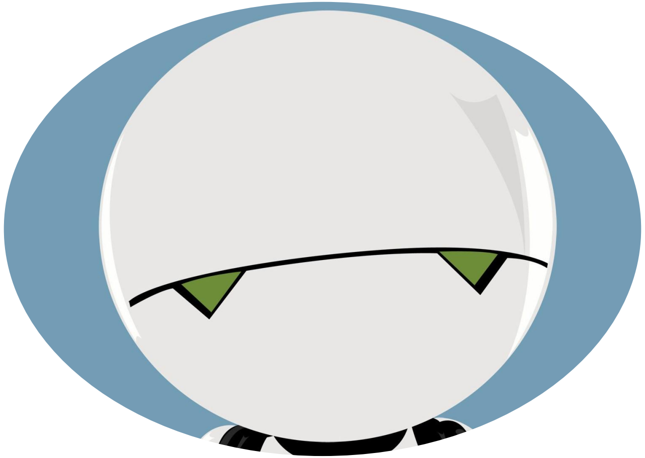
Deeplearning in production

the Data Engineer part

whoami



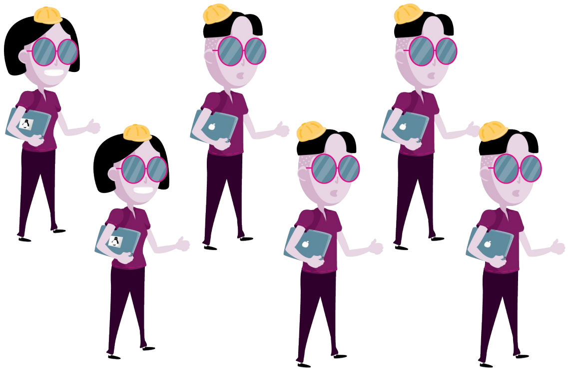
Scauglog
Data Engineer, Xebia

init project

Team Astro

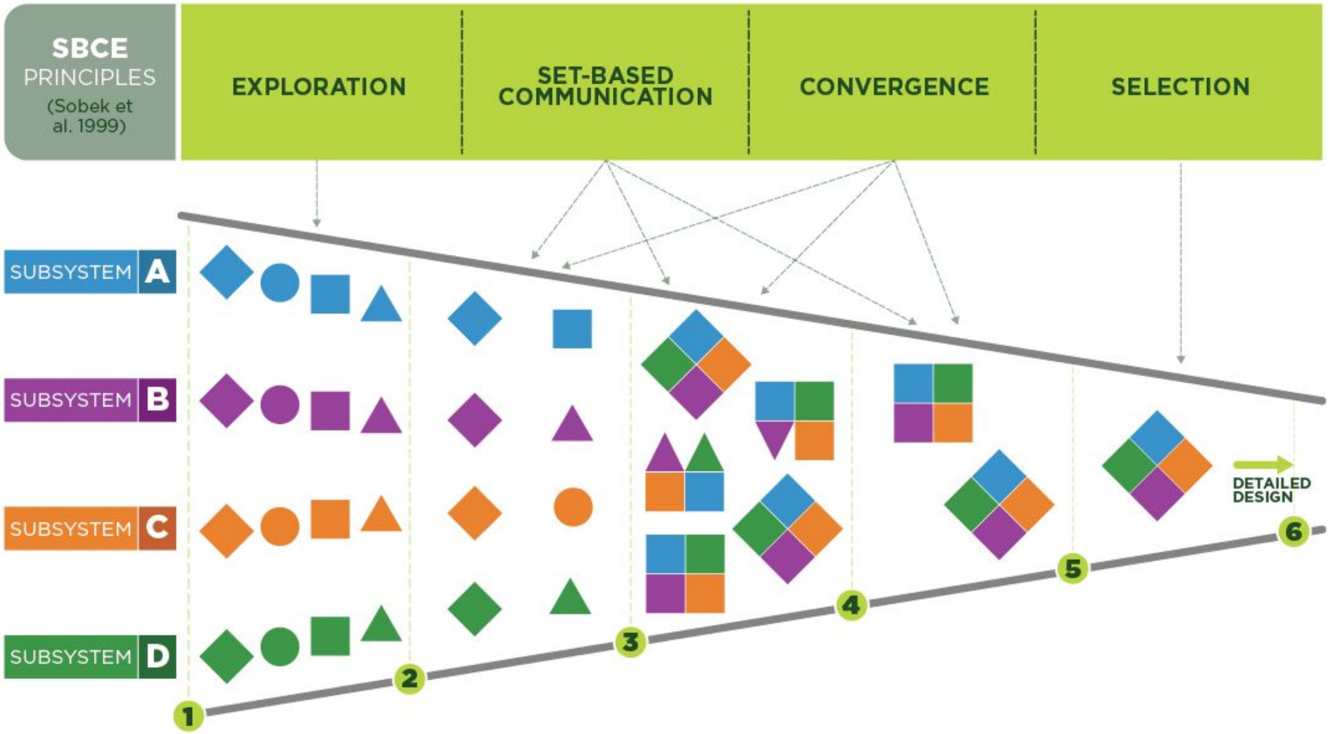
Product Owner

Scrum Master



Data Scientists, Data Engineers, Machine Learning Engineers

Choose your model



And the winner is

XGBoost

And the winner is

What is Deep Learning?

What is a Deep Learning model?

protobuf

Protocol Buffers

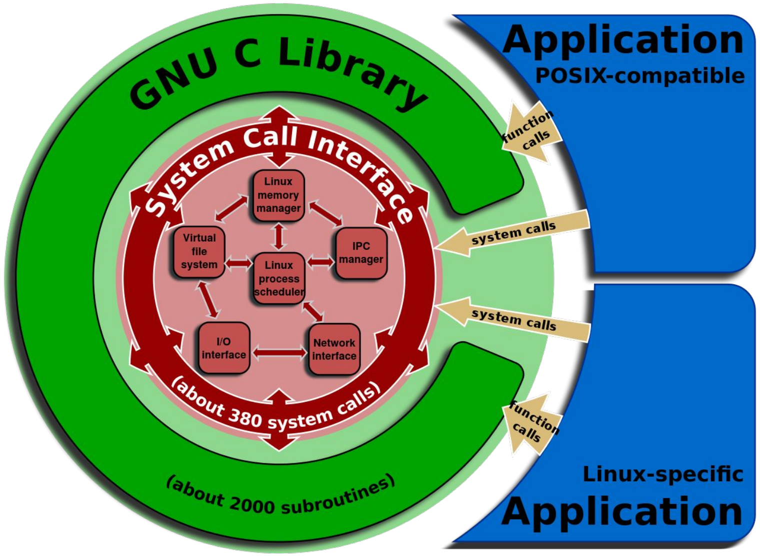
