詳説 Deep Learning と DL4J と MLOps 入門

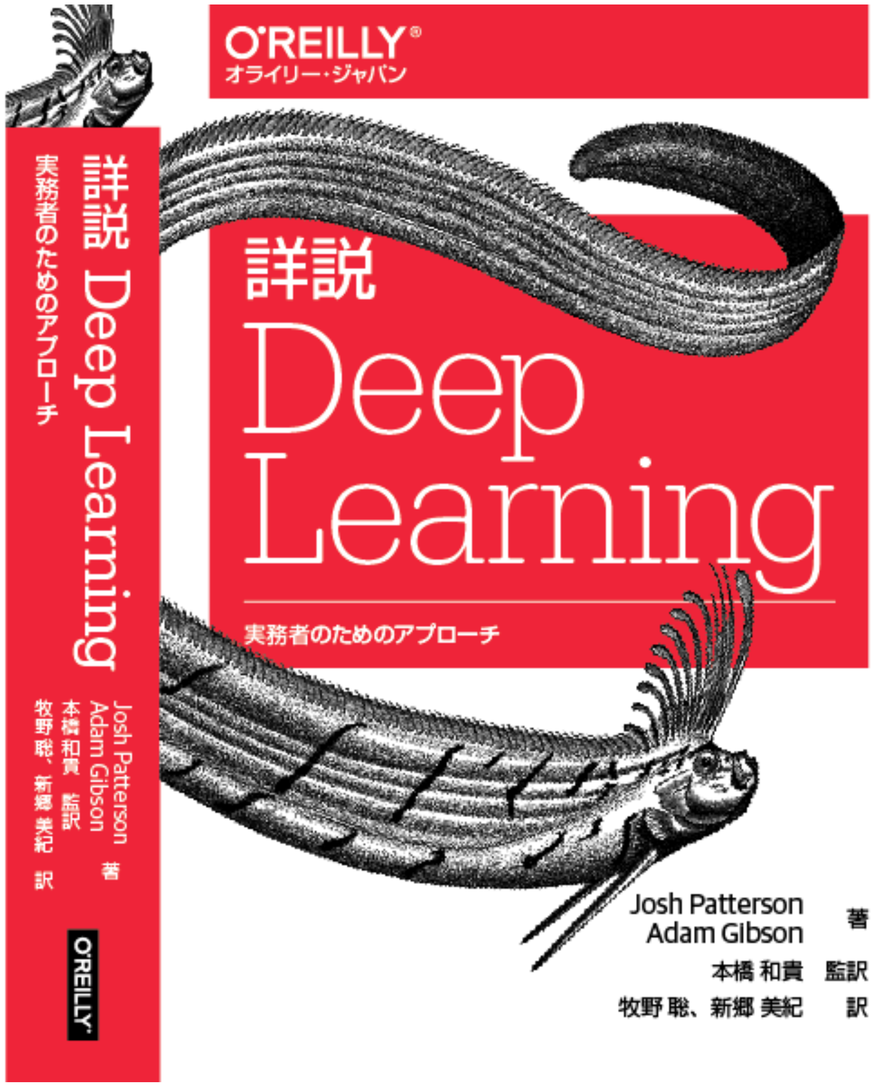
スカイマインド株式会社

本橋 和貴

自己紹介

- **本橋 和貴** @kmotohas
  - スカイマインド株式会社
    - Deep Learning Engineer (前職ではDL+ROS)
  - 素粒子物理学実験（LHC-ATLAS実験）出身
    - 博士（理学）
  - 好きな本：詳説 Deep Learning — 実務者のためのアプローチ

