Competence Network

ML & DL on Hadoop

Marvin Follmann | 20.09.2019

About Us

Seamless Analytics GmbH

- Founded in 2018
  - Marvin Follmann, M. Sc.
  - Daniel Müller, M. Sc.

- Consulting and services for data storage and analysis
  - Project planning and implementation
  - Apache Hadoop ecosystem
  - Data Analysis
  - Machine Learning and Deep Learning

- The Idea originated through research projects and scientific works
  - Hochschule Offenburg

https://seamless-analytics.de

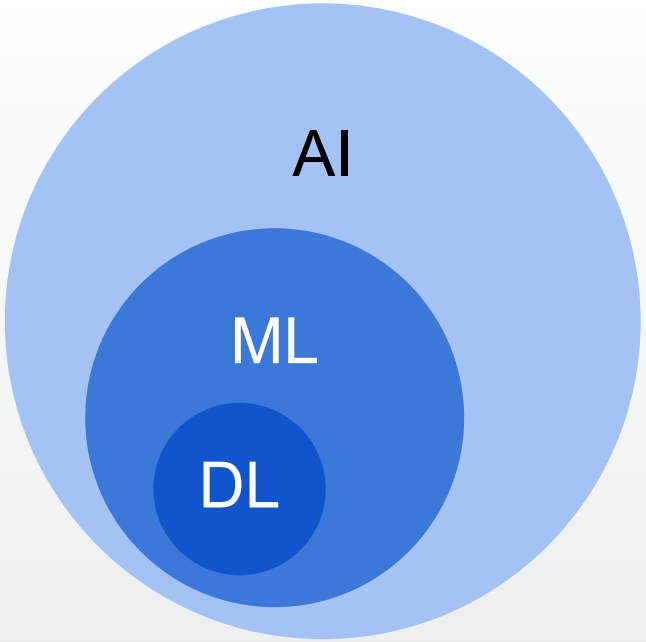
Agenda of Today's Workshop

- Why ML on Hadoop
- How to do ML on Hadoop
  - Python on YARN
  - Python with PySpark
- Basics
- Python on YARN
- Python with Spark/PySpark
- Movielens on Spark

Why ML on Hadoop

ML = Machine Learning
DL = Deep Learning

Why ML ?



- Computers can become more powerful with ML and DL
- Tasks which are repetitive and complex can be automated

- Why wasn't it done *X* years before?
  - Lack of computing power
  - Lack of data

- Today ⟶ we have more computational power, larger data sets and better algorithms
  - Computers can learn complex function approximations directly from data
  - New possibilities for automation

- Projected market sizes
  - 2020: 4.8 bn USD (Source: Statista)
  - 2023: 31.0 bn USD (Source: Statista)

Why ML?

- Because it is useful for…
  - Clustering → e.g. Customers
  - Text Analysis → e.g. Reviews, News-Articles,…
  - Spam-Detection → e.g. Email, SMS, …
  - Sentiment Analysis → e.g. Topic, Company, Product, Person… (+1,0,-1)
  - Recommender Systems → e.g. Product, Insurance, Movie, Music, Article, News…
  - Fraud-Detection → e.g. Bank-transactions, Cyber-security, …
  - Predictive Maintenance → e.g. Agricultural machines
  - Medical Diagnoses → e.g. X-Rax/CT/… Object detection, Segmentation, …
  - Customer Satisfaction → e.g. Analyse customer interaction
  - …
  - Demand forecasting, Price optimization, Market segmentation, Churn rate analysis, Upsell opportunity analysis, Credit scoring, Risk analysis, Trading exchange forecasting, Customer Lifetime Value Prediction, Price forecasting, Identify at-risk patients, Insurance product cost optimization, …

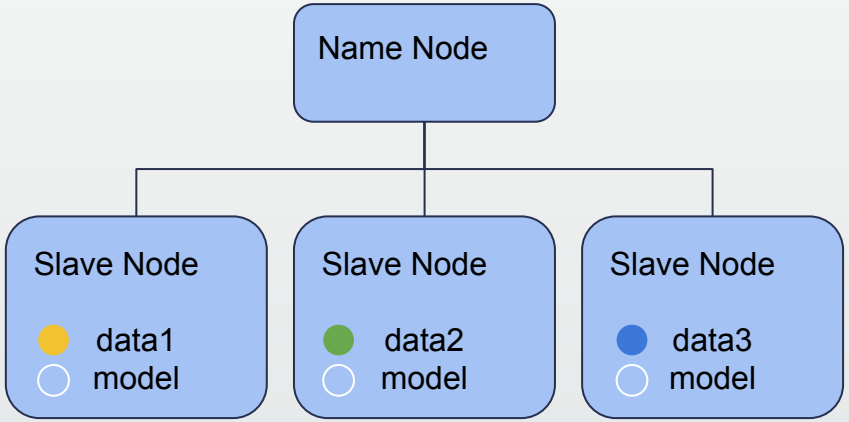
Why ML on Hadoop ?