Choose your Framework

- Distributed Prediction
- Can create complexe network
- Documentation
- Community

And the winner is

TensorFlow

Wait

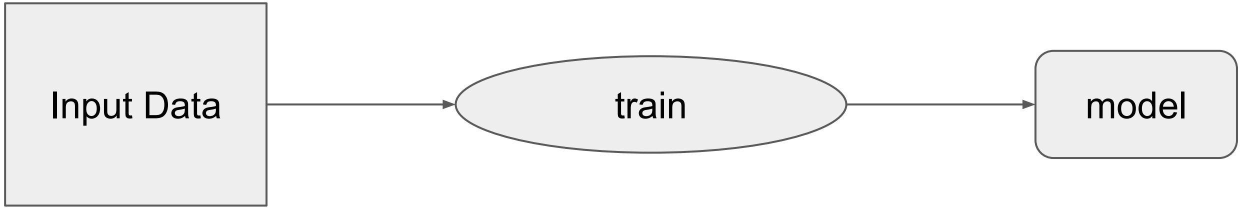

GNU C Library
(about 2000 subroutines)
System Call Interface
(about 380 system calls)
Linux memory manager
Virtual file system
Linux process scheduler
IPC manager
I/O interface
Network interface
Application
POSIX-compatible
function calls
system calls
system calls
function calls
Linux-specific
Application

