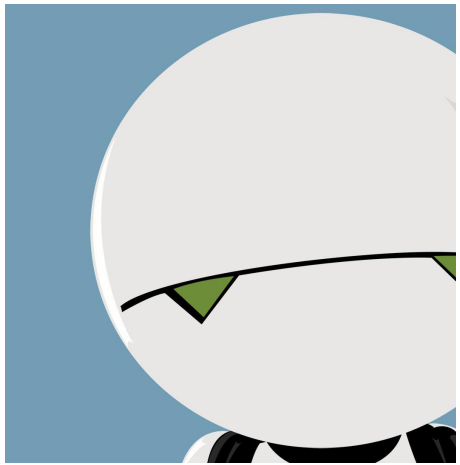

#DataXDay19



# Scauglog

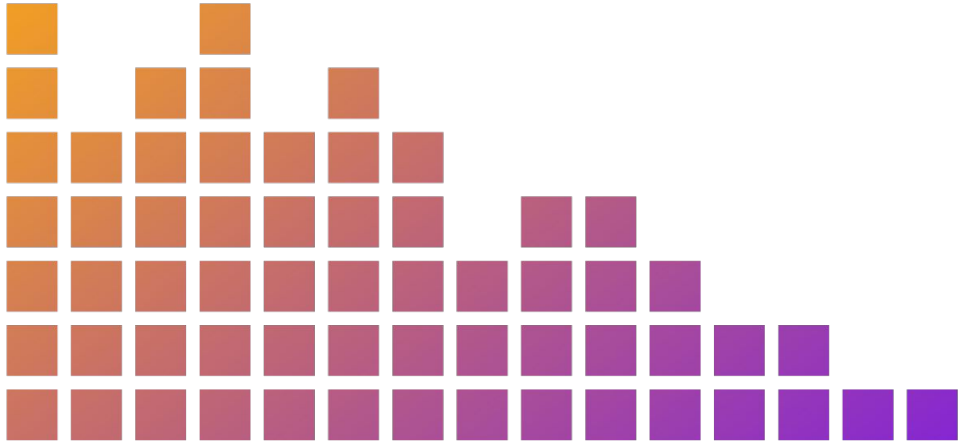
Data Engineer

Xebia

https://github.com/scauglog/prez

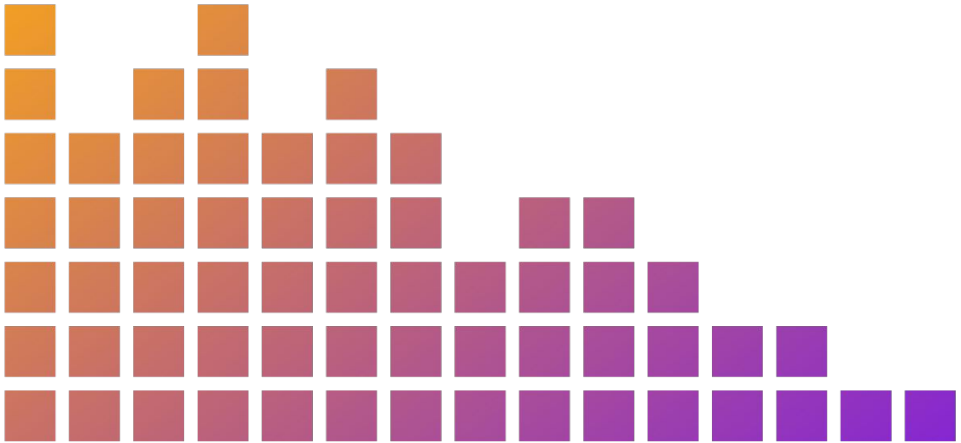
# Deeplearning in production

## the data engineer part



DATAXDAY
THE DATA CENTRIC CONFERENCE IN PARIS

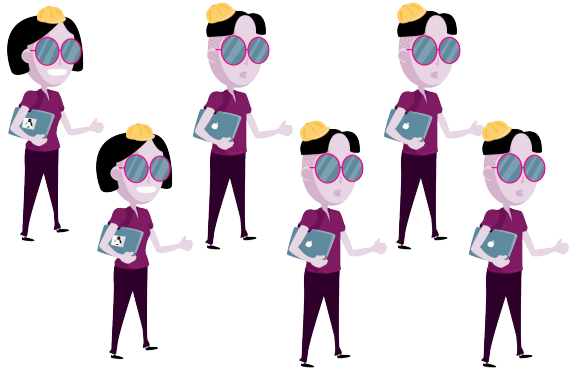