- Hadoop is a big data platform
  - often used as a data lake
- Data already in Hadoop
  - No duplication
  - No extra data lifecycle management
- Hadoop supports ML
  - Usage of ML/DL can be integrated as a regular processing step
- Hadoop supports parallel computations and is build for massive work load

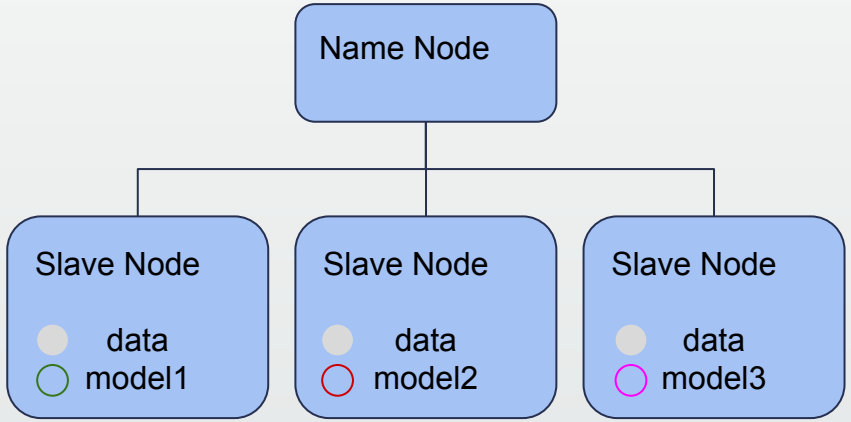
https://youtu.be/VvPaEsuz-tY

Why ML on Hadoop ?

- Parallelism
  - Either by distribution of the calculation of one algorithm over different nodes (A)
  - … or distribution of different models over different nodes with the same data (B)



- Parallelize the training of one model
- "Move processing to the data"



- Parallelize the training of different model
  - → Distributed Hyperparameter optimization

Why Hadoop (3.1)?

- Since 3.1, Hadoop supports direct GPU usage with YARN
- Advantages of GPU's
  - Dedicated memory
  - Large bandwidth (CPU: ~50GB/s; GPU: ~700GB/s)*
  - More cores (CPU: 32; GPU: ~5200 CUDA)*
- Disadvantages
  - Power consumption
  - Latency overhead → We calculate on big data, overhead doesn't play a role in the en
    → Compared to the "hadoop"-overhead, it's tiny
  - Harder to program → numpy, tf, pandas, … are vectorized/optimized, we don't need to care much

- ⇒ GPU allows parallel computations on large datasets
- ⇒ Good use the advantages of hadoop

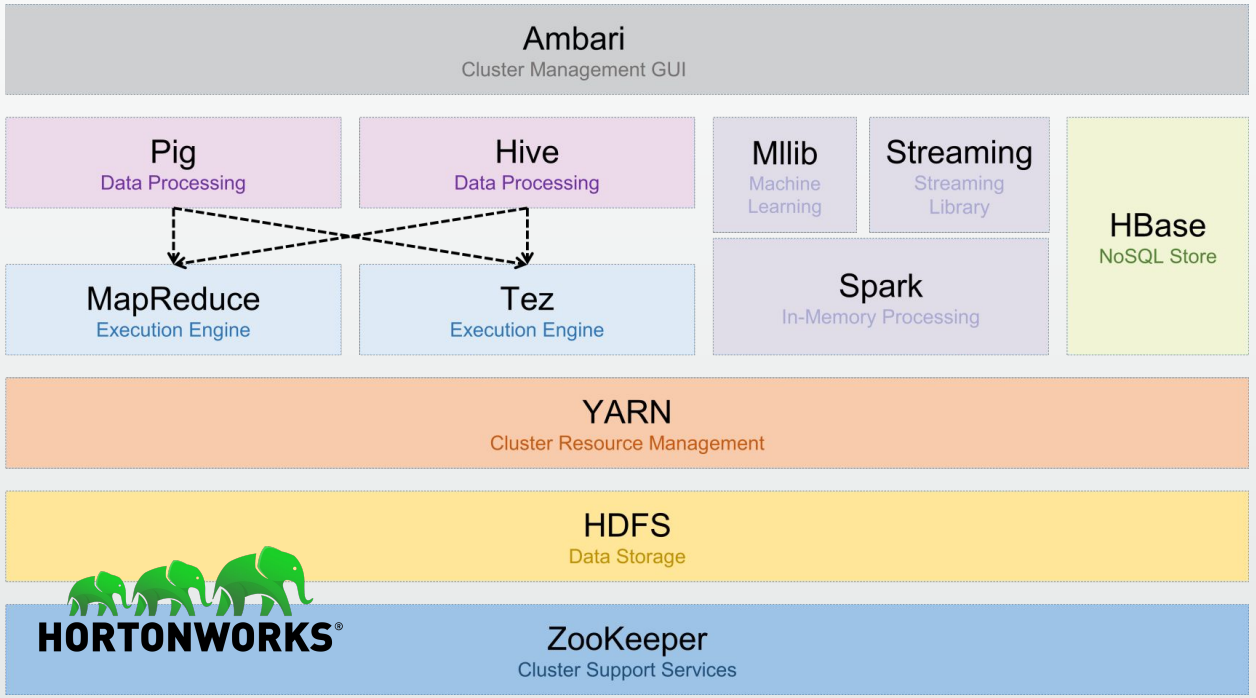
* Ryzen Threadripper 2990WX vs Titan V
Numbers rounded up

How to do ML on Hadoop

How to do ML on Hadoop ?

