

# Scalable Neural Networks

Tahsin Khan

COM6012 Scalable Machine Learning



# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References

# Introduction

- This lecture will cover some topics at the interface of NNs and Spark
  - Will build up on your knowledge of NNs
- The intersection of deep learning and Spark is a very active field of research
- Several open source tools started developing in the last few years
  - Technical content is getting updated rapidly

# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References

# Feedforward neural networks

- In this section, we will briefly review **feedforward neural networks** or **multilayer perceptrons**
- The MLP assumes that the input is a fixed-dimensional vector, say  $\mathbf{x} \in \mathbb{R}^p$
- This data is referred to as “**unstructured data**”: there are no assumptions about the form of the input
- Other types of neural networks assume that the input has variable size or shape
  - convolutional NN for variable-sized images
  - recurrent NN for variable-sized sequences
  - graph NN for variable-sized graphs

# The Iris flower dataset

- ❑ The Iris flower dataset is a classic dataset used in pattern recognition
- ❑ It was introduced by the British statistician Ronald Fisher in a paper from 1936
- ❑ The dataset contains 150 instances
- ❑ Four input features: sepal length, sepal width, petal length, and petal width  $\mathbf{x} \in \mathbb{R}^4$
- ❑ A categorical output, the species, which can take three values: Iris setosa, Iris virginica and Iris versicolor

# Iris flowers



Iris versicolor

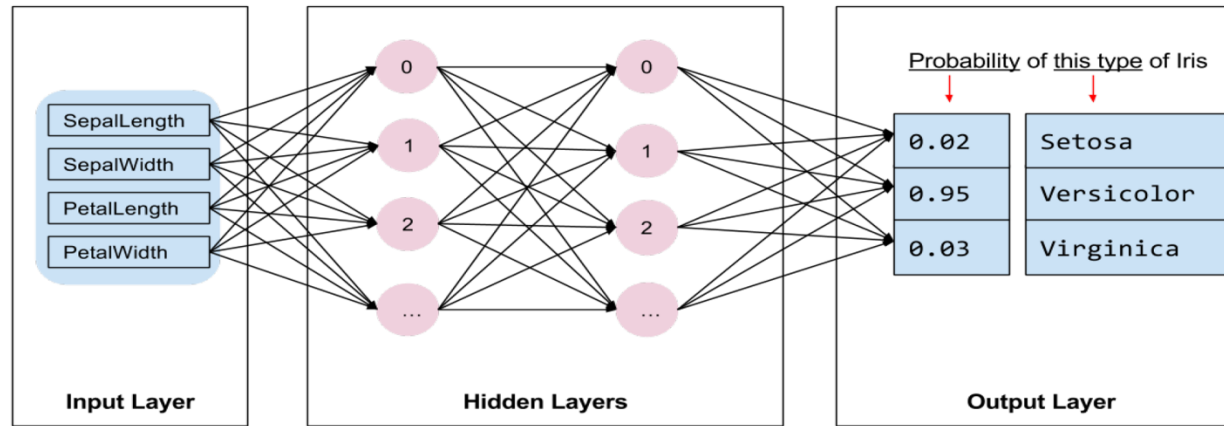


Iris setosa



Iris virginica

# A 2-layer MLP applied to the Iris dataset



(a) A 2-layer MLP

$$\begin{aligned} p(y|\mathbf{x}; \boldsymbol{\theta}) &= \text{Cat}(y|f_3(\mathbf{x}; \boldsymbol{\theta})) \\ f_3(\mathbf{x}; \boldsymbol{\theta}) &= \mathcal{S}(\mathbf{W}_3 f_2(\mathbf{x}; \boldsymbol{\theta}) + \mathbf{b}_3) \\ f_2(\mathbf{x}; \boldsymbol{\theta}) &= \varphi_2(\mathbf{W}_2 f_1(\mathbf{x}; \boldsymbol{\theta}) + \mathbf{b}_2) \\ f_1(\mathbf{x}; \boldsymbol{\theta}) &= \varphi_1(\mathbf{W}_1 f_0(\mathbf{x}; \boldsymbol{\theta}) + \mathbf{b}_1) \\ f_0(\mathbf{x}; \boldsymbol{\theta}) &= \mathbf{x} \end{aligned}$$

(b) Mathematical model

$\boldsymbol{\theta} = (\mathbf{W}_3, \mathbf{b}_3, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_1, \mathbf{b}_1)$ ,  $\mathcal{S}(\cdot)$  soft-max function,  $\text{Cat}(\cdot)$  categorical distribution,  $\varphi_1, \varphi_2$  activation functions



# $L$ -layer MLP

- An  $L$ -layer MLP is in general defined as a non-linear function

$$f(\mathbf{x}, \boldsymbol{\theta}) = f_L \left( f_{L-1} \left( \cdots \left( f_1(\mathbf{x}) \right) \cdots \right) \right)$$

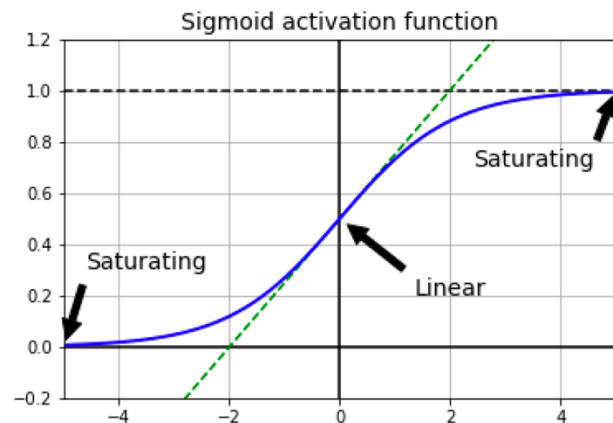
where  $f_l(\mathbf{z}_{l-1}) = \varphi_l(\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l) = \mathbf{z}_l$ ,  $\mathbf{z}_{l-1}$  are the hidden units at layer  $l - 1$  and  $\varphi_l(\cdot)$  are the activation functions at layer  $l$

- In the expression above,  $\boldsymbol{\theta} = [\mathbf{w}_L, \mathbf{b}_L, \cdots, \mathbf{w}_1, \mathbf{b}_1]$
- The activation functions  $\varphi_l(\cdot)$  are non-linear, differentiable functions such that  $\varphi_l: \mathbb{R} \rightarrow \mathbb{R}$

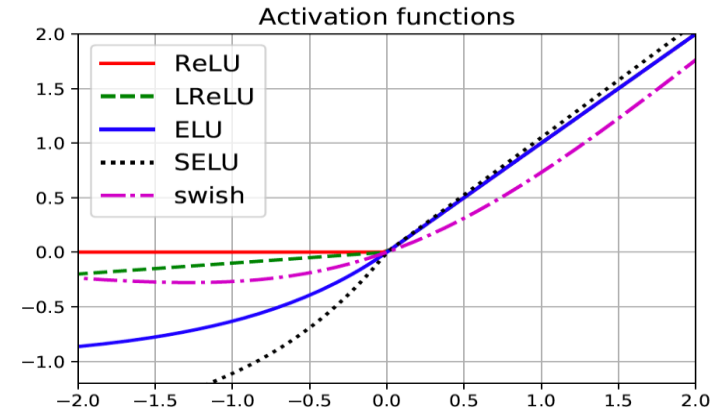
# Activation functions $\varphi_l$

Name	Definition	Range
Sigmoid	$\sigma(a) = \frac{1}{1+e^{-a}}$	$[0, 1]$
Hyperbolic tangent	$\tanh(a) = 2\sigma(2a) - 1$	$[-1, 1]$
Softplus	$\sigma_+(a) = \log(1 + e^a)$	$[0, \infty]$
Rectified linear unit	$\text{ReLU}(a) = \max(a, 0)$	$[0, \infty]$
Leaky ReLU	$\max(a, 0) + \alpha \min(a, 0)$	$[-\infty, \infty]$
Exponential linear unit	$\max(a, 0) + \min(\alpha(e^a - 1), 0)$	$[-\infty, \infty]$
Swish	$a\sigma(a)$	$[-\infty, \infty]$

(a) Mathematical form for different activation functions



(b) Sigmoid activation function



(c) Comparison of different activation functions

# Backpropagation

- ❑ Remember that the backpropagation algorithm is used to compute the gradient of the loss function wrt the parameters of the NN (MLAI module)
- ❑ Once the gradient is computed, it is passed to a gradient-based optimization algorithm

# Objective functions

- Given a dataset  $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$ , the NN parameters are usually fit using maximum likelihood estimation

$$\text{NLL}(\mathcal{D}; \boldsymbol{\theta}) = -\log p(\mathcal{D}|\boldsymbol{\theta}) = -\sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n; \boldsymbol{\theta})$$

- Practical considerations when training deep models:
  - tuning the learning rate
  - vanishing gradient problem. Solutions: modify the architecture (residual connections), standardize the activations at each layer (batch normalization) and careful parameter initialization
  - exploding gradients. Addressed with gradient clipping

# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References

# MultilayerPerceptronClassifier (I)

- ❑ Spark ML implements feed-forward neural networks through the `MultilayerPerceptronClassifier` class

- ❑ Nodes in the intermediate layers use the logistic sigmoid function

$$\varphi_l(z_i) = \frac{1}{1 + \exp(-z_i)}$$

- ❑ Nodes in the output layer use the softmax function

$$\mathcal{S}(z_k) = \frac{\exp(z_k)}{\sum_{c=1}^K \exp(z_c)}$$

where  $K$  is the number of classes corresponding to the same number of nodes in the output layer

## MultilayerPerceptronClassifier (II)

- ❑ Spark ML uses backpropagation for learning the model
- ❑ L-BFGS is the default optimization routine
- ❑ The parameter `layers` specifies the number of layers and the number of nodes per layer, from input layer to output layer
- ❑ E.g. if `layers = [780, 100, 10]`, this means a NN with 780 inputs, one hidden layer with 100 nodes and an output layer with 10 classes

# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References



# Beyond Spark ML

- ❑ Spark ML is not always the best solution for our machine learning needs
  - Predictions that require super low-latency (e.g. real-time, less than 10ms)
  - It does not have support for the algorithm we would like to use, e.g. deep learning
- ❑ In this cases, we can still use Spark, but not Spark ML
- ❑ There are different alternatives to use external models inside Spark ML
- ❑ For example, we can use Pandas UDFs (user-defined functions) for distributed inference of single-node models

# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References

# Deep learning and Spark

- Three major ways to use deep learning in Spark
  - Inference
  - Featurization and transfer learning
  - Model training

# Inference

- ❑ The simplest way to use deep learning is to use a pretrained model to make predictions in large datasets in parallel using Spark
- ❑ For example, the InceptionV3 model, a large neural network model for object recognition can be applied to our own collection of images
- ❑ In principle, one can simply use PySpark with a call to a framework like TensorFlow or Pytorch in a `map` function to get distributed inference
- ❑ In practice, there are further optimizations that can be made to make such a call efficient

# Featurization and transfer learning

- ❑ We can also use an existing model as a *feature extractor* instead of taking its final output
- ❑ As you probably know by now, deep learning models tend to learn useful feature representations in their lower layers
- ❑ Low-level features learned in Imagenet can be used a starting point to learn new models in a different dataset
- ❑ This method is called *transfer learning* and it is particularly useful when we do not have a large amount of training data for the new problem

# Model training

- ❑ Spark can also be used to train a new deep learning model from scratch
- ❑ One can either train a *single* model over multiple executors communicating between them
- ❑ Or one can train *multiple* instances of similar models in parallel to try various model architectures and hyper-parameters

# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References

# What are UDFs?

- ❑ Spark SQL has a lot of built-in functions that can be applied to dataframes
- ❑ Spark is flexible enough to allow users to define their own functions
- ❑ These are known as *user-defined-functions* (UDFs)
- ❑ A typical example in data analytics:
  - wrap a ML model within an UDF
  - query its predictions in Spark
  - there is no need to understand the internal behavior of the model SQL



# PySpark UDFs before Spark 2.3

- ❑ Up until Spark 2.3, a permanent issue with PySpark UDFs was that they had lower performance compared with Scala UDFs
- ❑ The reason for this was that PySpark UDFs required data movement between JVM and Python, which was expensive
- ❑ For details, see [Speeding up PySpark with Apache Arrow](#) by Bryan Cutler

# PySpark UDFs from Spark 2.3

- ❑ Pandas UDFs, introduced in Spark 2.3, addressed this issue by using Apache Arrow to transfer data and Pandas to work with the data
- ❑ Once the data is in Apache Arrow, there is no longer the need to serialize the data as it is already in a format consumable by the Python process
- ❑ Instead of operating on individual inputs row by row, now the operation is over Pandas Series or DataFrame
  - This is what is known as vectorized execution

# A simple example of speed-up

```
from pyspark.sql.functions import rand
df = spark.range(1 << 22).toDF("id").withColumn("x", rand())
df.printSchema()
```

```
root
|-- id: long (nullable = false)
|-- x: double (nullable = false)
```

```
%time pdf = df.toPandas()
```

```
CPU times: user 10.6 s, sys: 714 ms, total: 11.3 s
Wall time: 16.1 s
```

```
spark.conf.set("spark.sql.execution.arrow.enabled", "true")
```

```
%time pdf = df.toPandas()
```

```
CPU times: user 102 ms, sys: 134 ms, total: 236 ms
Wall time: 1.04 s
```

```
pdf.describe()
```

	id	x
count	4.194304e+06	4.194304e+06

# PySpark UDFs from Spark 3.0

- ❑ From Apache Spark 3.0 with Python 3.6+, the Pandas UDFs were split into two API categories: **Pandas UDFs** and **Pandas Function APIs**

## ❑ Pandas UDFs

- From Spark 3.0, Pandas UDFs infer the Pandas UDFs type from *Python type hints* such as `pandas.Series` or `pandas.Dataframes`
- Previously, users were required to manually define and specify each Pandas UDF type
- Currently, the supported cases of Python type hints in Pandas UDFs are
  - Series to Series
  - Iterator of Series to Iterator of Series
  - Iterator of multiple series to Iterator of Series
  - Series to Scalar.
  - All of the above replacing Series with DataFrame

# PySpark UDFs from Spark 3.0

- ❑ From Apache Spark 3.0 with Python 3.6+, the Pandas UDFs were split into two API categories: **Pandas UDFs** and **Pandas Function APIs**
- ❑ **Pandas Function APIs**
  - They allow to directly apply a local Python function to a PySpark Dataframe where both the input and output are Pandas instances
  - Currently, the supported Pandas Function APIs are
    - Grouped Map with `groupBy().applyInPandas()`
    - Map with `mapInPandas()`
    - Co-grouped Map with `groupBy().cogroup().applyInPandas()`
- ❑ This week's lab will contain example of using `mapInPandas()` for making inference with a NN
- ❑ [This blog post by Databricks](#) provides more details on Pandas UDFs and Pandas Function APIs in Apache Spark 3.0

# Iterator support in Pandas UDFs

- ❑ A typical use case for Pandas UDFs is to load a machine learning or deep learning model to perform distributed inference
- ❑ If the model is large (typical deep learning models can be tens or hundreds of Mbs), then there is a high overhead to load the same model again and again for every batch in the same Python process
- ❑ Pandas UDFs accept iterators of `pandas.Series` or `pandas.DataFrame`
- ❑ The pseudocode looks something like

```
@pandas_udf(...)
def predict(iterator):
    model = ... # load model
    for features in iterator:
        yield model.predict(features)
```

# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References

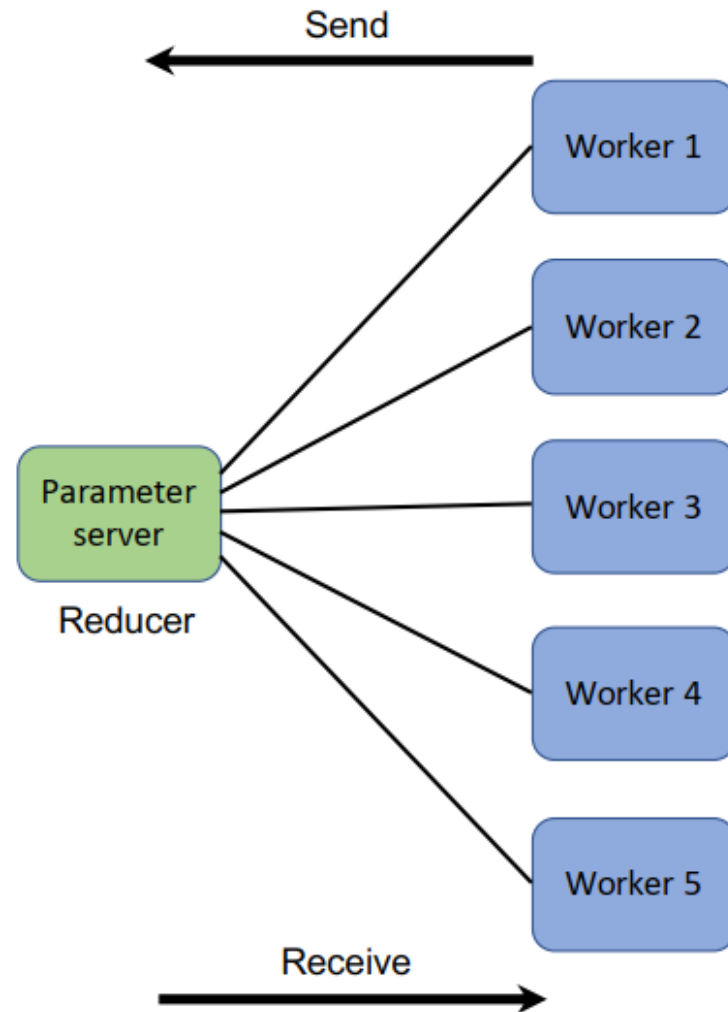
# Horovod

- ❑ Horovod is an open-source software framework for distributed deep learning training using TensorFlow, Keras, PyTorch and MXNet
- ❑ It was developed by Uber
- ❑ Initial release was from August 2017 and the latest stable release is from May, 2020
- ❑ Details on the following slides are from the blog post [Bringing HPC Techniques to Deep Learning](#) by Andrew Gibiansky





