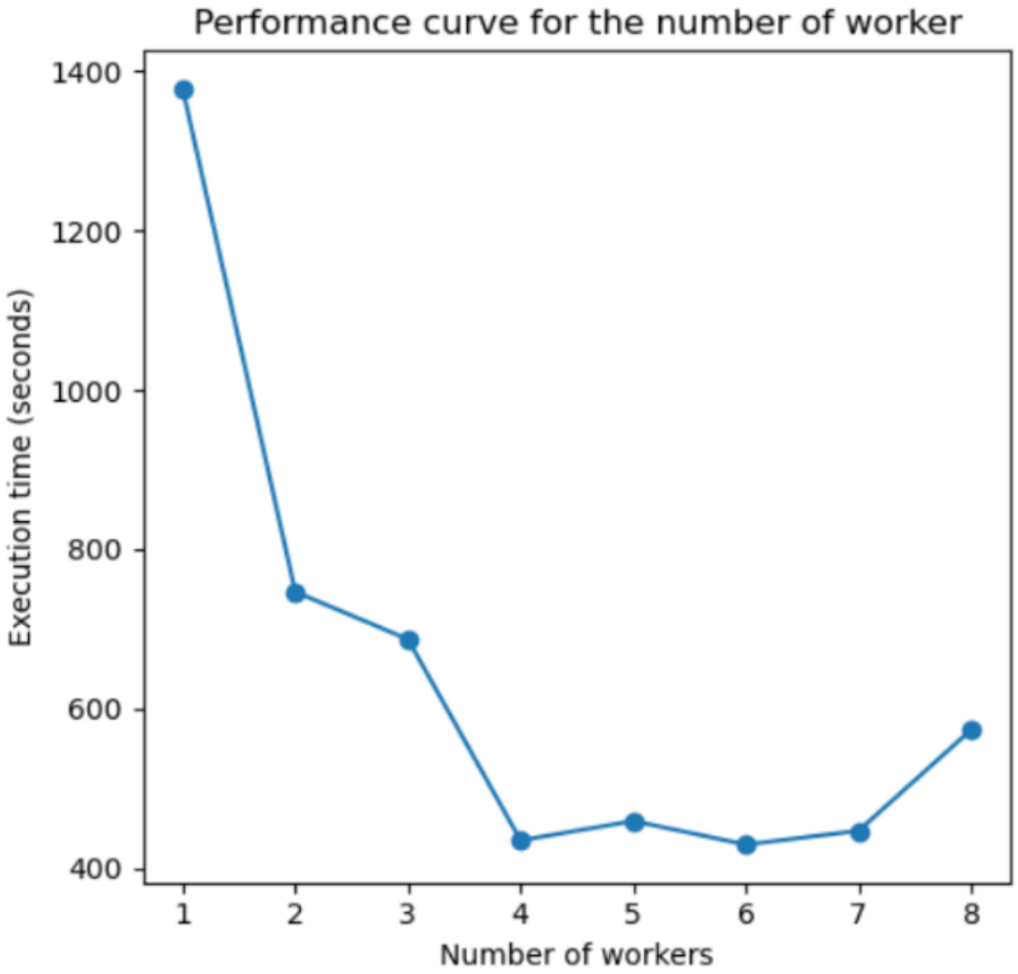
Number of iteration 10

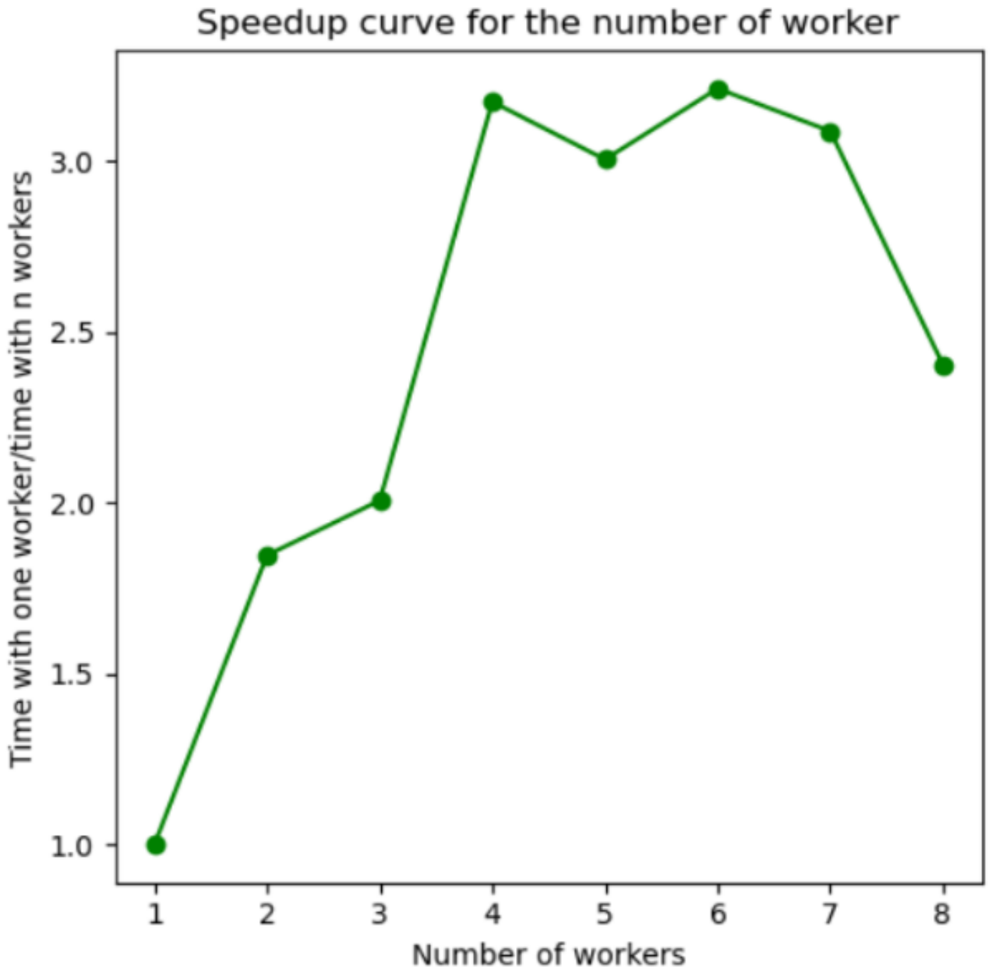
Number of cluster 3-11



Performance and speedup curve

4 cores





THANK YOU