# Init Project



# Team Astro

Product Owner

Scrum Master

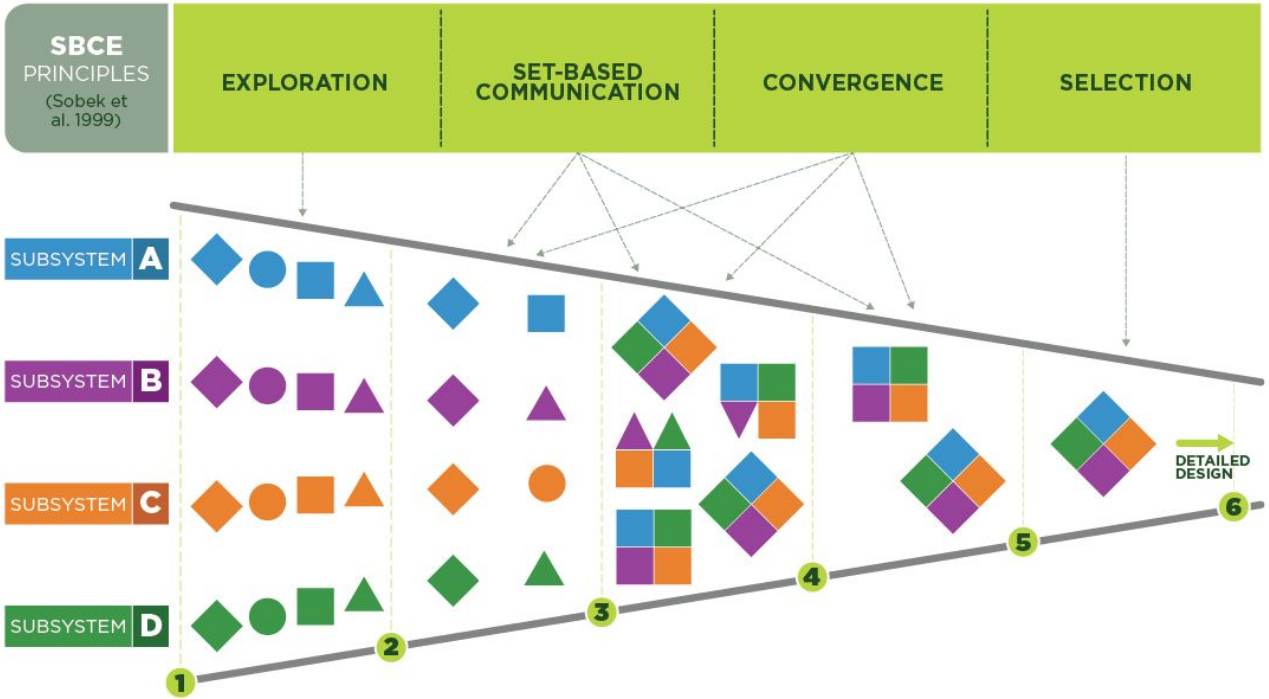


Data Scientists, Data Engineers,
Machine Learning Engineers

# Business

- Buy Sponsored link on google adwords
- 10M predictions in less than 1 Hour (~2700/s)
- Bid each Day
- Each Bid should cost less than that we earn