# The parameter server approach



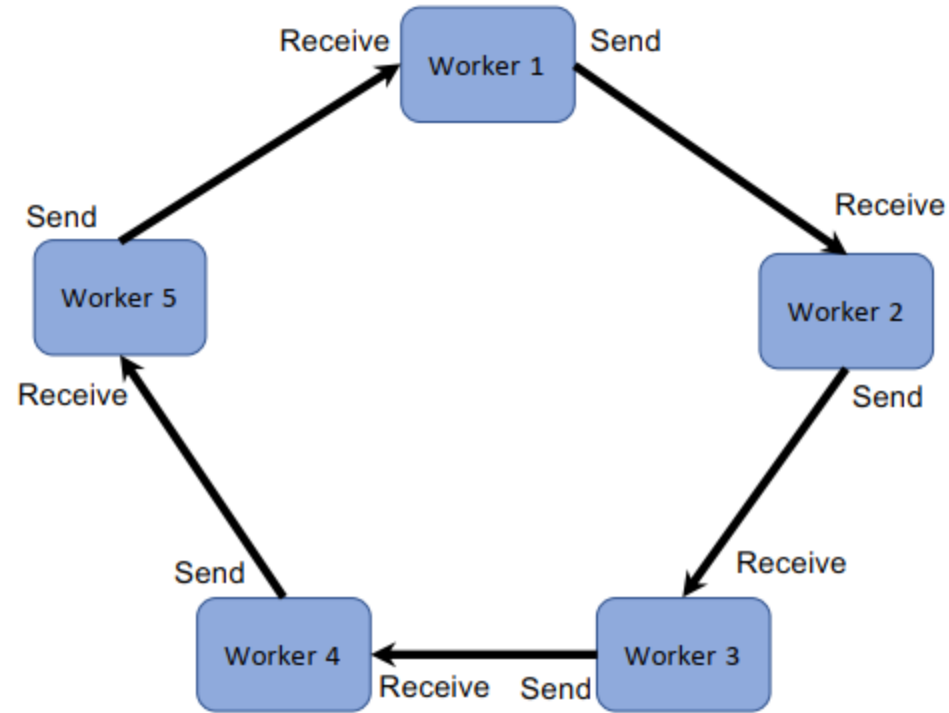
# The parameter server approach

- ❑ Deep learning models have potentially millions of parameters
- ❑ For example, AlexNet has 60 millions of parameters. There are models today that have even billions of parameters – LLMs
- ❑ In the parameter server approach, each worker requires a complete copy of the neural network
- ❑ Sending the parameters of these NNs back and forward between the server and the workers creates a communication overhead that slows training
- ❑ The more workers in the system, the greater the communication cost

# Ring reduce

- ❑ In contrast, ring reduce is an algorithm for which communication is constant
- ❑ If we only consider bandwidth as a factor in the communication cost, the ring allreduce is an optimal communication algorithm
- ❑ The workers in ring reduce are arranged in a logical ring

# Ring allreduce

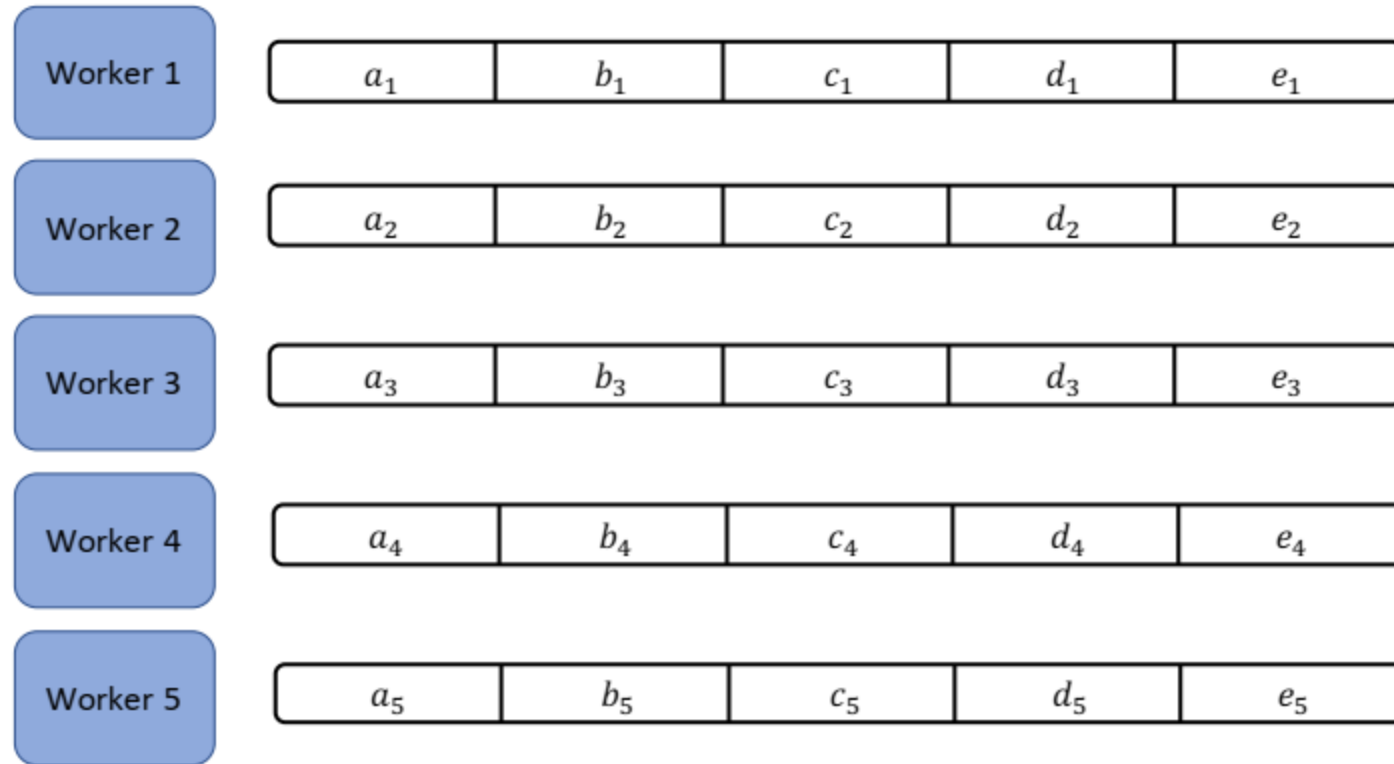


- ❑ Each worker has a right neighbour and a left neighbour
- ❑ Each worker sends data to its right neighbour and receives data from its left neighbour