2019年8月9日発売

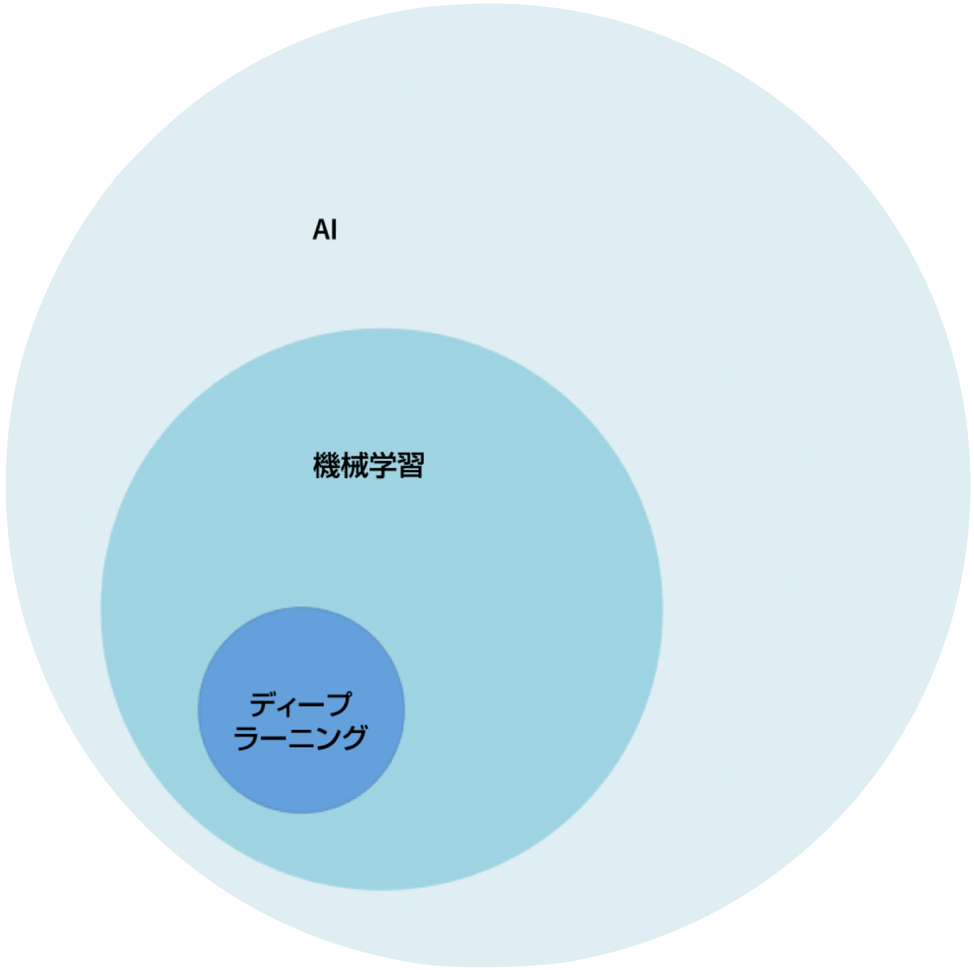


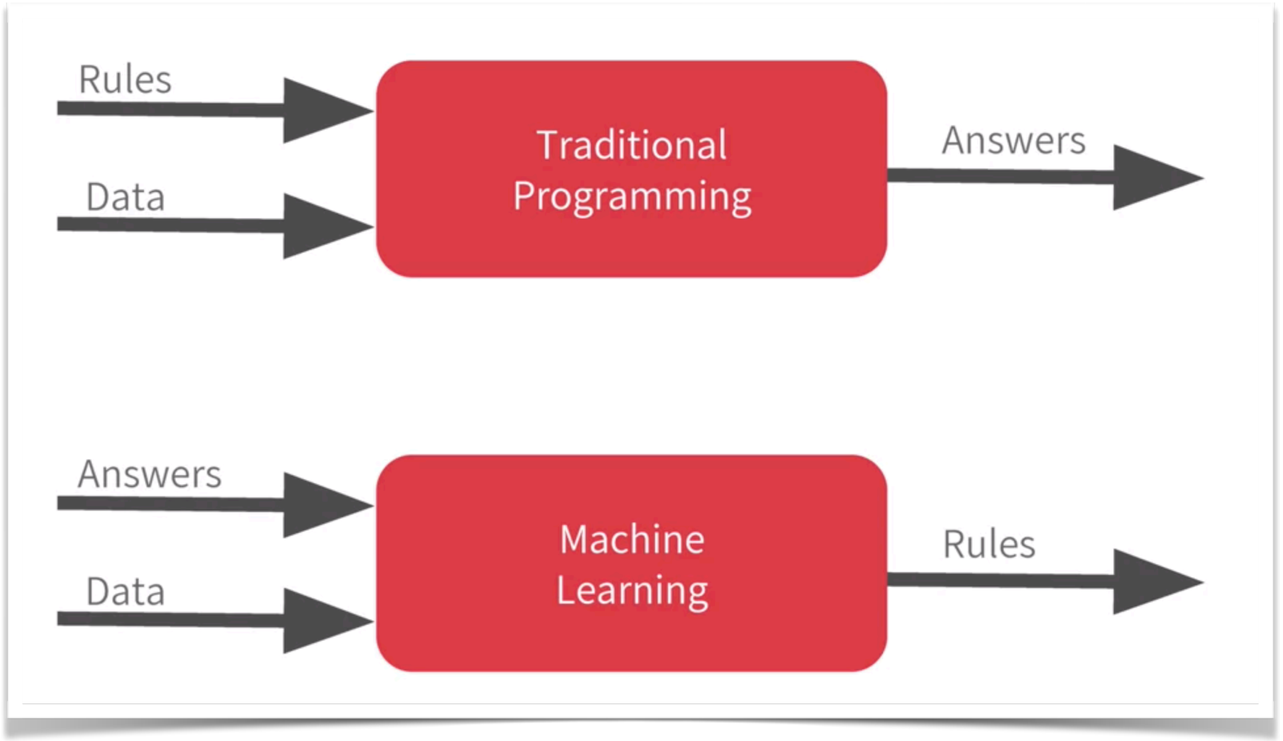
- 原著 “Deep Learning — A Practitioner’s Approach” は 2017年8月発売
- JVM言語用ディープラーニング開発フレームワーク Deeplearning4j (DL4J) を用いた解説書
  - 著者は DL4J の開発者 Adam Gibson、Skymind Inc を創業
  - ソフトウェア/アプリケーション/システム・エンジニアなどがメインターゲット
  - ディープラーニングの基礎からHadoop/Sparkといったビッグデータ分析基盤との連携まで解説

詳説 Deep Learning の 目次

- 1. 機械学習の概要
- 2. ニューラルネットワークとディープラーニングの基礎
- 3. 深層ネットワークの基礎
- 4. 深層ネットワークの主要なアーキテクチャー
- 5. 深層ネットワークの構築
- 6. 深層ネットワークのチューニング
- 7. 特定の深層ネットワークのアーキテクチャーへのチューニング
- 8. ベクトル化
- 9. Spark上でDL4Jを用いて機械学習を行う
- 付録．人工知能とは何か？、RL4Jと強化学習、etc