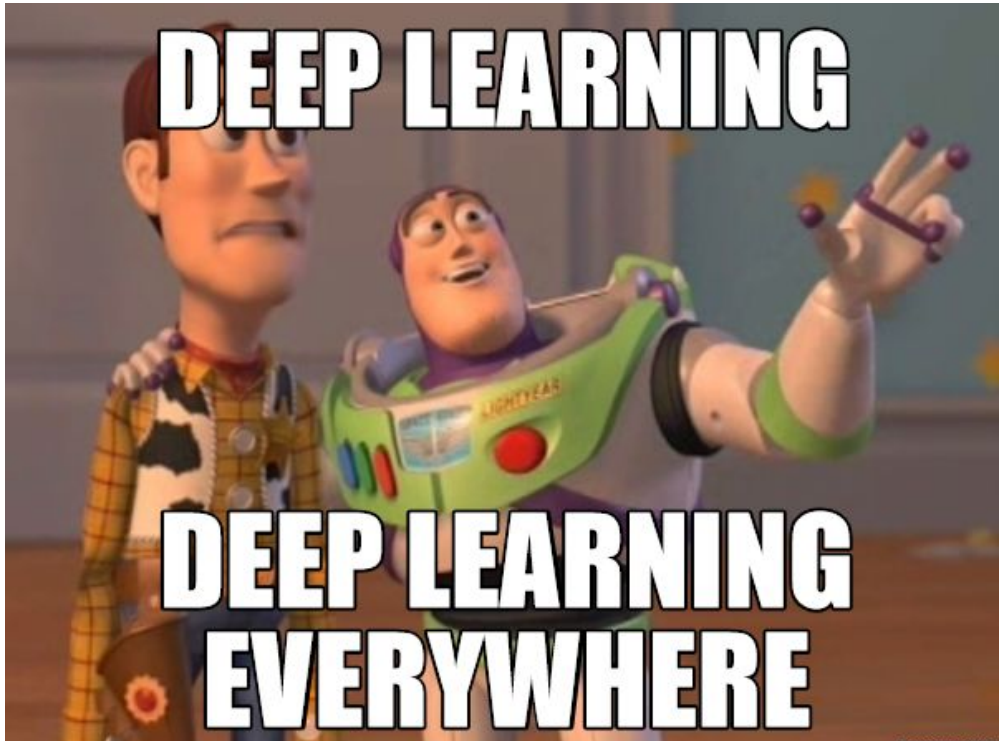
# Choose your model



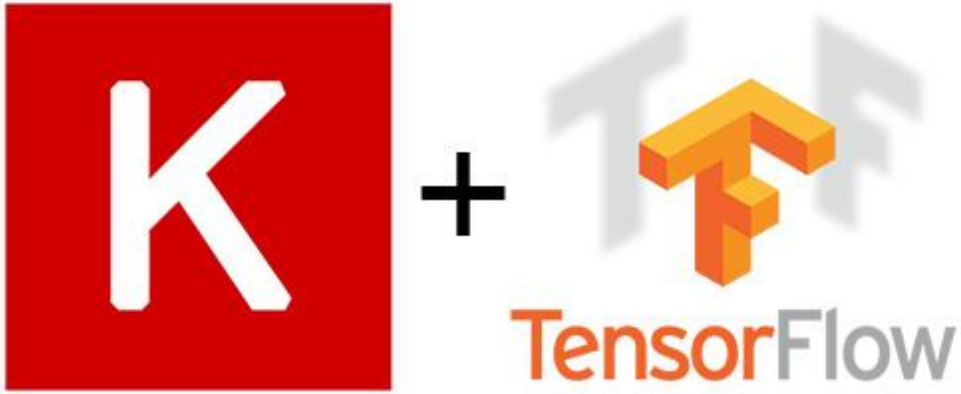
# And The winner is

# XGBoost

# And the winner is



# What is Deep Learning?



# What is a Deep Learning model?

protobuf
Protocol Buffers

# Deeplearning in Production at scale

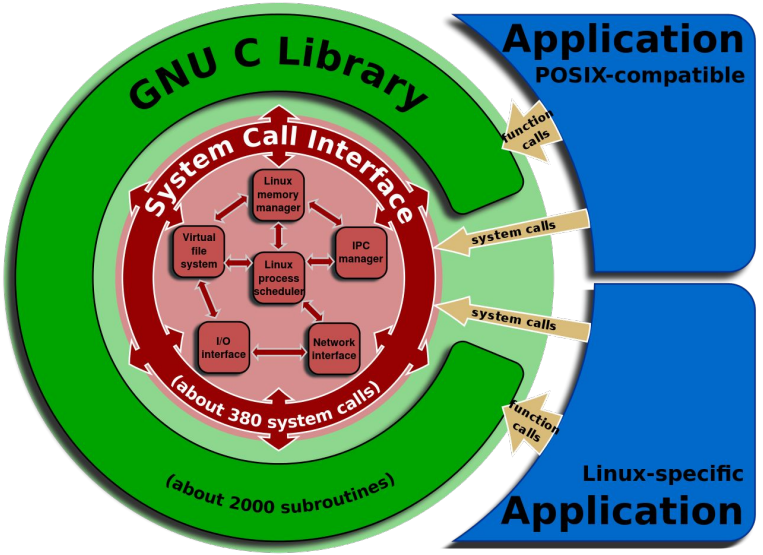

# Choose You Framework

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