Option 1: Spark/Mllib/SystemML

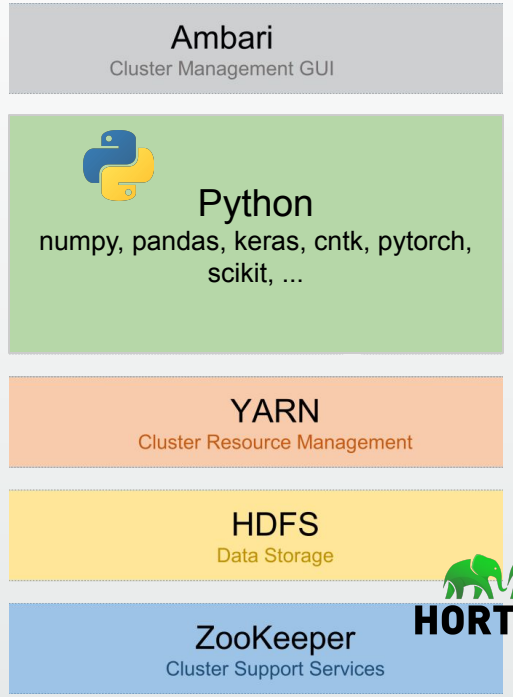
- Java
- Focus on ML, not much DL



How to do ML on Hadoop ?

Option 2. YARN/Python

- Python
- Known environment
  - numpy, pandas, scikit, etc…
- Interchangeable environment (same models everywhere)
- State-of-the-art algorithms (keras, tensorflow, cntk, sklearn, …)

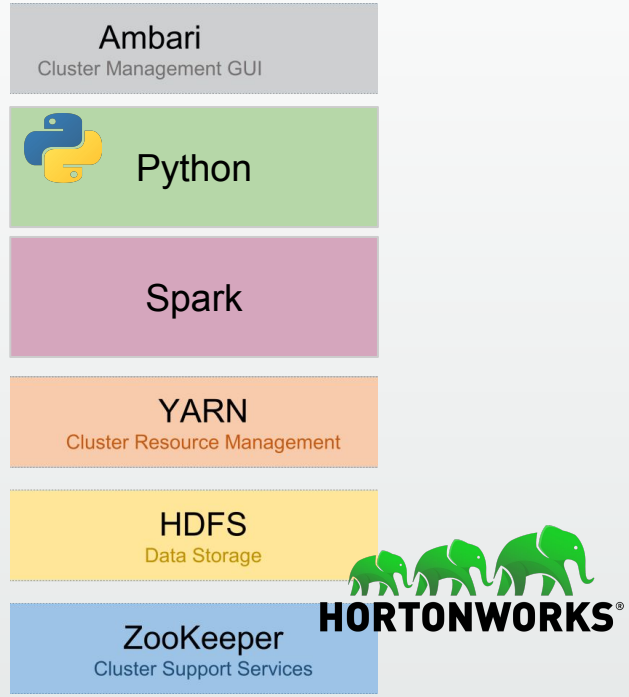


How to do ML on Hadoop ?

Option 3: PySpark

- PySpark comes with Spark
- PySpark offers Python library (**`pyspark`**)
  - **`pyspark.ml`** for ML
  - **`pyspark.sql`** for data access

- Elephas as a keras model "converter" for DL
  - **`elephas`**



Basics

Basics: Pandas

- Pandas is a high performance data structures and data analysis library
- One of the main objects is the DataFrame
  - It supports arithmetic operations on columns and rows
  - Convertible to a Spark DataFrame, to Numpy Arrays and many more

```
1:   import pandas as pd
2:   df = pd.read_csv(path, delimiter=';', names=column_names)
3:   df.head()
```

Basics: PySpark

- PySpark is an extension to Spark
- PySpark also brings ML-capabilities (**`pyspark.ml`**)

- Important Components of pyspark

  - **`DataFrame`**
    - DataFrame from Spark-SQL
  - **`Transformer`**
    - They basically transform a DataFrame into another DataFrame.
    - A trained ML model is a Transformer which transforms an input DataFrame (features) into a DataFrame with predictions
  - **`Estimator`**
    - It is fitted onto a DataFrame to produce a Transformer (trained model)
  - **`Pipeline`**
    - It pipelines/chains Transformers and Estimators to create a ML workflow