And the winner is

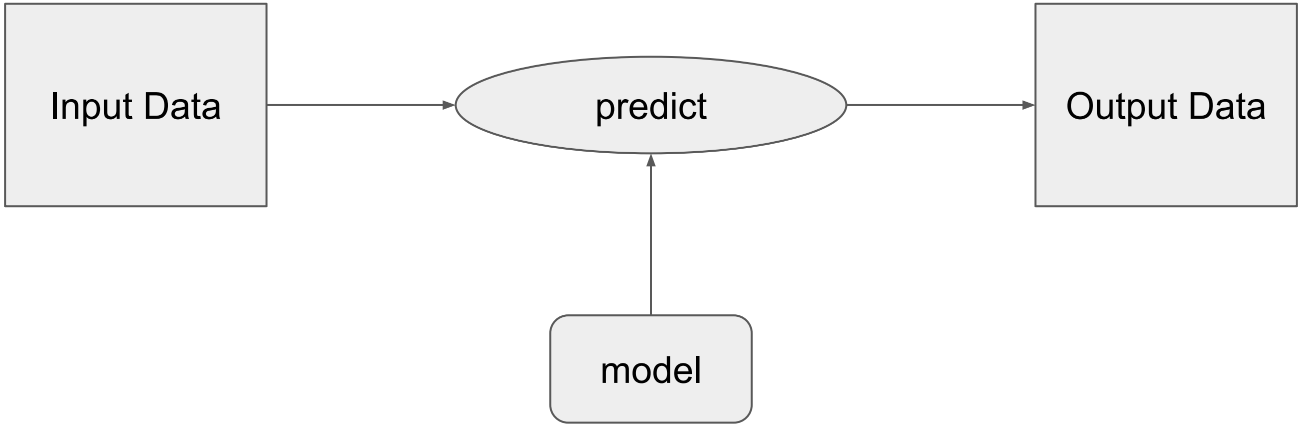
DL4J

How to Deep Learn

Train Workflow



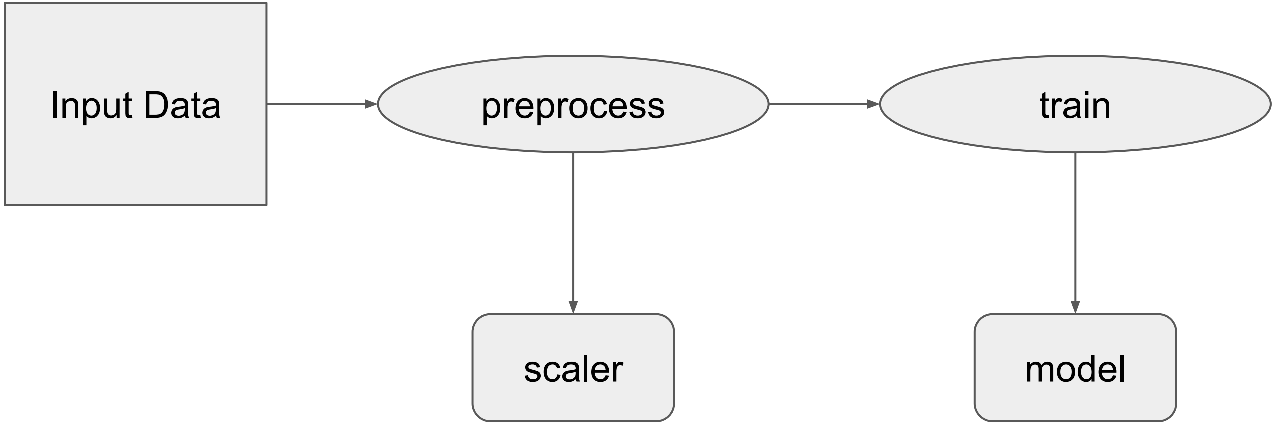
Predict Workflow



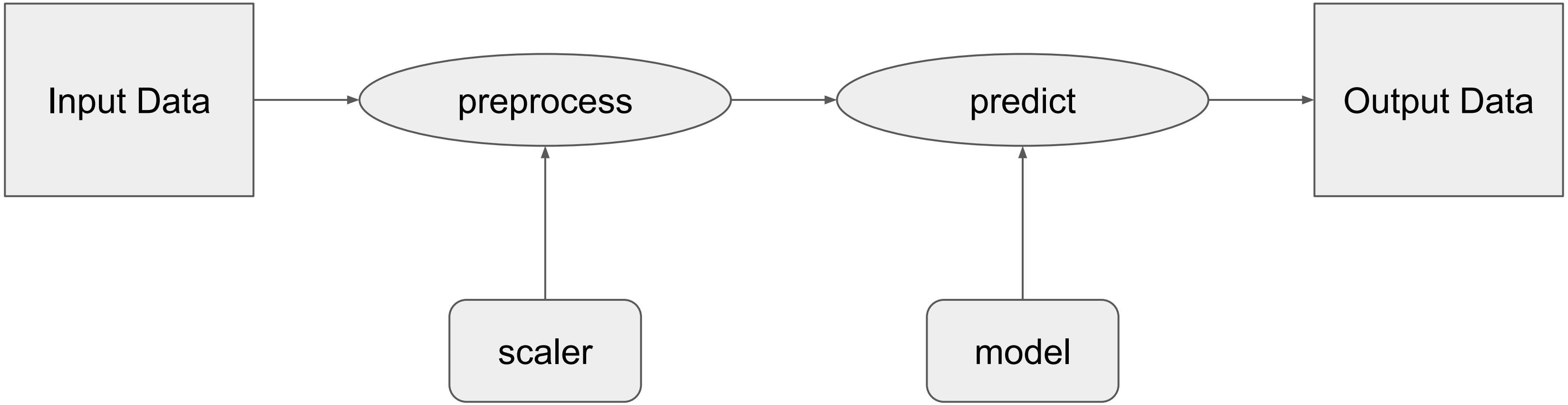
Preprocessing

- Scaling (normalisation, min max, …)
- Replace null
- Lagging

Train Workflow



Predict Workflow



Predict at scale

Scaling Prediction: naïve approach

```
def predict(input: RDD[PreprocessRow], modelPath: String): RDD[SinglePredictionRow]
= {
 input.map { row =>
   // load model
   val hdfs = FileSystem.get(new Configuration())
   val source  = new Path(modelPath)
   val model = ModelSerializer.restoreComputationGraph(hdfs.open(source), true)

   // make prediction
   val prediction = model.output(row.features)(0).getColumn(0).toFloatVector

   // return prediction
   SinglePredictionRow.fromPreprocessRow(row, prediction(0))
 }
}
```

Scaling Prediction: faster

```scala
def predict(input: RDD[PreprocessRow], modelPath: String): RDD[SinglePredictionRow]
= {
 // load model
 lazy val hdfs = FileSystem.get(new Configuration())
 lazy val source  = new Path(modelPath)
 lazy val model = ModelSerializer.restoreComputationGraph(hdfs.open(source), true)

 input.map { row =>
   // make prediction
   val prediction = model.output(row.features)(0).getColumn(0).toFloatVector

   // return prediction
   SinglePredictionRow.fromPreprocessRow(row, prediction(0))
 }
}
```

Scaling Prediction: fastest

```
def predict(input: RDD[PreprocessRow], modelPath: String):
RDD[SinglePredictionRow] = {
 // load model
 lazy val hdfs = FileSystem.get(new Configuration())
 lazy val source  = new Path(modelPath)
 lazy val model = ModelSerializer.restoreComputationGraph(hdfs.open(source),
true)

 input.mapPartitions { partition =>
   val partitionSeq = partition.toSeq
   if (partitionSeq.isEmpty) {