ディープラーニングの立ち位置





https://www.coursera.org/learn/introduction-tensorflow

DL4Jを用いた手書き数字認識モデル訓練のサンプル

```
DataSetIterator mnistTrain = new MnistDataSetIterator(batchSize, true, rngSeed);

MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(rngSeed) //include a random seed for reproducibility
    .activation(Activation.RELU).weightInit(WeightInit.XAVIER)
    .updater(new Nesterovs(rate, 0.98))
    .list()
    .layer(new DenseLayer.Builder().nIn(784).nOut(12).build())  // first  layer.
    .layer(new DenseLayer.Builder().nOut(12).build())  // second layer
    .layer(new OutputLayer.Builder(LossFunction.NEGATIVELOGLIKELIHOOD)  // output layer
    .activation(Activation.SOFTMAX)
    .nOut(10).build())
    .build();

MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
model.setListeners(new ScoreIterationListener(5));  // print the score with every iteration

for( int i=0; i<numEpochs; i++ ){
    log.info("Epoch " + i);
    model.fit(mnistTrain);
}
```

← データの準備

← 重みの初期化

← 最適化手法の選択

← モデル構造の定義

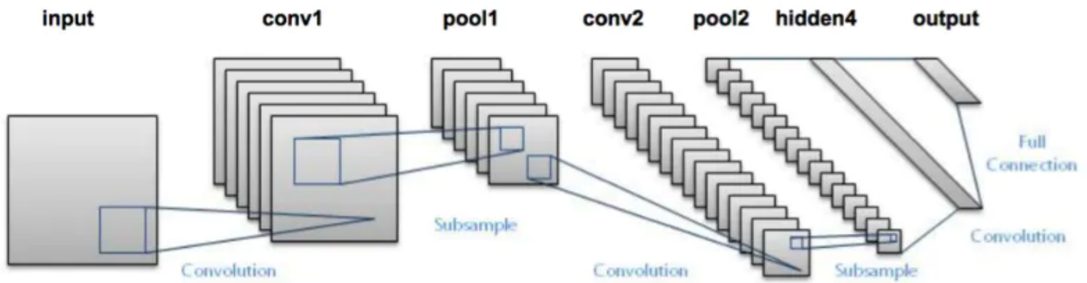
← 損失関数の定義

← 計算グラフの構築

← 繰り返し訓練

Builderパターンを用いてニューラルネットワークのコンフィグを記述

畳み込みニューラルネットワークのサンプル

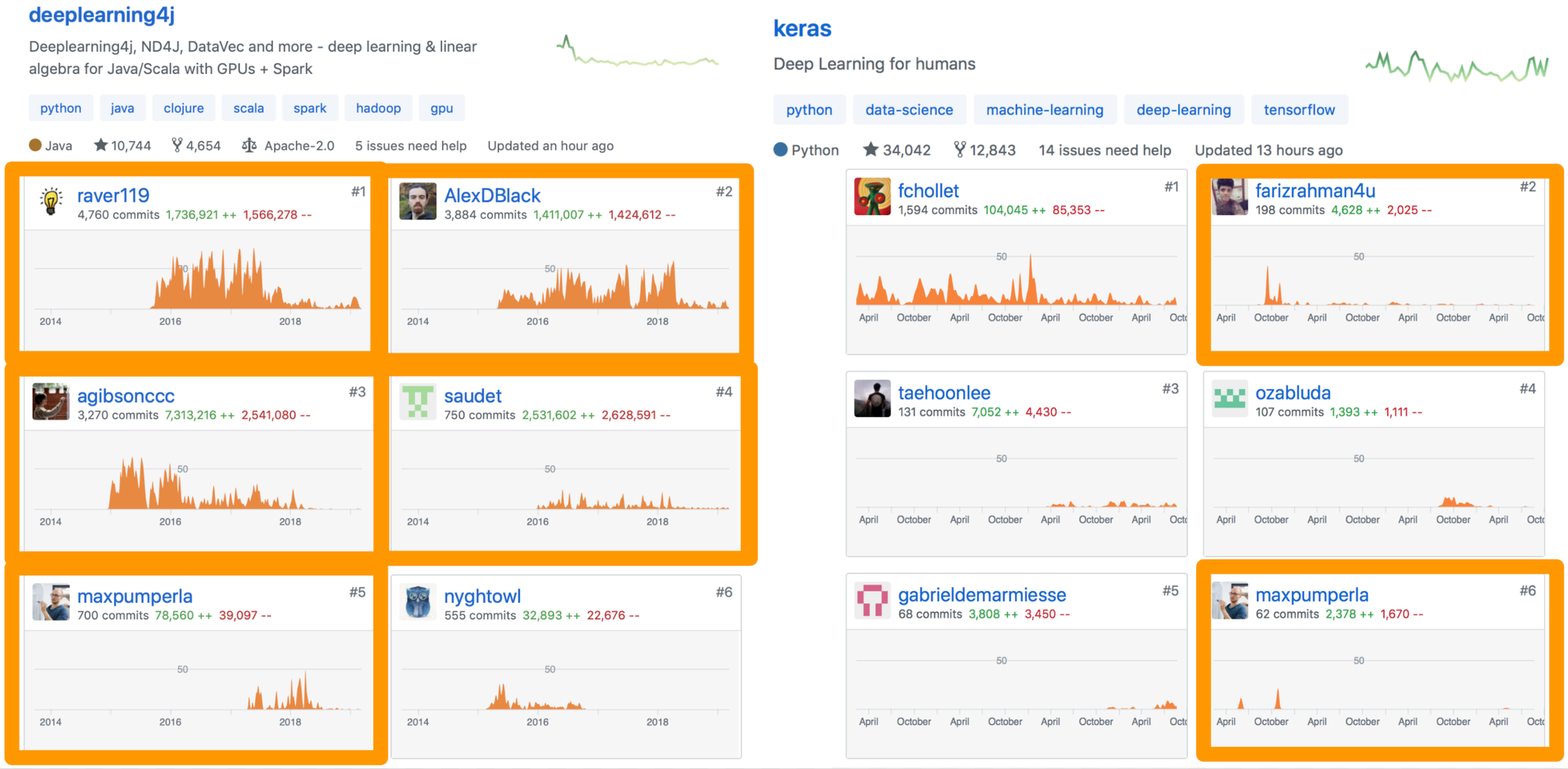



```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
     .updater(new AMSGrad(0.05))
     .l2(5e-4).activation(Activation.RELU)
     .list(
          new ConvolutionLayer.Builder(5, 5).stride(1, 1).nOut(20).build(),
          new SubsamplingLayer.Builder(PoolingType.MAX).kernelSize(2, 2).build(),
          new ConvolutionLayer.Builder(5, 5).stride(1, 1).nOut(50).build(),
          new SubsamplingLayer.Builder(PoolingType.MAX).kernelSize(2, 2).padding(2,2).build(),
          new DenseLayer.Builder().nOut(500).build(),
          new DenseLayer.Builder().nOut(nClasses).activation(Activation.SOFTMAX).build(),
          new LossLayer.Builder().lossFunction(LossFunction.MCXENT).build()
     )
     .setInputType(InputType.convolutionalFlat(28, 28, 1))
     .build()
MultiLayerNetwork model = new MultiLayerNetwork(conf);
model.init();
model.fit(...);
```