Basics: PySpark - “Connect” Example Code

```
1:  from pyspark import SparkContext, SparkConf
2:  from pyspark.sql import SQLContext, SparkSession, HiveContext


3:  sparkSession = (SparkSession.builder
                    .appName(“Demo03”)
                    .enableHiveSupport()
                    .getOrCreate())
4:  df = sparkSession.sql(“SELECT COUNT(1) FROM movielens.movies”)
5:  df.show()
```

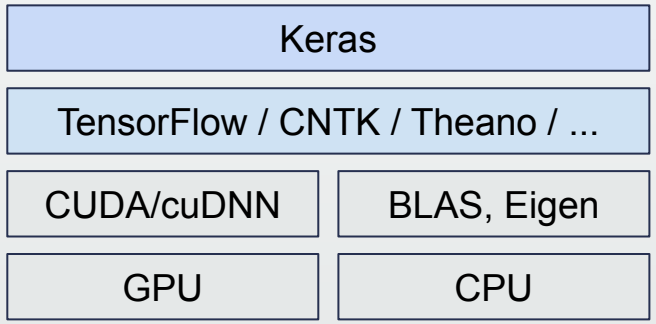
Basics: PySpark - ML Pipeline

```
1:  from pyspark.ml import Pipeline
2:  from pyspark.ml.features import VectorAssembler, StringIndexer


3:  stages = []
4:  features_proc = VectorAssembler(inputCols=['x0','x1'], outputCol='features')
5:  label_proc = StringIndexer(inputCol='y', outputCol='label')
6:  stages += [features_proc , label_proc]
7:  preprocessing_pipeline = Pipeline(stages=stages)


8:  pipeline_model = preprocessing_pipeline.fit(df)
9:  df_processed = pipeline_model.transform(df)
10: df_final = df_processed.select('features','label')
```

x0	x1	y	features	label
0.5	0.25	1	[0.5,0.25]	1
0.12	0.22	1	[0.12,0.22]	1
-1	0	0	[-1,0]	0

Basics: Keras

- Keras is a “High-Level” API for DL
- Keras supports tensorflow, cntk and theano underlying drivers (backends)
- Models can be described in a library independant way



Basics: Keras

```
1:   import keras
2:   from keras.models import Sequential
3:   from keras.layers import Dense

4:   model = Sequential()
5:   model.add(Dense(32), activation='relu', input_dim=100)
6:   model.add(Dense(64), activation='relu')
7:   model.add(Dense(64), activation='relu')
8:   model.add(Dense(4), activation='sigmoid')
9:   model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.01),metrics=['accuracy'])
10:  model.fit(x_train, y_train, epochs=25)
11:  model.evaluate(x_test, y_test)
12:  model.save('my_model.h5')

13:  model.load('my_model.h5')
14:  y_predicted = model.predict(x_input)
```