     val emptySeq: Seq[SinglePredictionRow] = Seq()
     emptySeq.toIterator
   } else {
     val features = partitionSeq.map(_.features).reduce( (x, y) => Nd4j.concat(0,
x, y))

     val predictions = model.output(features)(0).getColumn(0).toFloatVector
     partitionSeq.zip(predictions).map { case (row, prediction) =>
       SinglePredictionRow.fromPreprocessRow(row, prediction)
     }.toIterator
   }
 }
}
```

OOM

- ND4J Array are C++ offheap object
- Cache your dataframe or look stage details to estimate memory usage
- Set `spark.yarn.executor.memoryOverhead`
- Use ND4J workspace to properly manage memory deallocation
- Repartition your dataframe before prediction to ensure equals partition
- Set `spark.sql.shuffle.partitions`

OOM

```
def predict(input: RDD[PreprocessRow], modelPath: String, numFeatures: Int, timeSteps: Int): RDD[SinglePredictionRow] =
{
   // load model ...
  lazy val basicConfig: WorkspaceConfiguration = WorkspaceConfiguration.builder().initialSize(0)
    .policyLearning(LearningPolicy.NONE).policyAllocation(AllocationPolicy.STRICT).build()
  lazy val workspace = Nd4j.getWorkspaceManager.getAndActivateWorkspace(basicConfig, "myWorkspace")

  input.mapPartitions { partition =>
    val partitionSeq = partition.toSeq
    if (partitionSeq.isEmpty) {
      val emptySeq: Seq[SinglePredictionRow] = Seq()
      emptySeq.toIterator
    } else {
      workspace.notifyScopeEntered()
      val features = Nd4j.create(partitionSeq.flatMap(_.features).toArray, Array(partitionSeq.size, numFeatures,
timeSteps))
      val predictions = model.output(false, workspace, features)(0).toFloatVector
      workspace.notifyScopeLeft()

      partitionSeq.zip(predictions).map { case (row, prediction) =>
        SinglePredictionRow.fromPreprocessRow(row, prediction)
      }.toIterator
    }
  }
}
```