Keras-likeな高レベルAPIをJavaで

2nd/6th Kerasコントリビューターはスカイマインド所属

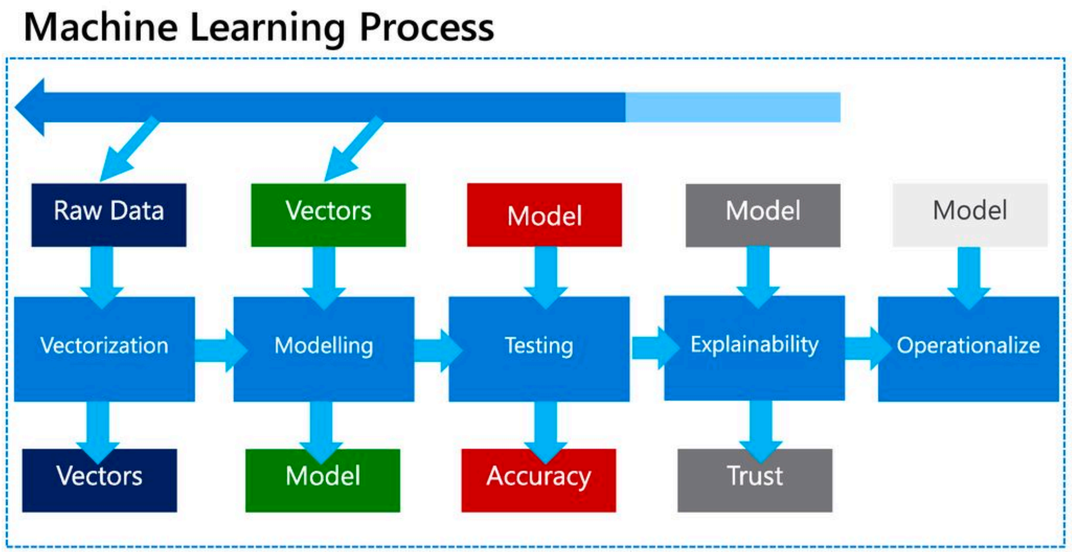
Skymind's team has contributed millions of lines of code to Open Source



DL4Jの訓練用UI

Helps with training and tuning by tracking gradients and updates works with Spark

DataVec: DL4J suite 標準のベクトル化ライブラリー



https://slideplayer.com/slide/16326648/

- ニューラルネットワークは基本的に行列計算の塊なのでまず生データをベクトルに変換する
  - DataVecはJava版Pandasのようなイメージ
- テキスト、CSV、オーディオ、画像、ビデオといったメジャーな形式のデータのベクトル化をサポート
  - CPU / GPU / Spark における実行をネイティブにサポート

ETLツールとしてのDataVec

- DataVecは生データを訓練しやすくベクトル化するためのあらゆる機能を持つ
  - InputSplit + RecordReader -> DataSetIterator -> next() -> DataSet
  - 多数のRecordReader (CSV/libsvm/matlab/json etc)
  - 前処理・正規化モジュール (MinMaxScaler, Tokenizer etc)
- 入力データのスキーマやtransform processを定義
  - 定義したプロセスをjsonにシリアル化可能
  - プロダクション環境でもポータブルに扱いやすく