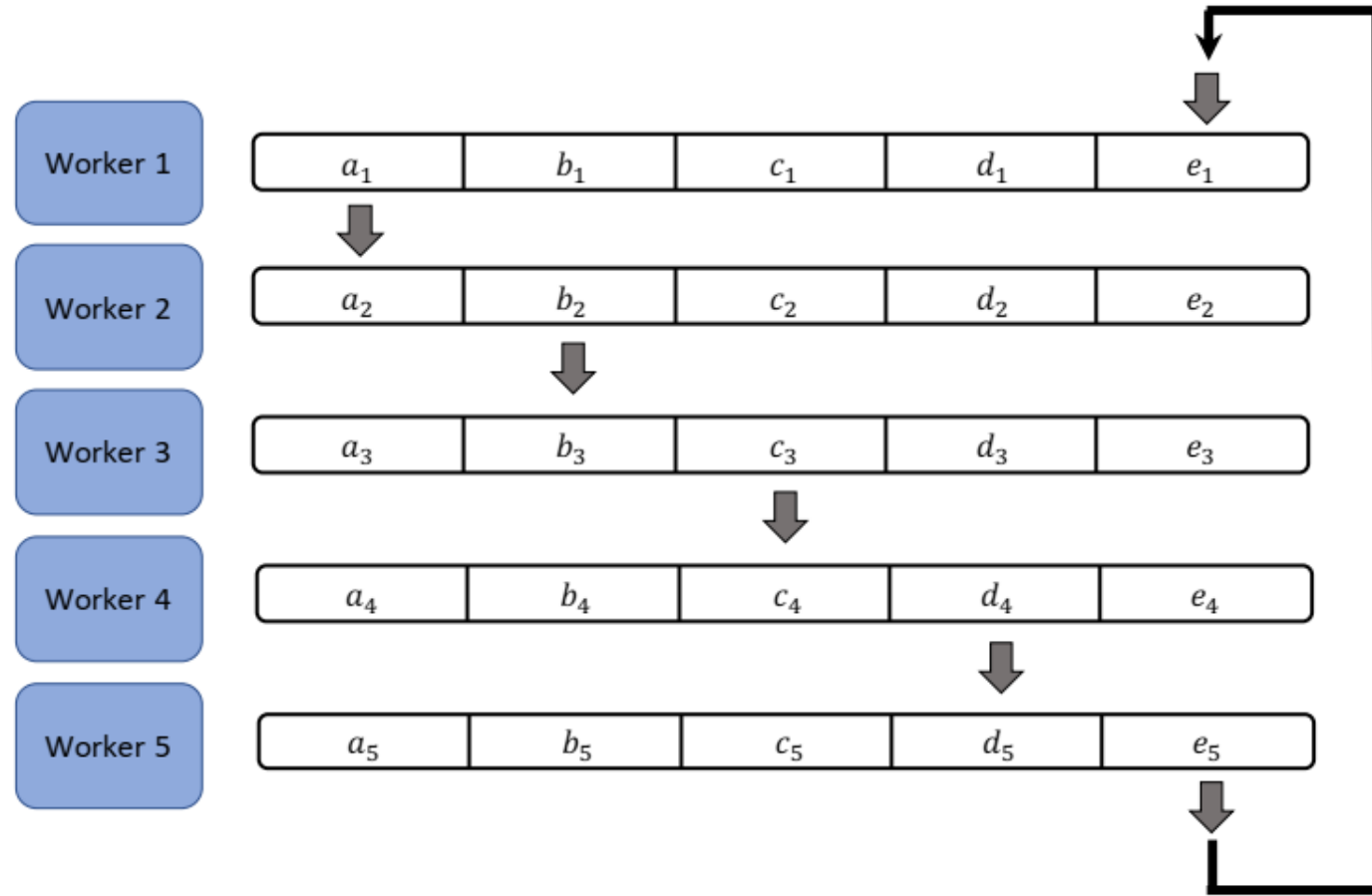
# Ring allreduce: two-stages

- ❑ Ring all reduce is performed in two stages
- ❑ In the scatter-reduce stage, workers exchange data such that every worker ends up with a chunk of the final result
- ❑ In the allgather stage, the workers exchange those chunks such that each worker ends up with the complete final result