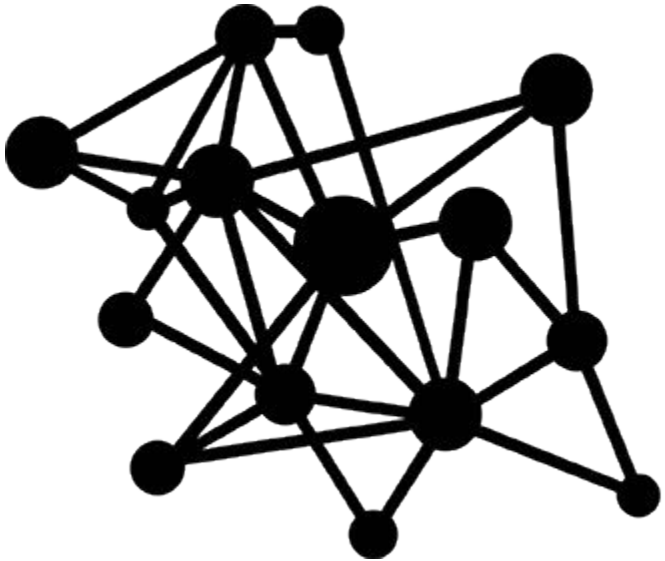
# And the winner is

# Wait


CentOS


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

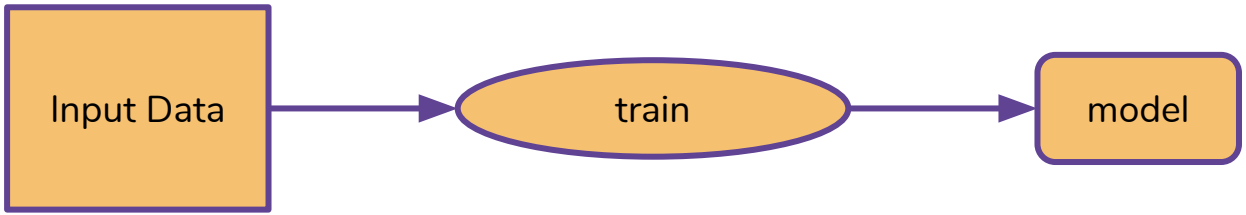
# And the winner is



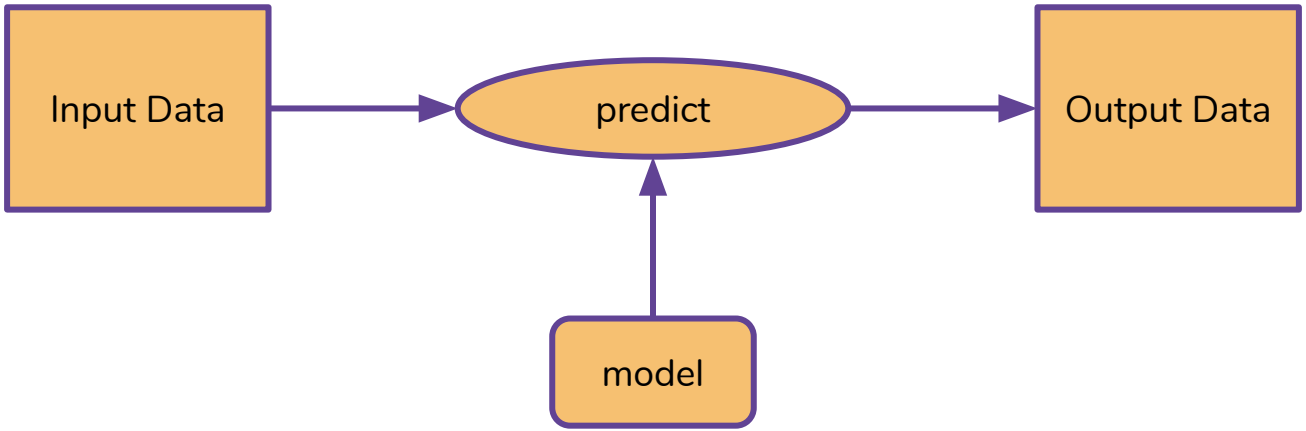
DL4J

# How to Deep Learn

# Train Workflow



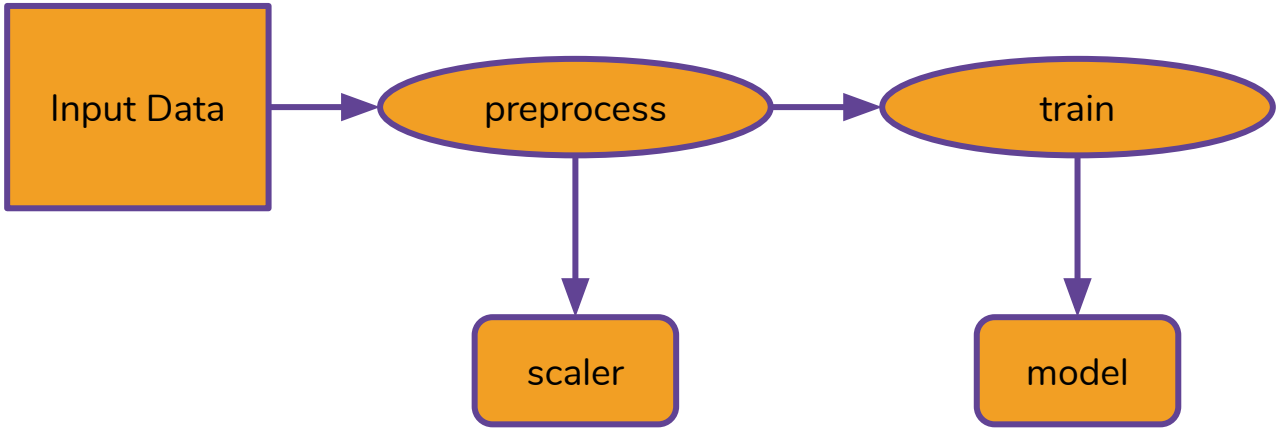
# Predict Workflow



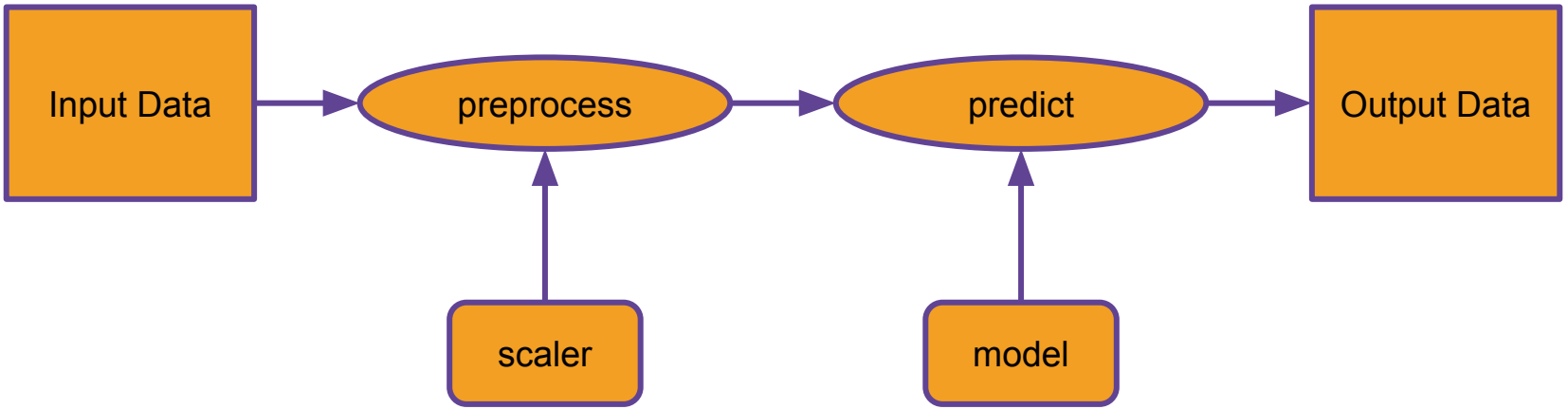
# Preprocessing

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