DataVecのスキーマ定義のサンプル

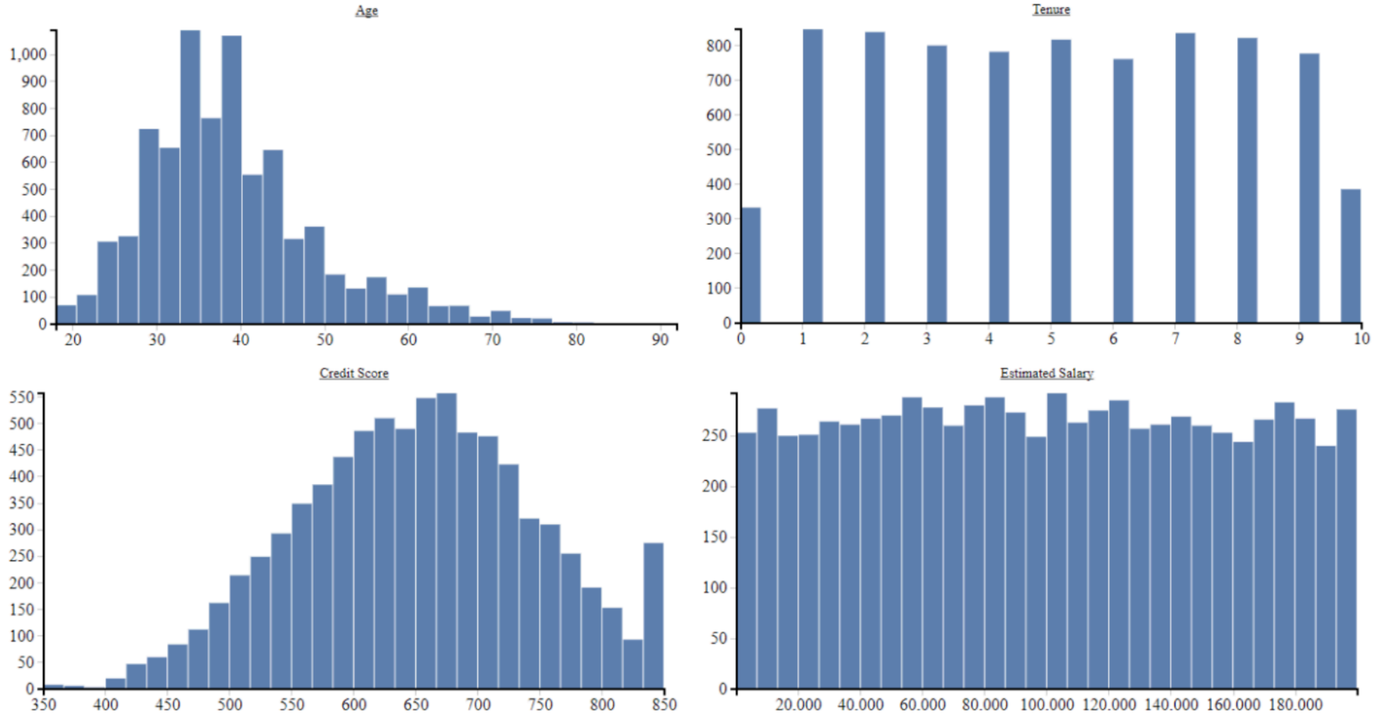
Define Schemas

```
Schema inputDataSchema = new Schema.Builder()
    .addColumnsString("CustomerID", "MerchantID")
    .addColumnInteger("NumItemsInTransaction")
    .addColumnCategorical("MerchantCountryCode",
Arrays.asList("USA","CAN","FR","MX"))
     .addColumnDouble("TransactionAmountUSD",0.0,null,false,false)
     //$0.0 or more, no maximum limit, no NaN and no Infinite values
    .addColumnCategorical("FraudLabel",
Arrays.asList("Fraud","Legit"))
    .build()
```

json化サンプル: https://gist.github.com/eraly/3b15d35eb4285acd444f2f18976dd226

DataVec Data Analysis

```
DataAnalysis dataAnalysis =
        AnalyzeSpark.analyze(schema, parsedInputData, maxHistogramBuckets);
HtmlAnalysis.createHtmlAnalysisFile(dataAnalysis, new File("DataVecAnalysis.html"));
```



skymind

Import into DL4J

Keras 1 and Keras 2 support with the same API

`KerasModelImport.importKerasSequentialModelAndWeights`

or

`KerasModelImport.importKerasModelAndWeights`

Import into DL4J

Importing the Full HDF5 File (recommended):

```
MultiLayerNetwork network =
    KerasModelImport.importKerasSequentialModelAndWeights("<H5_FILE>")
```

OR

```
ComputationGraph network =
    KerasModelImport.importKerasModelAndWeights("<H5_FILE>")
```

skymind

Inference with a DL4J model

```
//Import model
 model = KerasModelImport.import...

//Featurize input data into an INDArray
INDArray features = …

//Get prediction
INDArray prediction = model.output(features)
```