Compile

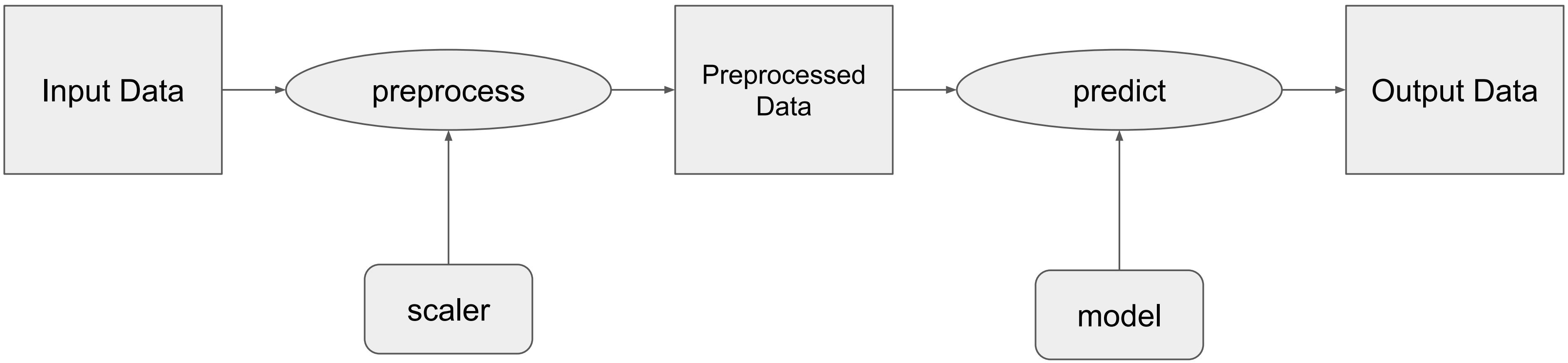
- Maven
- `-Djavacpp.platform=linux-x86_64`
- Exclude
  - `deeplearning4j-datasets`
  - `deeplearning4j-datavec-iterators`
  - `deeplearning4j-ui-components`

Training at scale

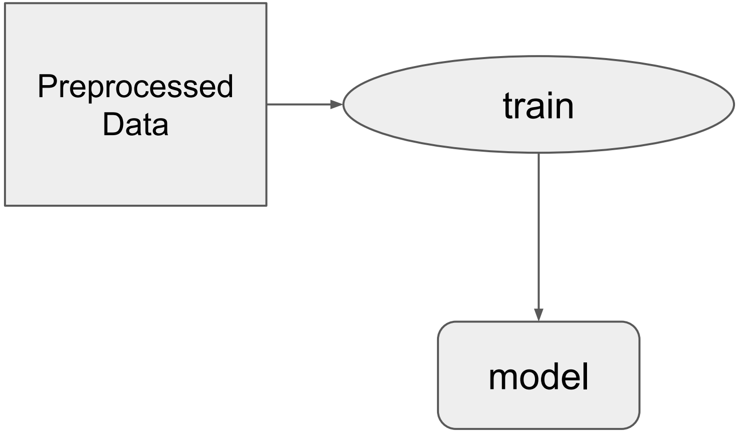
Retrain again and again and again...