Training, Eval., Save

Use

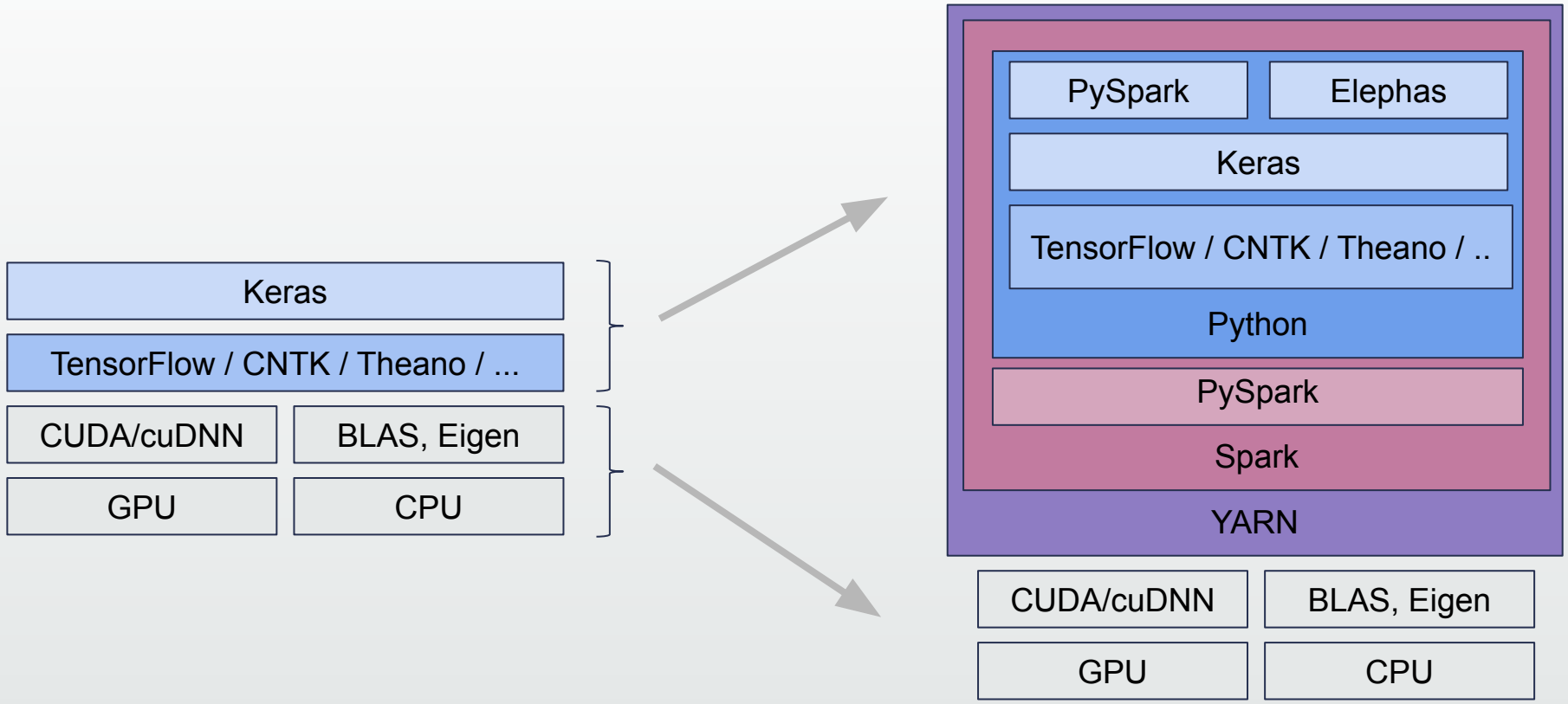
Basics: Elephas

- Elephas is an extension to enable running distributed deep learning models at scale with Spark.
- Supports
  - Data-parallel training of deep models
  - Distributed hyper-parameter optimization
  - Distributed training of ensemble models
- Elephas can import models defined in keras

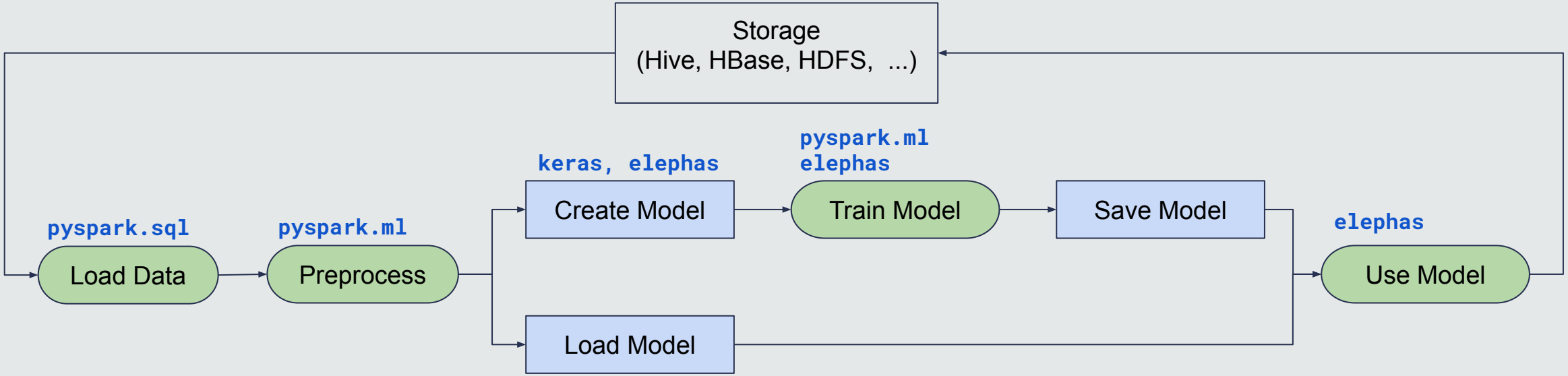
Basics: Elephas

```
1:   from elephas.ml_model import ElephasEstimator
2:   …

4:   model = get_my_keras_model()    # our keras model

10:  estimator = ElephasEstimator()
11:  estimator.setFeaturesCol("features")
12:  estimator.setLabelCol("label")
13:  estimator.set_keras_model_config(model.to_yaml())
14:  estimator.set_epoch(25)
15:  estimator.set_batch_size(64)
16:  estimator.set_num_workers(3)
17:  optimizer = optimizers.Adam(lr=0.01)
18:  optimizer_config = optimizers.serialize(optimizer)
19:  estimator.set_optimizer_config(optimizer_config)
20:  estimator.set_metrics(["acc"])
21:  estimator.setLabelCol("label_index")

22:  dl_pipeline = Pipeline(stages=[estimator])
```