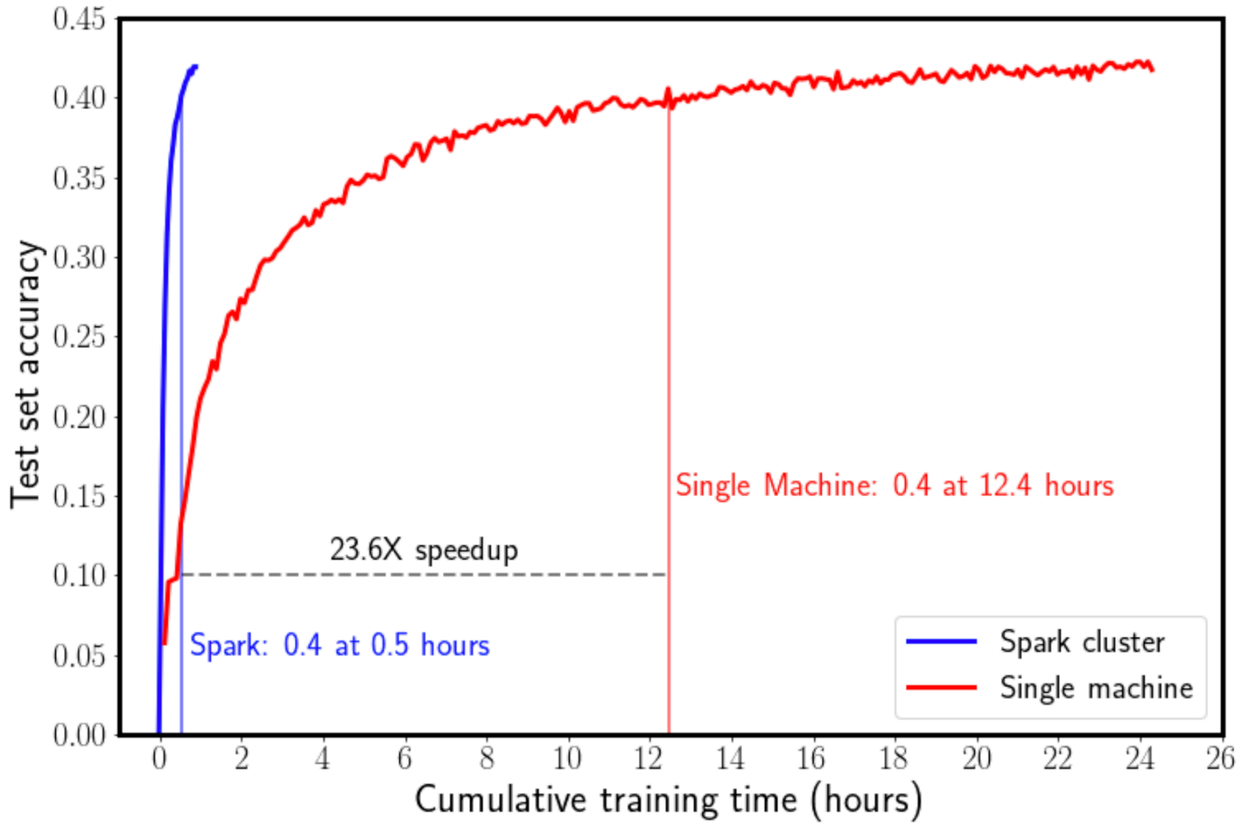
skymind TF Model Import, Open Source

One liner, currently in beta-4 release

```
SameDiff graph =
                    TFGraphMapper.getInstance()
                        .importGraph(new File(<FROZEN_PB_FILE>));
```