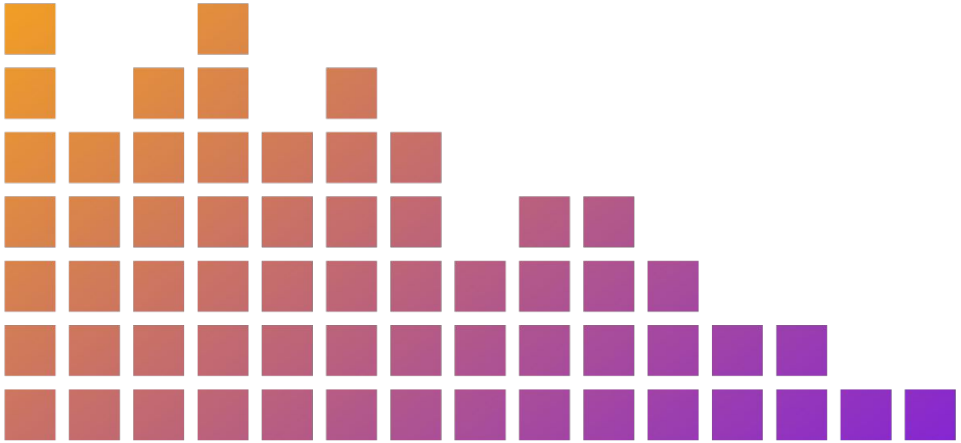
# Train Workflow



# 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

```
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) {
     Iterator(): Iterator[SinglePredictionRow]
   } 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
   }
 }
}
```

# Out Of Memory

- 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 (my god)

```
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) {
      Iterator(): Iterator[SinglePredictionRow]
    } 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`

# AVX2

The video of this presentation
will be soon available at **dataxday.fr**

Stay tuned by following
**@DataXDay**

Thanks to our sponsors

La vidéo de cette conférence
sera prochainement sur **dataxday.fr**

*Pour en être informé, restez connecté à*
**@DataXDay**

Merci à nos sponsors