Basics: The PySpark Stack



Basics: ML with PySpark - Bringing it all Together

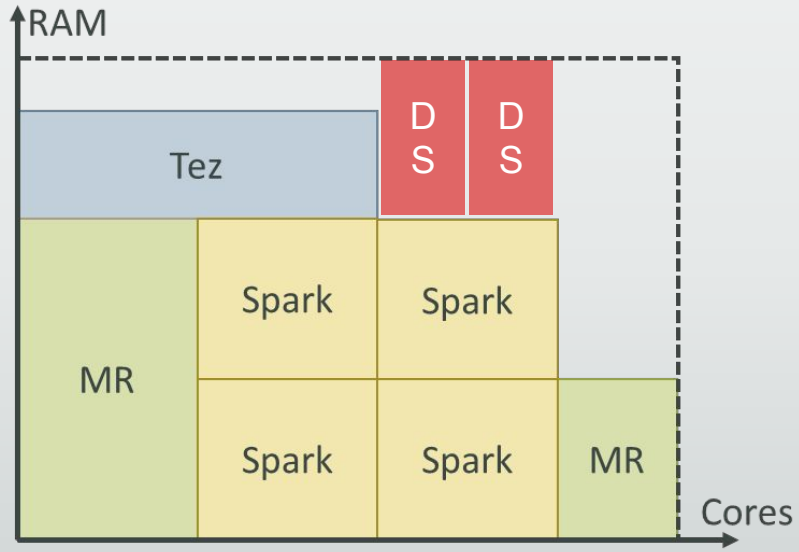


All "expensive" operations can be parallelized by Spark when using Pipelines.

Python on YARN

Apache YARN – **Y**et **A**nother **R**esource **N**egotiator

- Responsible for managing available resources in the cluster
  - YARN allocates a portion of the resources of each node (default 80%) for cluster computations
- Assigns hardware resources (CPU, RAM, GPU) to programs in the form of containers
  - Container = fixed hardware assignments that are made available to a (sub-) program
- Spark processes run e.g. in such containers
  - Separate container for each Driver and Executor process
- Scheduling of incoming requests
  - First Come First Serve
  - Default: Wait at cluster utilization



Python on YARN

- Python can run in YARN
  - Given that all dependencies are resolved
  - Use of the installed Python runtime from each worker node (slave)
  - Execute the script via the >>hadoop-yarn-applications-distributedshell<<
- Advantage
  - No modification of scripts → "just run"
  - Multiple models can be simultaneously run on different nodes → Distributed Hyperparameter optimization
- Disadvantage
  - If a script needs data from a different node, it will be transferred

Demo1 - Python Script Execution on YARN

Check: Check if GPU's are available for Yarn nodes

Call the hadoop-yarn-applications-distributedshell and execute >> nvidia-smi <<

Demo 1: Run a small example python script ("Connect" Example)

The script uses pyhive to connect to hive and counts movielens.movies

Demo1 - Python Script Execution on YARN

```
1:   from pyhive import hive

2:   cursor = hive.connect(
                host='localhost',
                username="mfollmann",
                port=10000).cursor()

3:   cursor.execute("SELECT COUNT(1) FROM movielens.movies_ext")

4:   results = cursor.fetchall()
5:   print(results)
```

Demo1 - Python Script Execution on YARN

Syntax

```
yarn jar <distributed_shell.jar>
        -jar <distributed_shell.jar>
        -shell_command <program>
        -container_resources <resources>
        -num_containers <num_containers>
```

Check if GPU's are available

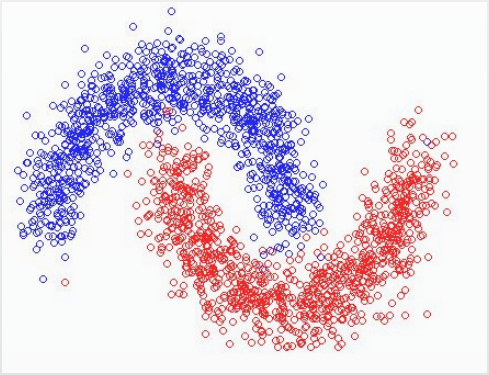
```
yarn jar /usr/hdp/3.1.0.0-78/hadoop-yarn/hadoop-yarn-applications-distributedshell.jar
        -jar /usr/hdp/3.1.0.0-78/hadoop-yarn/hadoop-yarn-applications-distributedshell.jar
        -shell_command nvidia-smi
        -container_resources memory-mb=3072,vcores=1,yarn.io/gpu=1
        -num_containers 1
```

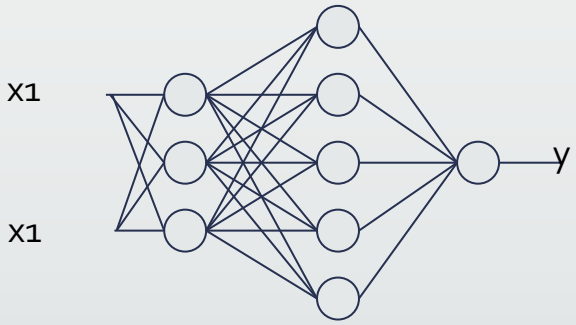
Execute a python script (Demo1 - "Connect" Example)

```
yarn jar /usr/hdp/3.1.0.0-78/hadoop-yarn/hadoop-yarn-applications-distributedshell.jar
        -jar /usr/hdp/3.1.0.0-78/hadoop-yarn/hadoop-yarn-applications-distributedshell.jar
        -shell_command "python3 /opt/python_examples/demo01.py"
        -container_resources memory-mb=3072,vcores=1,yarn.io/gpu=1
        -num_containers 1
```