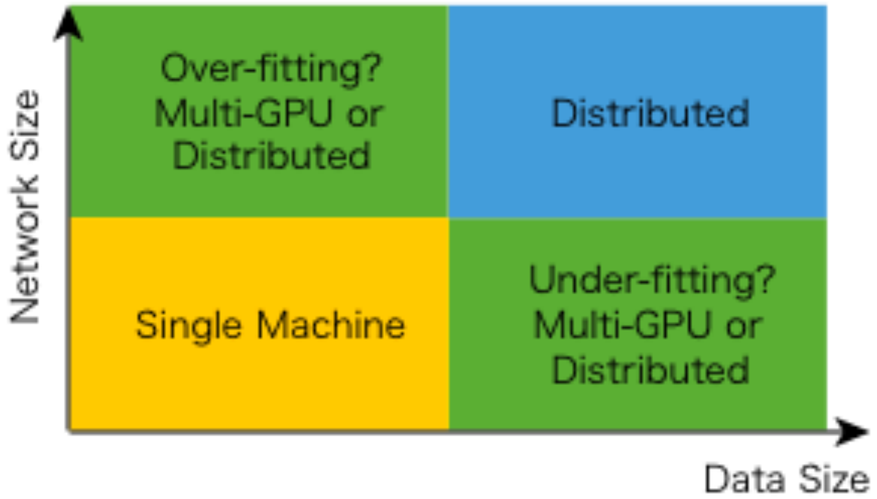
Sparkを用いた分散学習機能

- Spark: Hadoopベースのビッグデータ分散処理基盤



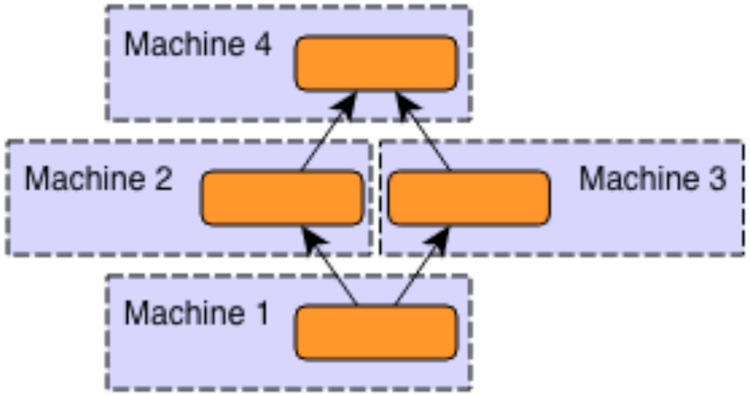
いつ分散学習を行うか

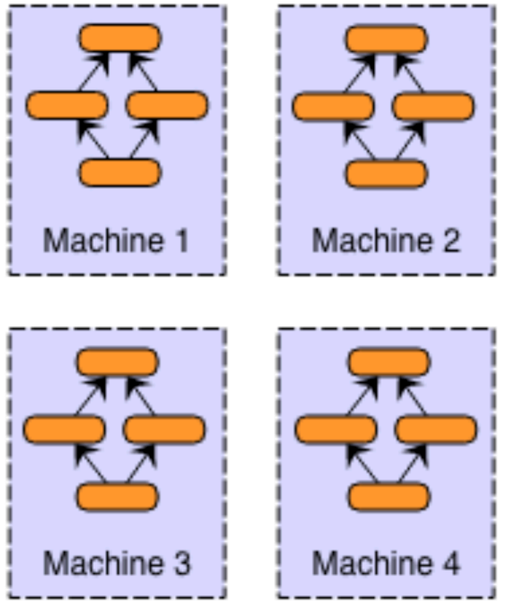
- 一台のPC / サーバーで十分なときはそれでOK
  - Multi-GPUのマシーンは複雑なネットワーククラスターよりも高性能なことも
  - DL4J は ParallelWrapper クラスを使えば multi-GPU で並列訓練できる
- データ量やモデル自体が大きすぎて訓練に時間がかかりすぎる場合は分散学習
  - クラスターの通信・同期のコストとのトレードオフ