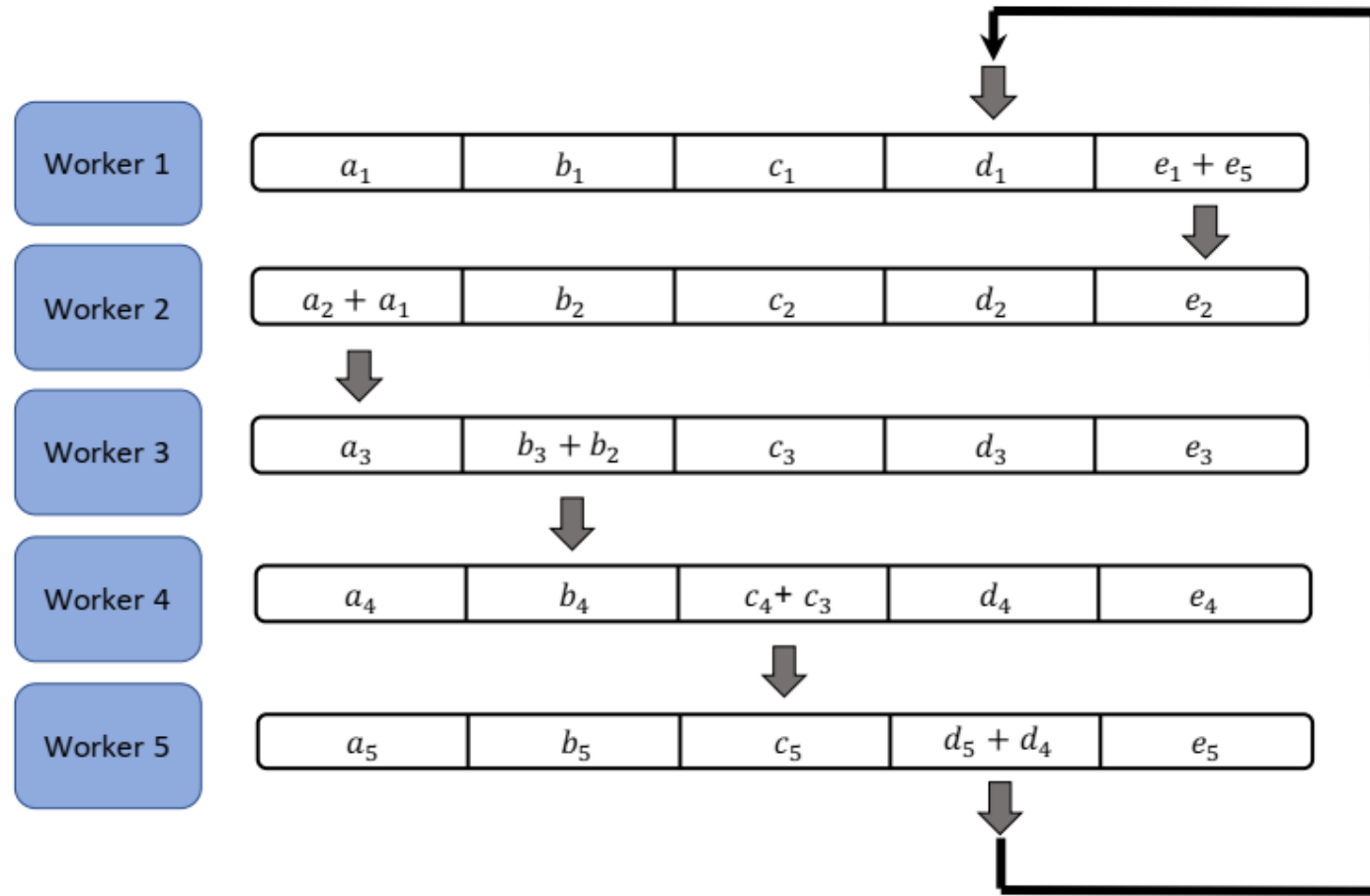
# Summing arrays: scatter-reduce



# Summing arrays: scatter-reduce

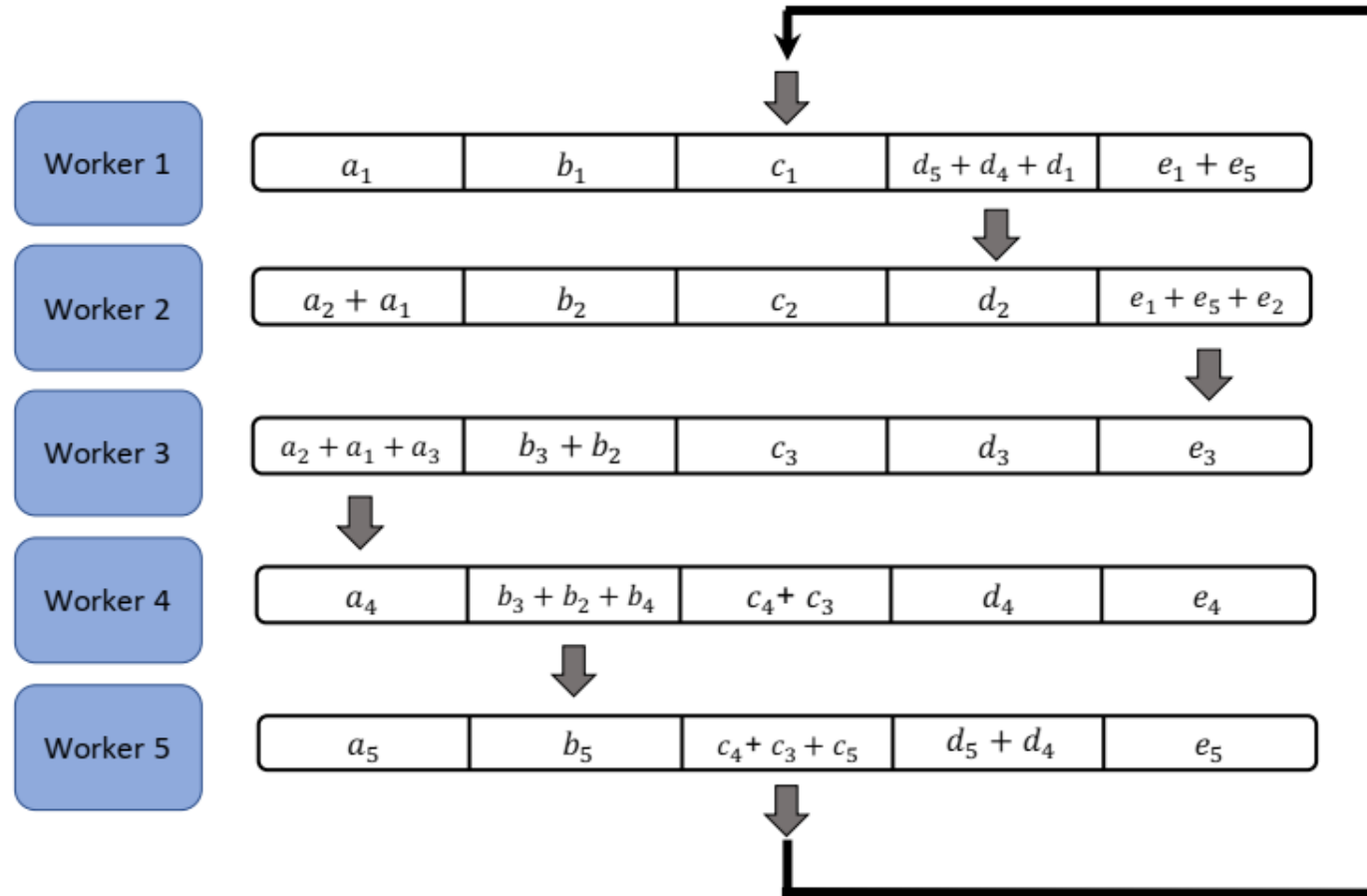


# Summing arrays: scatter-reduce

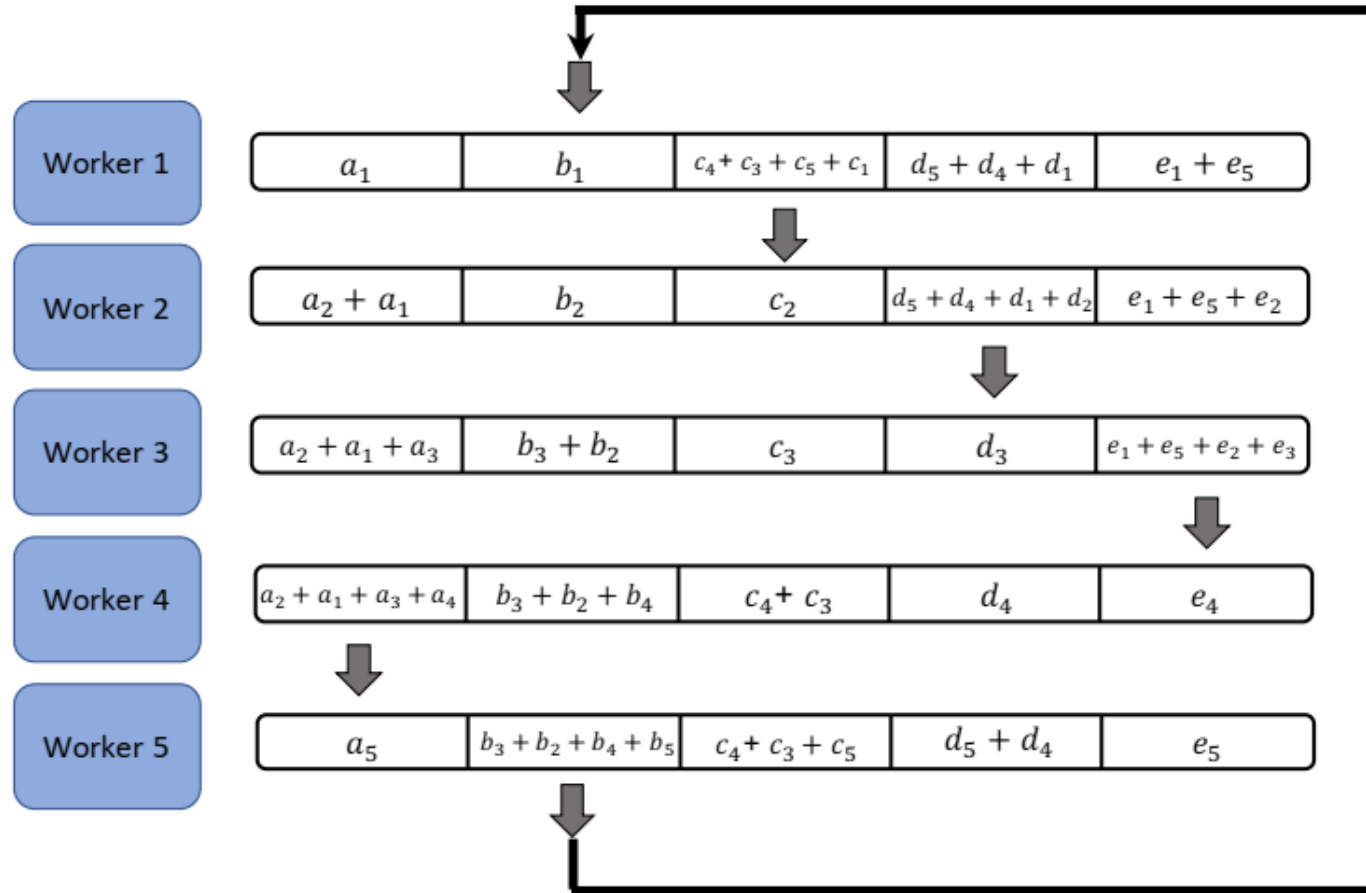




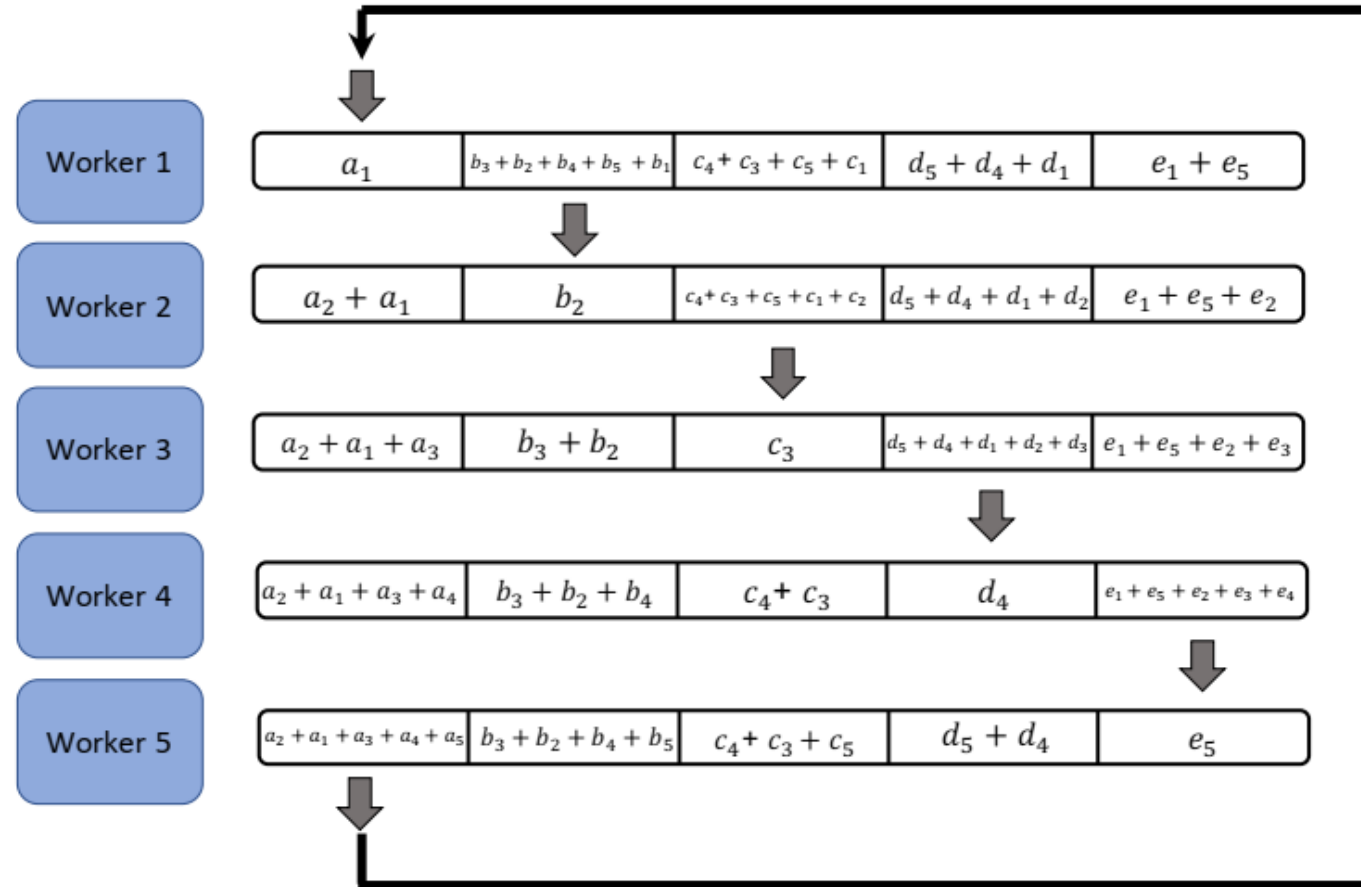
# Summing arrays: scatter-reduce



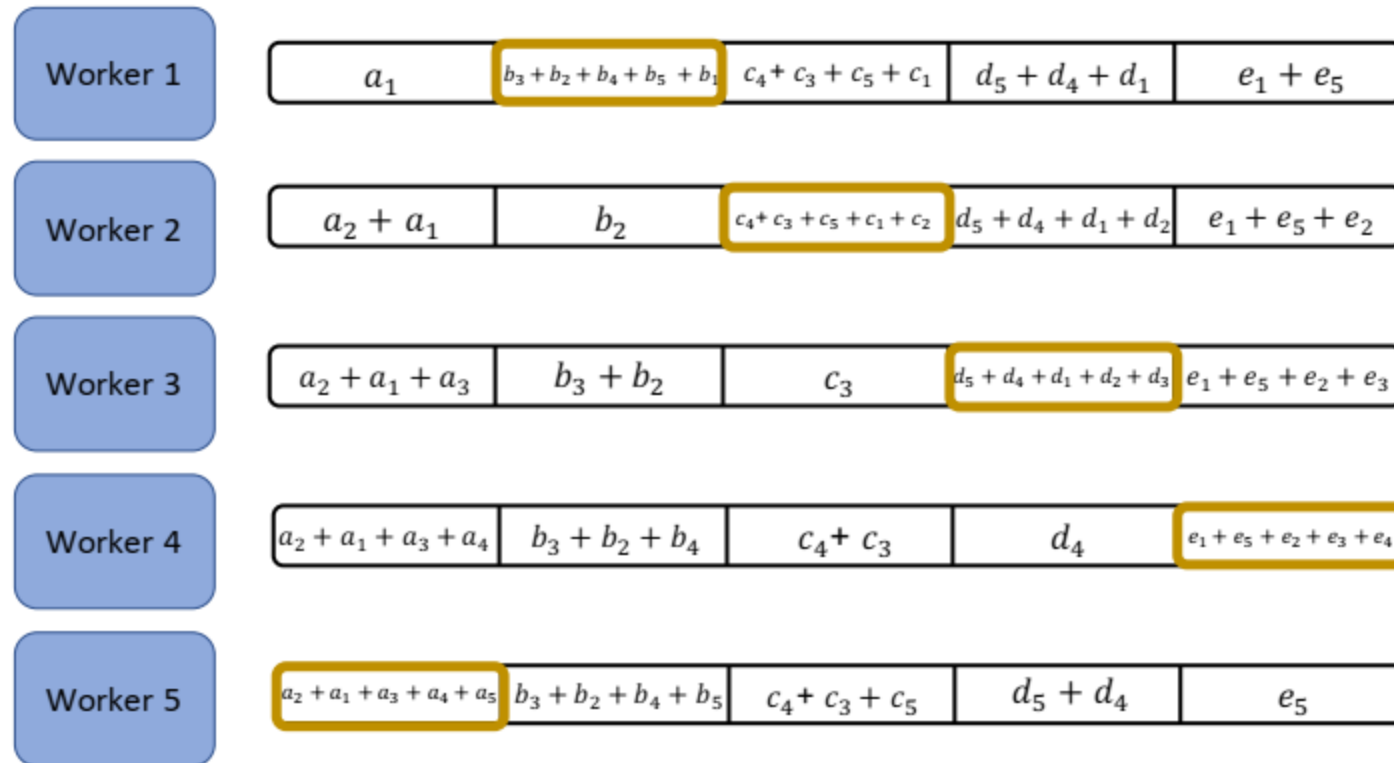
# Summing arrays: scatter-reduce



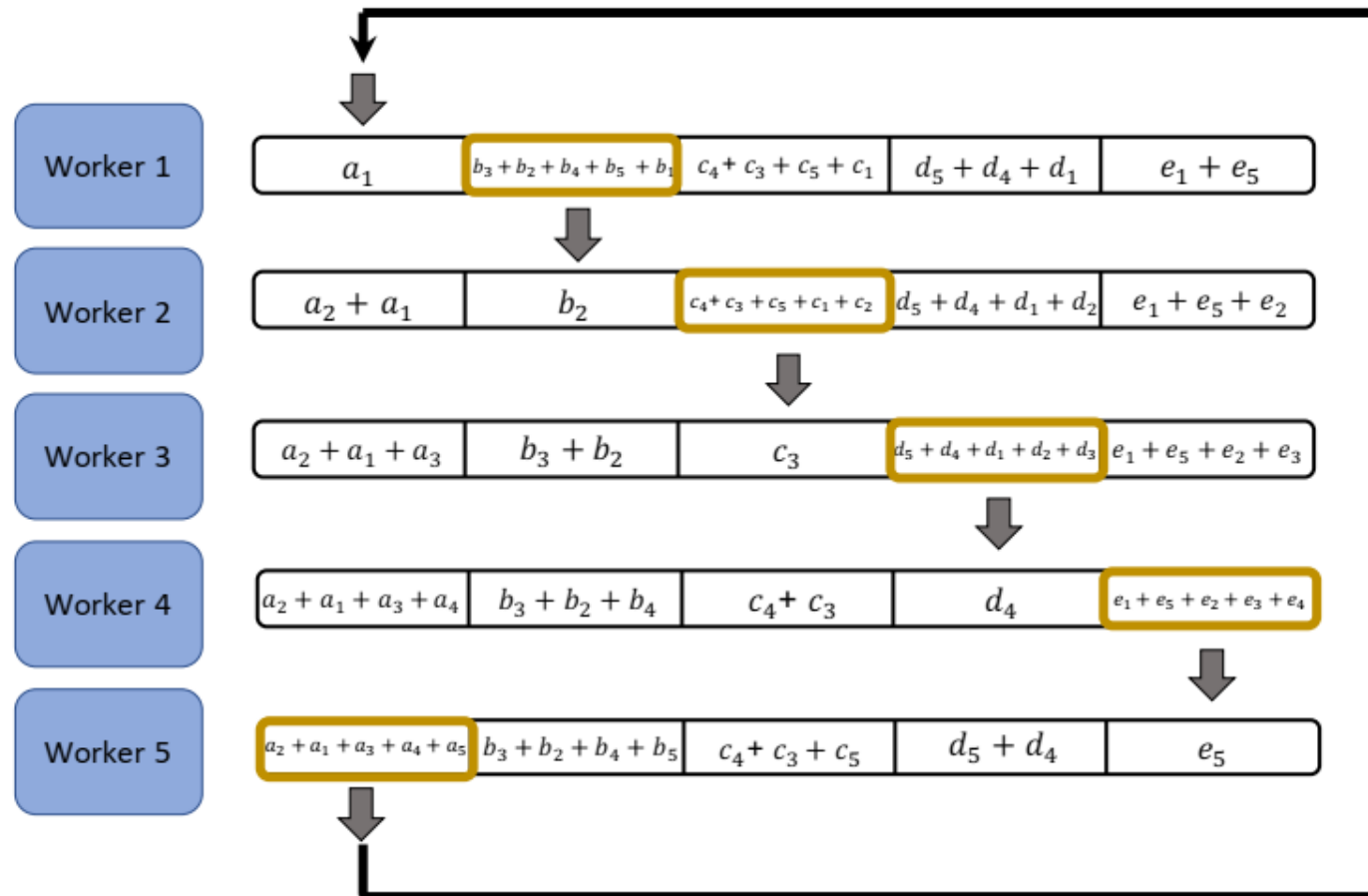
# Summing arrays: scatter-reduce



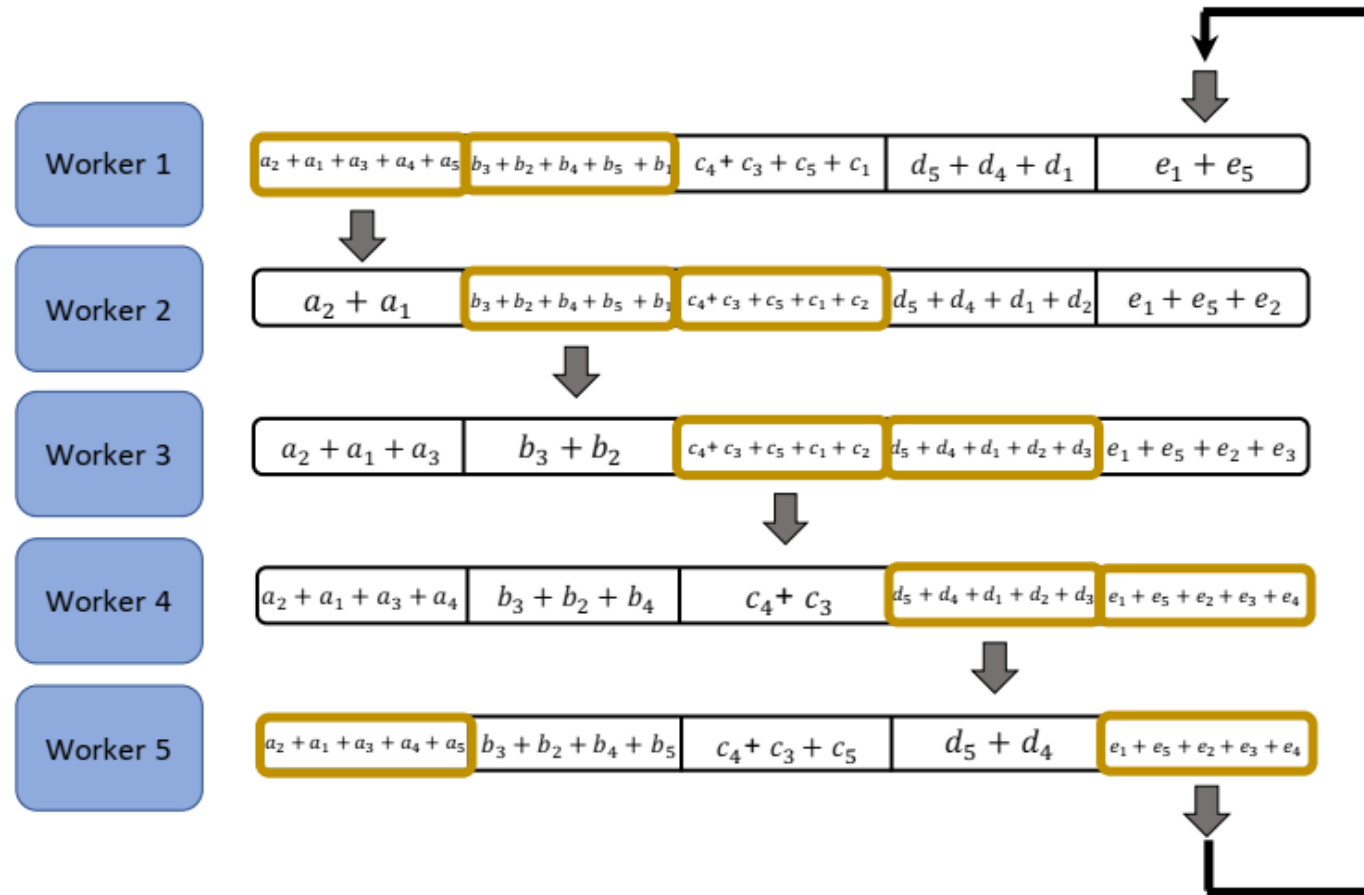
# Summing arrays: scatter-reduce



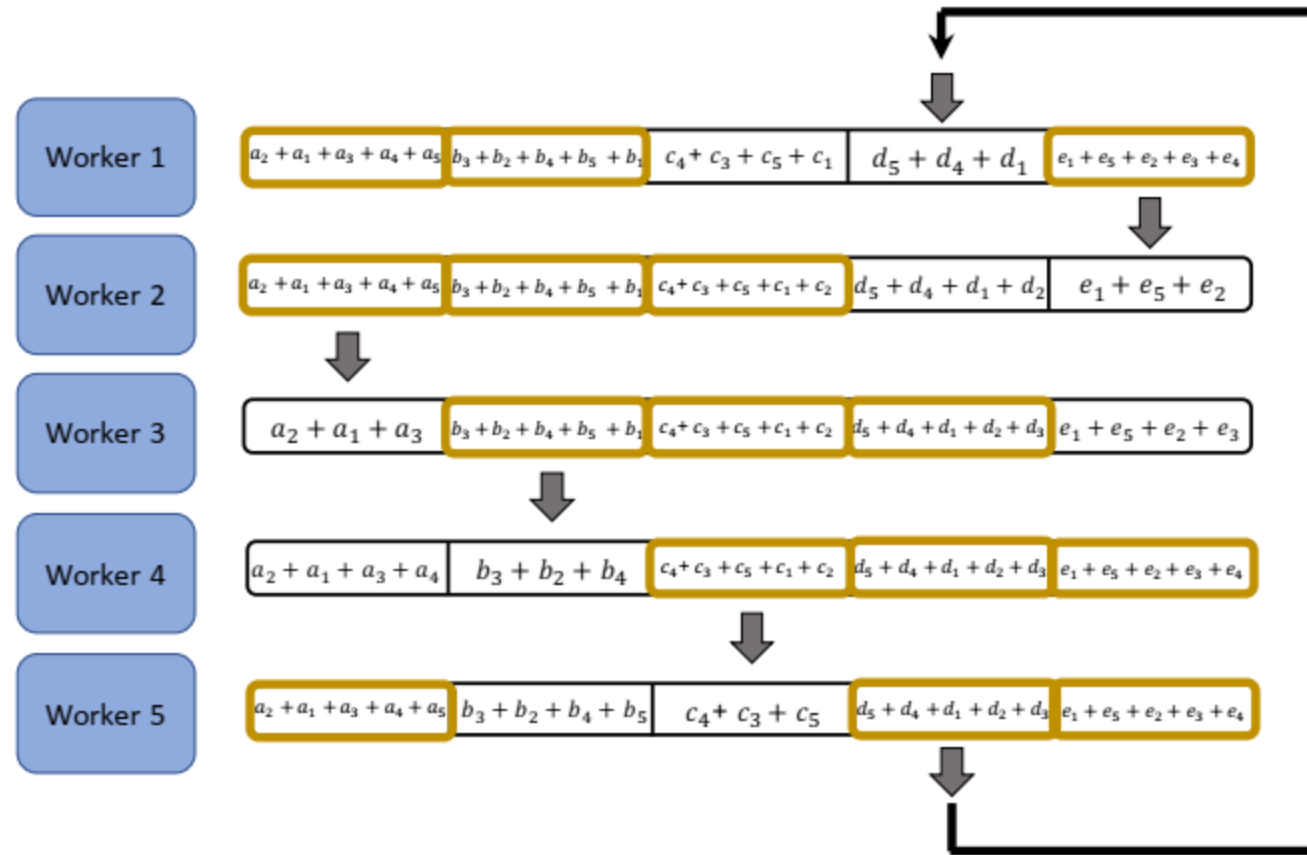
# Summing arrays: allgather



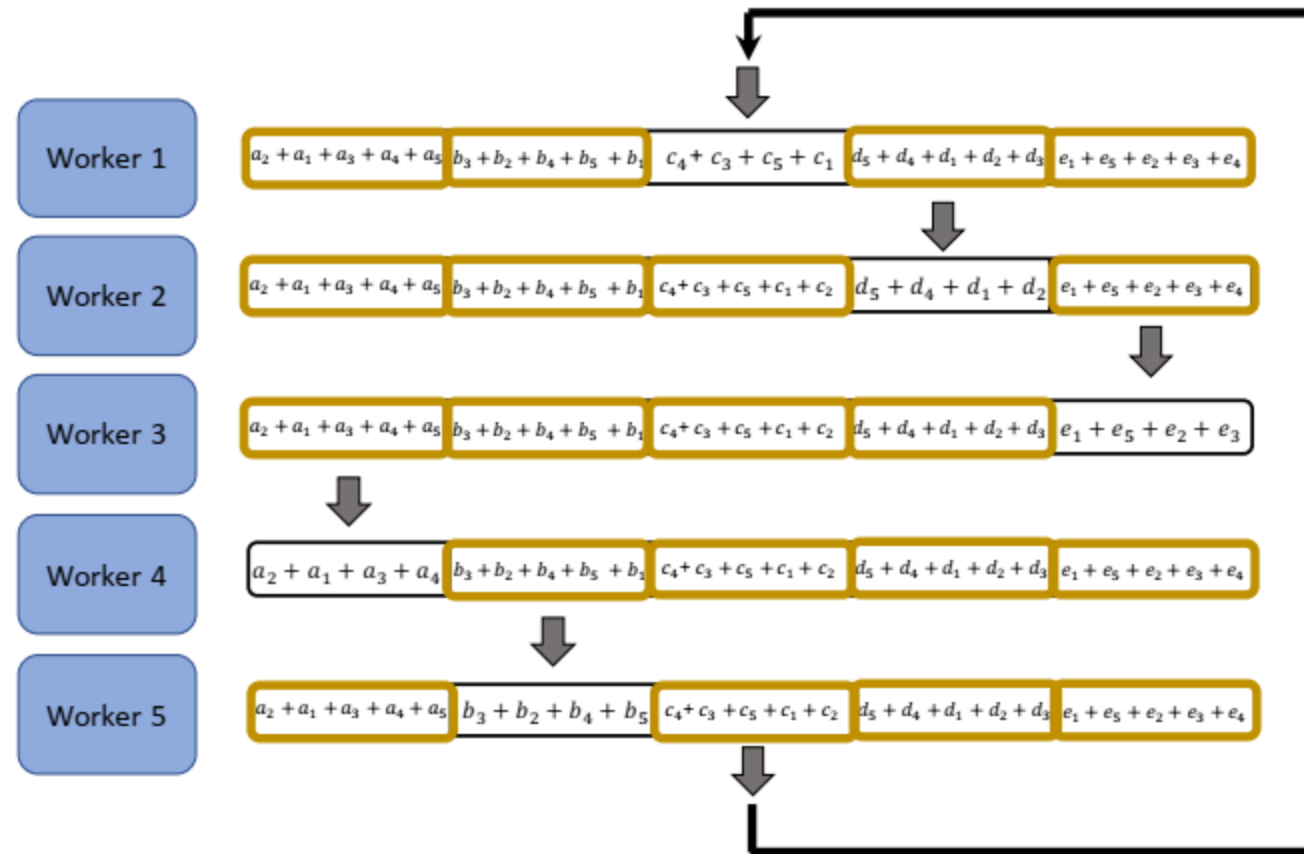
# Summing arrays: allgather



# Summing arrays: allgather

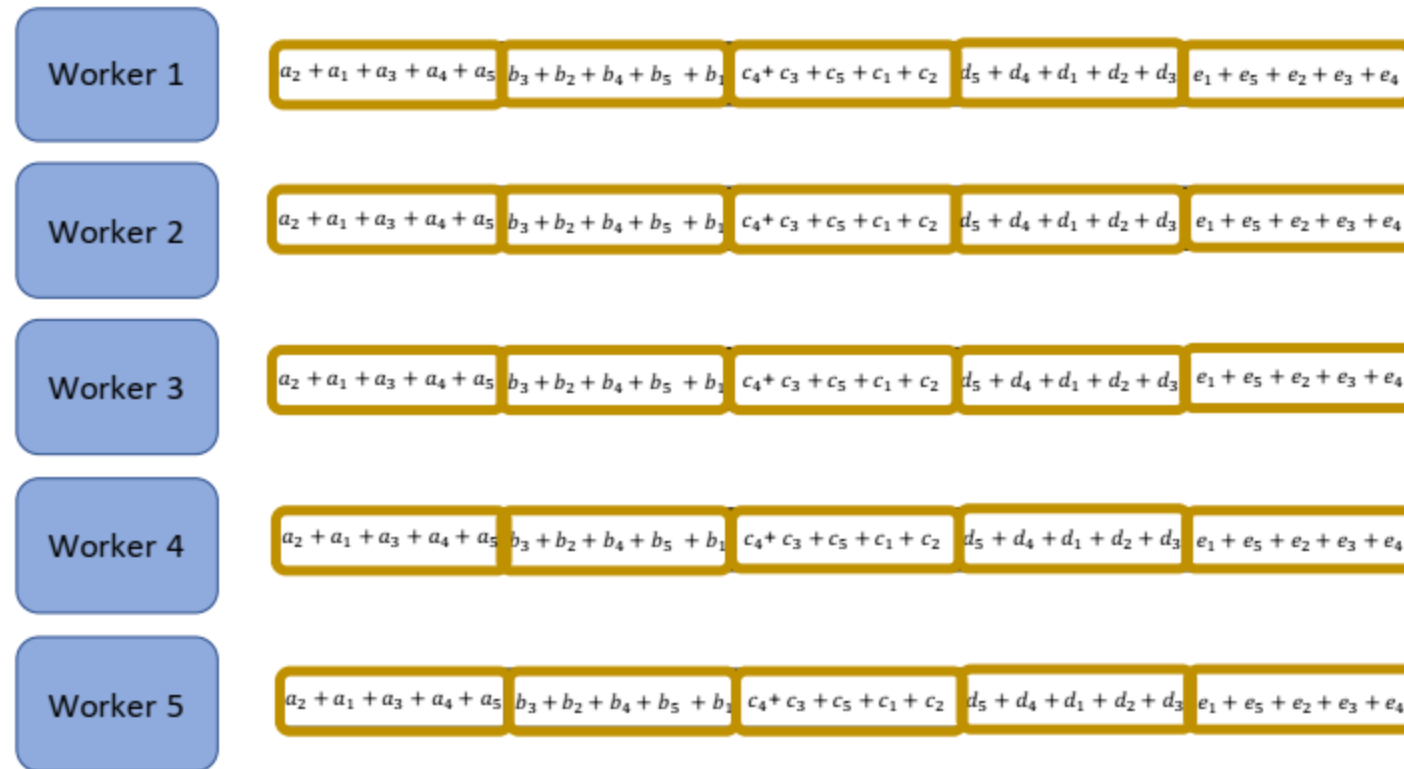


# Summing arrays: allgather





# Summing arrays: allgather



# Communication cost

- ❑ If there are  $N$  workers, each worker will send and receive values  $N-1$  times for scatter-reduce and  $N-1$  times for allgather
- ❑ Each time, the workers will send  $K/N$  values, where  $K$  is the total number of values in the array that are being summed across the different workers
- ❑ Then, the total amount of data transferred to and from every worker is

$$data_{transferred} = 2(N - 1) \frac{K}{N}$$

which is independent of the number of workers

# Origins of Horovod

- ❑ The idea of using ring allreduce to efficiently train deep learning models was introduced by a team at Baidu in early 2017
- ❑ Uber later adopted Baidu's draft implementation of the TensorFlow ring-allreduce algorithm and built upon it