Demo 2 - “Moons” ML Example

1. Import libraries
2. Create sample data set (train, test)
3. Create a neural network with keras
4. Train the neural network with our sample data set
5. Verify with test data
6. Create new random data points and apply the neural network on this random data points





Demo 2 - “Moons” ML Example

Imports

```
import numpy as np
from sklearn.datasets import make_moons

from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
```

Sample Data

```
x, y = make_moons(100, noise=0.06,
                  random_state=0)
x = (x)/np.std(x)
x_train, y_train = x[0:90], y[0:90]
x_test, y_test = x[90:99], y[90:99]

x_train = x_train.reshape((-1,2))
y_train = y_train.reshape((-1))
x_test = x_test.reshape((-1,2))
y_test = y_test.reshape((-1))
```

Model

```
model = Sequential()
model.add(Dense(3, activation='tanh',
          input_shape=(2,)))
model.add(Dense(5, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
model.compile(loss='mse',
              optimizer=Adam(),
              metrics=['accuracy'])
```

Training

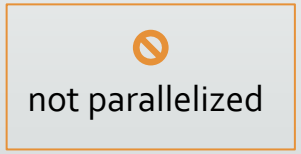
```
history = model.fit(x_train, y_train,
                    batch_size=500,
                    epochs=2000,
                    verbose=0,
                    validation_data=(x_test, y_test))
```

Eval.

```
score = model.evaluate(x_test, y_test, verbose=0)


print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Usage

```
random_data = ((np.random.rand(2500)-0.5)*5*1000)/1000
random_data = random_data.reshape((1250,2)) # 2 dim.

random_data[:,0] = random_data[:,0]+0.6

classified = model.predict(random_data)
```



Demo 2 - ML Example

Local execution of the demo with plots

```
python3 demo02_plt.py
```

Execute a python script

```
yarn jar /usr/hdp/3.1.0.0-78/hadoop-yarn/hadoop-yarn-applications-distributedshell.jar
         -jar /usr/hdp/3.1.0.0-78/hadoop-yarn/hadoop-yarn-applications-distributedshell.jar
         -shell_command "python3 /opt/python_examples/demo02.py"
         -container_resources memory-mb=3072,vcores=1,yarn.io/gpu=1
         -num_containers 3
```

Python with Spark/PySpark

Demo 3 - “Connect” PySpark Example

```
1:  from pyspark.sql import SparkSession
2:  sparkSession = SparkSession.builder
            .appName("Demo03")
            .enableHiveSupport()
            .getOrCreate()
3:  df = sparkSession.sql("SELECT COUNT(1) FROM movielens.movies_ext")
4:  df.show()
```

Execute python script

```
spark-submit /opt/python_examples/demo03.py --master yarn --deploy-mode cluster
```

Check:

Ambari > YARN > ResourceManager UI > Applications
to see your Spark Application Run

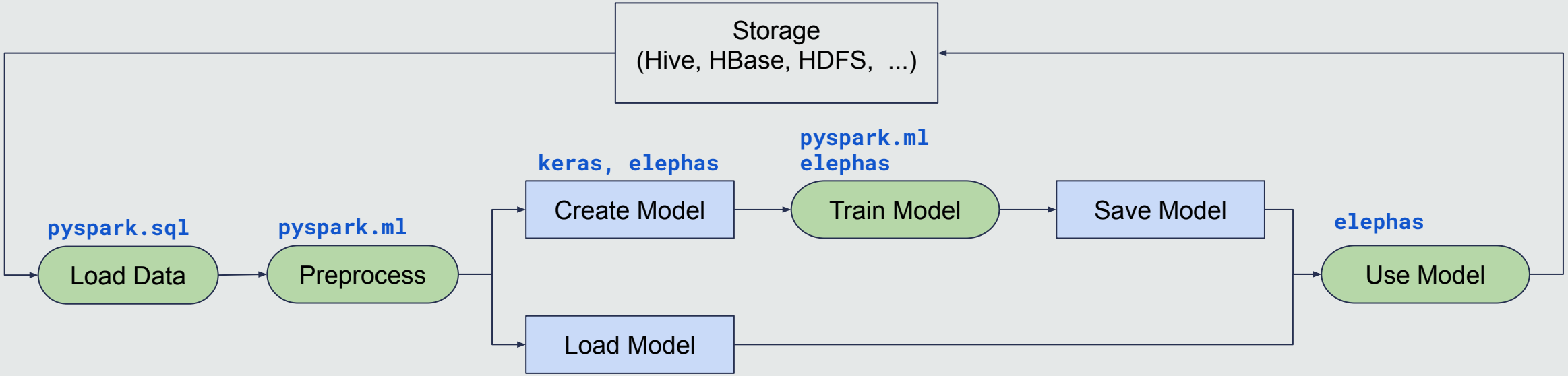
Demo 4 - “Moons” ML Example

- The “Moons” example (Demo2) will be changed so that it can run with pySpark

- What can be parallelized?
  - Loading of data
  - Preparation/Manipulation of data
  - Training of the model
  - Usage of the trained model

- Our goal is to use pipelines to parallelize
  - the data preparation
  - and the training of the model
  - Usage of the trained model

Basics: ML with PySpark - Bringing it all Together



All "expensive" operations can be parallelized by Spark when using Pipelines.

Demo 4 - "Moons" ML Example

Imports

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_moons

from elephas.ml_model import ElephasEstimator
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext, SparkSession
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml import Pipeline
from pyspark.sql.types import *
from pyspark.mllib.evaluation import MulticlassMetrics
from keras import optimizers

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam
```