- Model performance decline over time
- Hyperparameter tuning
- Deep Learning model rarely comes alone (clustering)

Predict Workflow



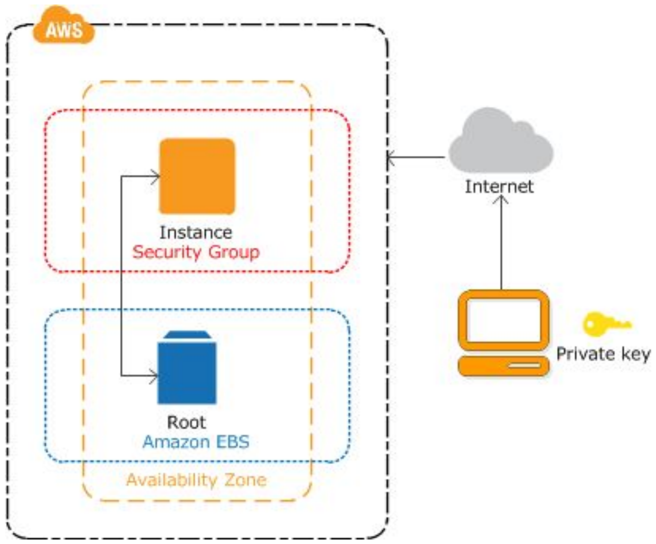
Train Workflow



Training at scale: AWS EC2

Training at scale: AWS EC2

- Create VPC
- Create Subnet associated to VPC
- Create an IGW associated to VPC
- Create a route table associated to IGW
- Create a Security Group associated to VPC
    - Authorize ssh only for my IP
- Create a key pair
- Create EC2 server with EBS volume



Training at scale: AWS EC2

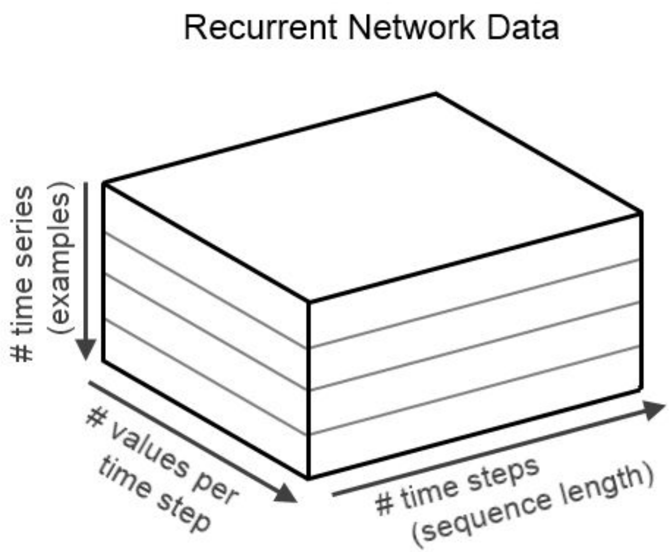
- Add ssh keys of team members
- Install cuda, cudnn, nccl and configure them
- Deploy train jar to EC2 instance
- Deploy train pipeline to EC2 instance
- Deploy preprocessed data to EC2 instance
- Deploy auto shutdown script

Training at scale: AWS EC2

- Ansible
- Transfert preprocess data to S3
- Store model in S3
- Check CPU vs GPU training time
- Keep track of training config and performance
- Share knowledge with Data Scientist
- Put your data in ESB volume if they fit

Training with DL4J: Lessons learned

- Beware of tensor order
- Prefetch data in memory (InMemoryDatasetIterator)
- Add listener to monitor your training compute performance
- Use the UI