- ❑ The resulting Python package was named Horovod, “named after a traditional Russian folk dance in which performers dance with linked arms in a circle”



Horovod dancing. Source: Wikimedia commons

# Horovod on Spark

- ❑ Two APIs provided: a high level **Estimator API** and a lower level **Run API**
- ❑ The Estimator API is recommended if
  - training is done with Keras or PyTorch
  - you want to train directly on a Spark Dataframe from *pyspark*
  - you are using a standard gradient descent optimization process for training
- ❑ The Run API offers more fine-grained control



Distributed training framework for TensorFlow, Keras, PyTorch, and Apache MXNet.

Star 11,118

## Navigation

- Overview
- Concepts
- Horovod Installation Guide
- API
- Horovod with TensorFlow
- Horovod with Keras
- Horovod with PyTorch

## Horovod on Spark

The `horovod.spark` package provides a convenient wrapper around Horovod that makes running distributed training jobs in Spark clusters easy.

In situations where training data originates from Spark, this enables a tight model design loop in which data processing, model training, and model evaluation are all done in Spark.

We provide two APIs for running Horovod on Spark: a high level **Estimator API** and a lower level **Run API**. Both use the same underlying mechanism to launch Horovod on Spark executors, but the Estimator API abstracts the data processing (from Spark DataFrames to deep learning datasets), model training loop, model checkpointing, metrics collection, and distributed training.