Spark

```
sparkConfig = SparkConf().setAppName("Demo04")
sparkContext = SparkContext(conf=sparkConfig)
sqlContext = SQLContext(sparkContext)
```

Data

```
x, y = make_moons(100, noise=0.06, random_state=0)
x = (x)/np.std(x)
x_train, y_train = x[0:90], y[0:99]
```

Convert to Spark DataFrame

```
df1 = pd.DataFrame(x_train, columns=["x0","x1"])
df2 = pd.DataFrame(y_train, columns=["y"])
pdf = pd.concat([df1,df2], axis=1)

# pdf:
#   x0    |   x1     | y
# -0.09..|  1.38.. | 0
#  2.45..| -0.28.. | 1

schema = StructType([
                StructField("x0", DoubleType(), True),
                StructField("x1", DoubleType(), True),
                StructField("y", IntegerType(), True)])
df = sqlContext.createDataFrame(pdf, schema=schema)
                .toDF("x0","x1","y")
```

Demo 4 - "Moons" ML Example

Data Pipeline

```
stages = []
features = VectorAssembler(inputCols=["x0","x1"],
                           outputCol="features")
label = StringIndexer(inputCol="y", outputCol="label")
stages += [features, label]

pipeline = Pipeline(stages=stages)
pipeline_model = pipeline.fit(df)

df_transform = pipeline_model.transform(df)
df_transform = df_transform.select("features", "label")

train_data, test_data = df_transform.randomSplit([0.9, 0.1],
                                                 seed=1)
```

Model

```
model = Sequential()
input_dim = len(train_data.select("features").first()[0])
model.add(Dense(3, activation='tanh',
                input_shape=(input_dim,)))
model.add(Dense(5, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')
model.summary()
```

Elephas

```
estimator = ElephasEstimator()
estimator.setFeaturesCol("features")
estimator.setLabelCol("label")
estimator.set_keras_model_config(model.to_yaml())
estimator.set_categorical_labels(True)
estimator.set_nb_classes(2)
estimator.set_epochs(100)
estimator.set_batch_size(64)
estimator.set_verbosity(1)
estimator.set_validation_split(0.10)
estimator.set_num_workers(3)

optimizer_config = optimizers.Adam(lr=0.01)
elephas_optimizer_config =
          optimizers.serialize(optimizer_config)

estimator.set_optimizer_config(elephas_optimizer_config)
estimator.set_mode("synchronous")
estimator.set_loss("binary_crossentropy")
estimator.set_metrics(["acc"])

dl_pipeline = Pipeline(stages=[estimator])
```

Q: Which code line is not necessary (has no effect)?

Demo 4 - “Moons” ML Example

Train

```
fit_dl_pipeline = dl_pipeline.fit(train_data)
```

Evaluation

```
train_predicted = fit_dl_pipeline.transform(train_data)
test_predicted  = fit_dl_pipeline.transform(test_data)

train = train_predicted.select("label", "prediction")
test  = test_predicted.select("label", "prediction")

prediction_train_y = train.map(lambda row: (row['label'], row['prediction']))
prediction_test_y  = test.map(lambda row: (row['label'], row['prediction']))

metrics_train = MulticlassMetrics(prediction_train_y)
metrics_test  = MulticlassMetrics(prediction_test_y)
```

USage

```
print("Train Acc: {}".format(round(metrics_train.precision(),4)))
print("Train Conf. Matrix")
display(train.crosstab("label","prediction").toPandas())

print("Test Acc: {}".format(round(metrics_test.precision(),4)))
print("Test Conf. Matrix")
display(test.crosstab("label","prediction").toPandas())
```

parallelized

Demo 4 - “Moons” ML Example

Execute python script

```
spark-submit /opt/python_examples/demo04.py --master yarn --deploy-mode cluster
```

Now, our data is processed by Spark,
our model runs with keras and tensorflow on GPU’s managed by YARN

We are now able to train and use machine learning and deep learning models with big data

That's it :-)

Public GitHub repository
https://github.com/seamless-analytics/summer-school-2019

Our Website
https://www.seamless-analytics.de/

Competence Network

Daniel Müller
daniel.mueller@seamless-analytics.de

Marvin Follmann
marvin.follmann@seamless-analytics.de