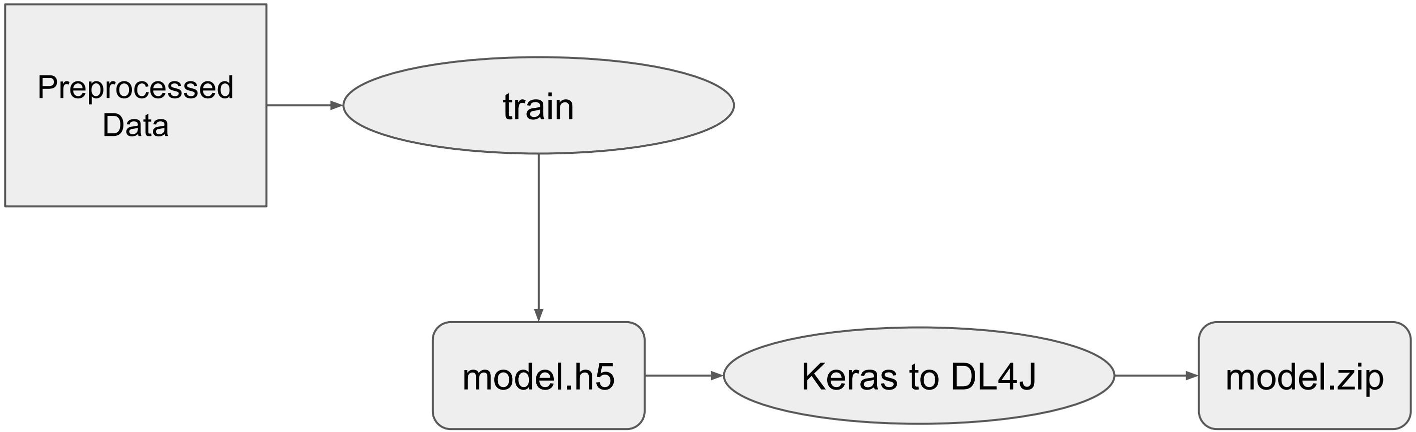


Keras to DL4J

- Data Scientist love Keras
- Keras is easier to import on notebook
- Training on Keras is faster
- Keras is compliant with cloud training (Sagemaker, CloudML)

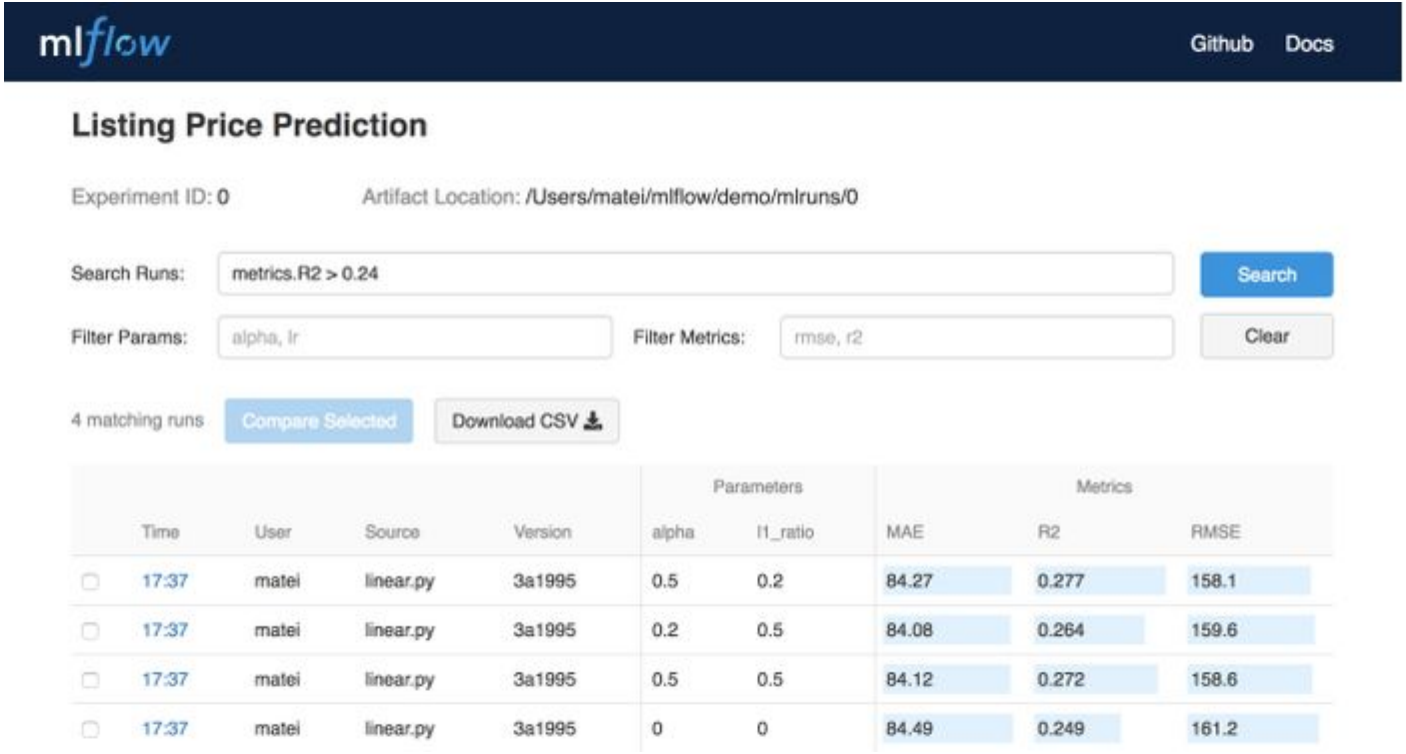
```
def execute(config: Config): Unit = {
 val kerasModel = KerasModelImport.importKerasModelAndWeights(
  config.kerasModelPath, false)
 ModelSaver.writeModel(kerasModel, config.outputModelPath)
}
```