We recommend using Horovod Spark Estimators if you:

- Are using Keras (`tf.keras` or `keras`) or PyTorch for training.
- Want to train directly on a Spark DataFrame from `pyspark`.
- Are using a standard gradient descent optimization process as your training loop.

If for whatever reason the Estimator API does not meet your needs, the Run API offers more fine-grained control.

# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References

# Ray



[docs](#) [passing](#) [Ray](#) [Join Slack](#) [Discuss](#) [Ask Questions](#)

**Ray provides a simple, universal API for building distributed applications.**

Ray is packaged with the following libraries for accelerating machine learning workloads:

- [Tune](#): Scalable Hyperparameter Tuning
- [RLlib](#): Scalable Reinforcement Learning
- [RaySGD](#): Distributed Training Wrappers
- [Ray Serve](#): Scalable and Programmable Serving

There are also many [community integrations](#) with Ray, including [Dask](#), [MARS](#), [Modin](#), [Horovod](#), [Hugging Face](#), [Scikit-learn](#), and others. Check out the [full list of Ray distributed libraries here](#).

Install Ray with: `pip install ray`. For nightly wheels, see the [Installation page](#).

`https://github.com/ray-project/ray`

# Analytics Zoo



Distributed TensorFlow, PyTorch, Keras and BigDL on Apache Spark & Ray

---

Analytics Zoo is an open source **Big Data AI** platform, and includes the following features for scaling end-to-end AI to distributed Big Data:

- [Orca](#): seamlessly scale out TensorFlow and PyTorch for Big Data (using Spark & Ray)
- [RayOnSpark](#): run Ray programs directly on Big Data clusters
- [BigDL Extensions](#): high-level Spark ML pipeline and Keras-like APIs for BigDL
- [Zouwu](#): scalable time series analysis using AutoML

For more information, you may [read the docs](#).

---

`https://github.com/intel-analytics/analytics-zoo`

# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References



# Acknowledgement

Lecture slides were adapted from [Dr Mauricio Alvarez](#), who contributed to this module from 2017 to 2022

# Contents

- Neural networks for unstructured data
- NNs in Spark ML
- Deep learning with Spark
  - Ways of using neural networks/deep learning with Spark
  - Pandas UDFs
  - Horovod
  - Other technologies to watch out
- Acknowledgement
- References

# References

## □ Books:

- Probabilistic Machine Learning: An introduction by Kevin P Murphy, MIT Press, 2022 (<https://probml.github.io/pml-book/book1.html>)
- Book: Spark: The Definitive Guide by Bill Chambers & Matei Zahaira, O'Reilly, 2018
- Book: Learning Spark: Lightning-Fast Data Analytics by Jules Damji, Brooke Wenig, Tathagata & Denny Lee, O'Reilly, 2020

## □ Research paper:

- Horovod: fast and easy distributed deep learning in TensorFlow by Alexander Sergeev and Mike Del Balso (<https://arxiv.org/abs/1802.05799>)

## □ Web resource:

- Web resource: Spark Python API Documentation (<https://spark.apache.org/docs/latest/api/python/reference/index.html>)