Workflow Train



Monitoring

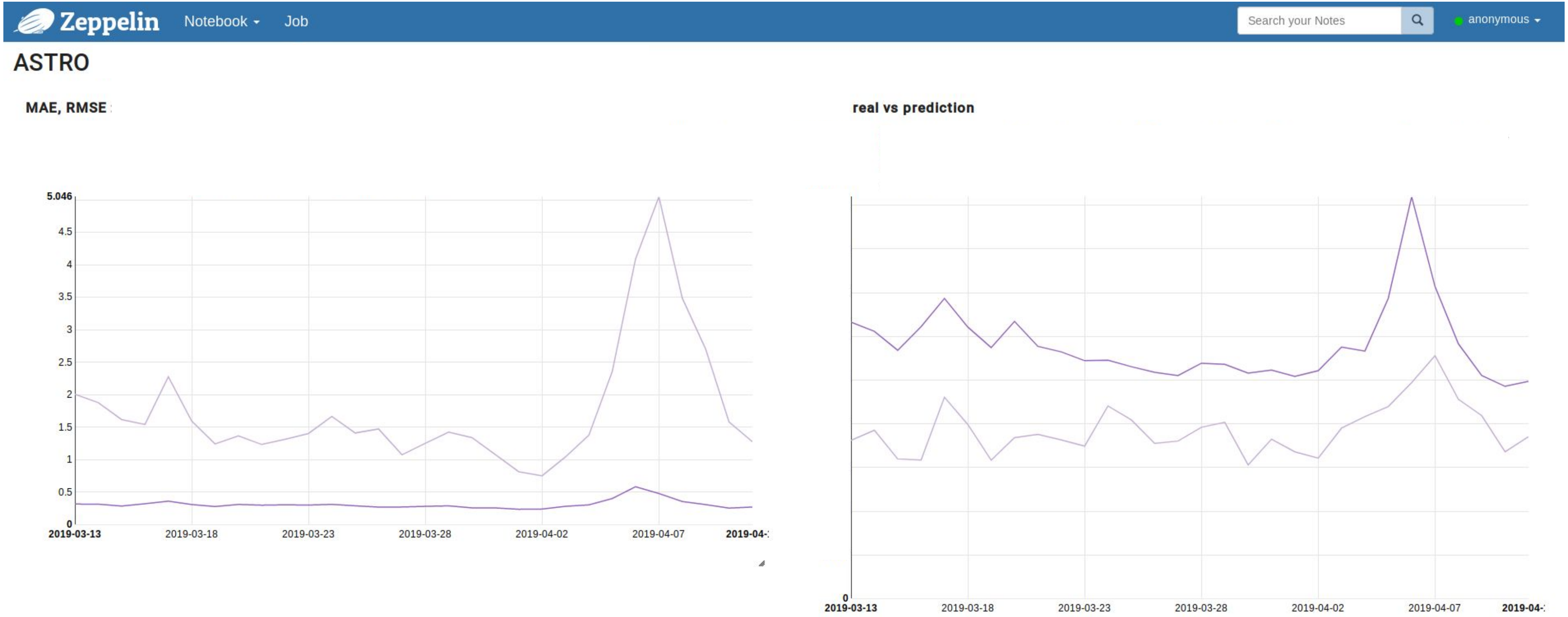
Monitoring: mlflow



Monitoring: mlflow

- Ensure your training machine can reach mlflow server
- Keep track of your experiment
  - Training parameter
  - Performance
- Compare results
- (model repository, standardize model packaging, easy deployment)

Monitoring: Zeppelin



Monitoring: Zeppelin

- Already in HDP
- Authentication
- Scheduling
- Report View
- Auto shutdown
- Can mix sources (Scala, JDBC, C*, …)
- API to automate deployment

Thank you for your attention

Any questions ?

- Data Science to Production: https://youtu.be/Gr2SS0xv0xE
- DL4J: https://youtu.be/QfnCcPcZogI
- Zeppelin: https://youtu.be/w78gZW6BQJI
- cloudML: https://youtu.be/oDpBRdjwNik