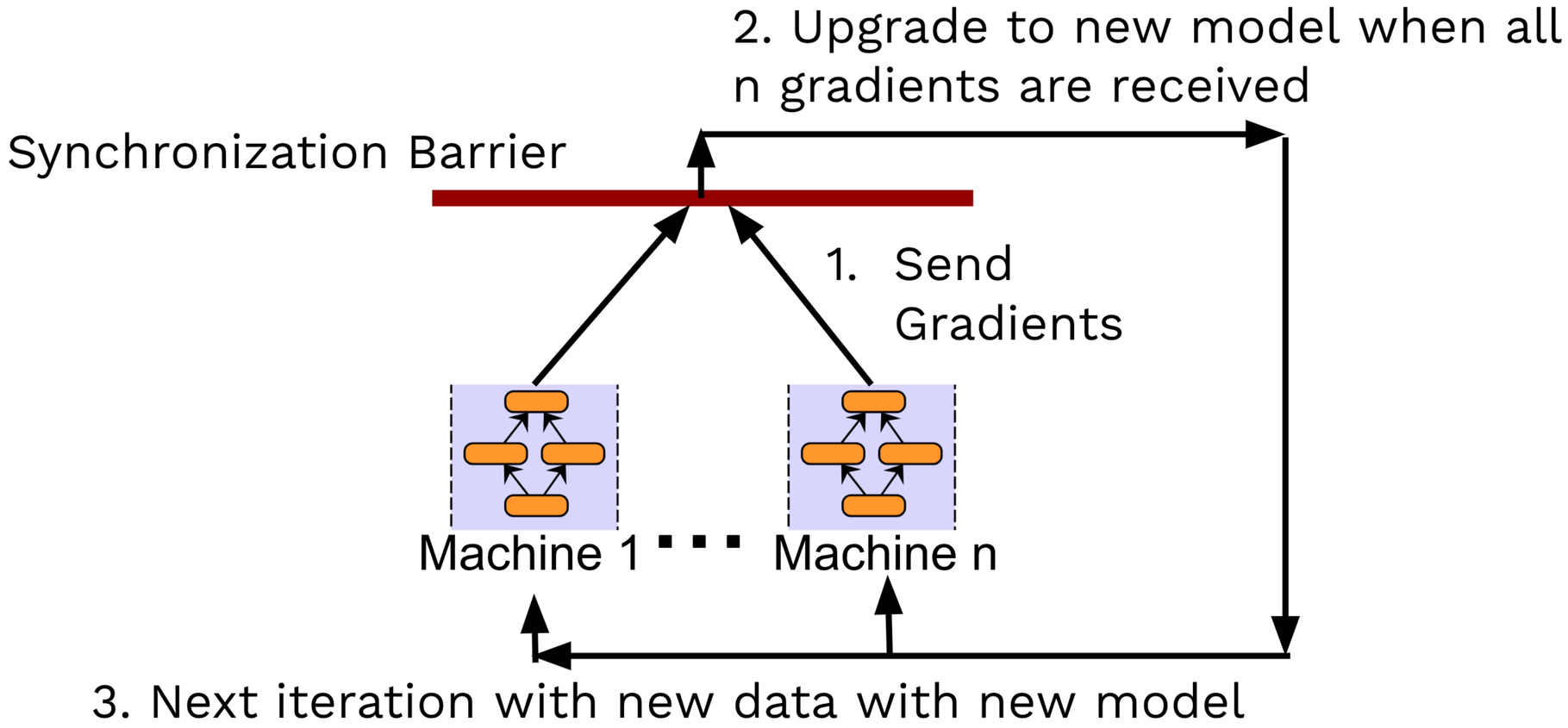
分散学習のアプローチ

Model parallelism vs Data parallelism

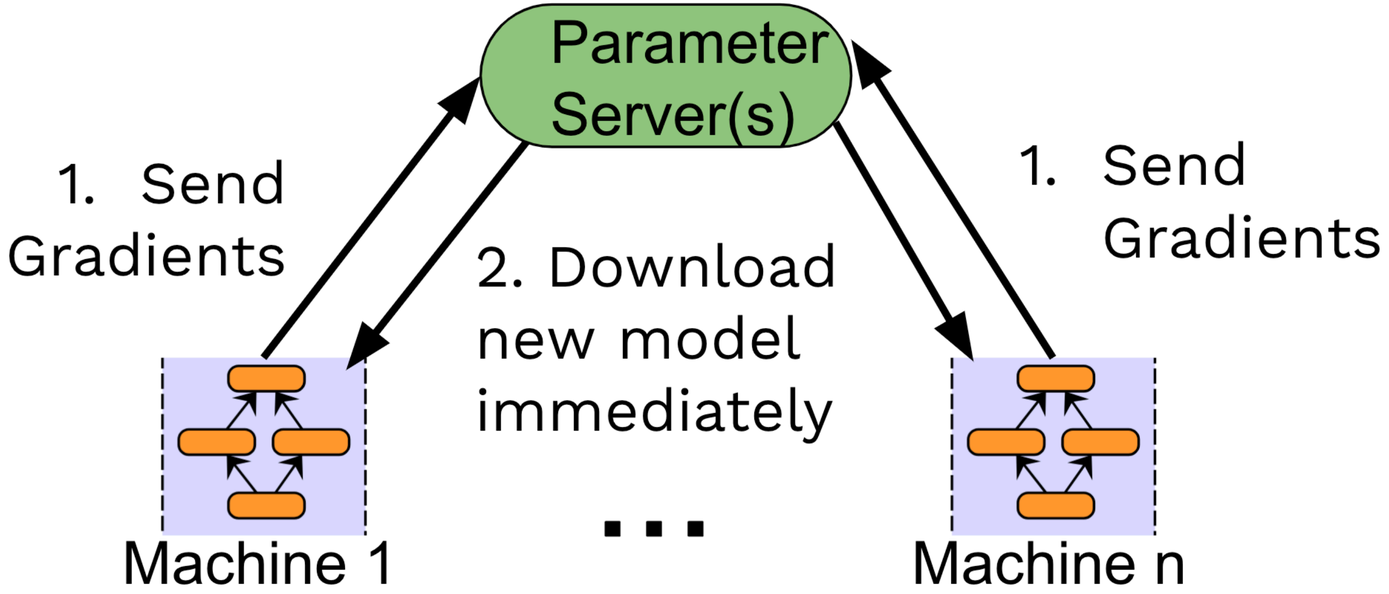




Synchronous SGD



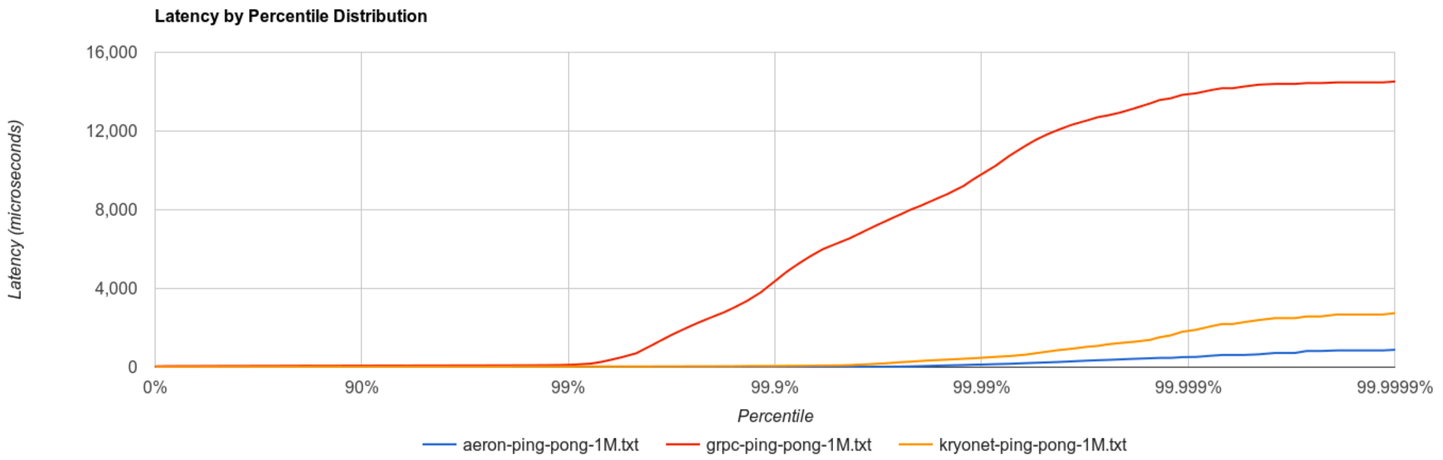
Asynchronous SGD



- 利点
  - 高いスループット (ワーカーの待機時間が減り、より計算に専念できる)
- 懸念点
  - それぞれのワーカーが別々にモデルを更新するため、無駄な重みの勾配を剪定する必要あり

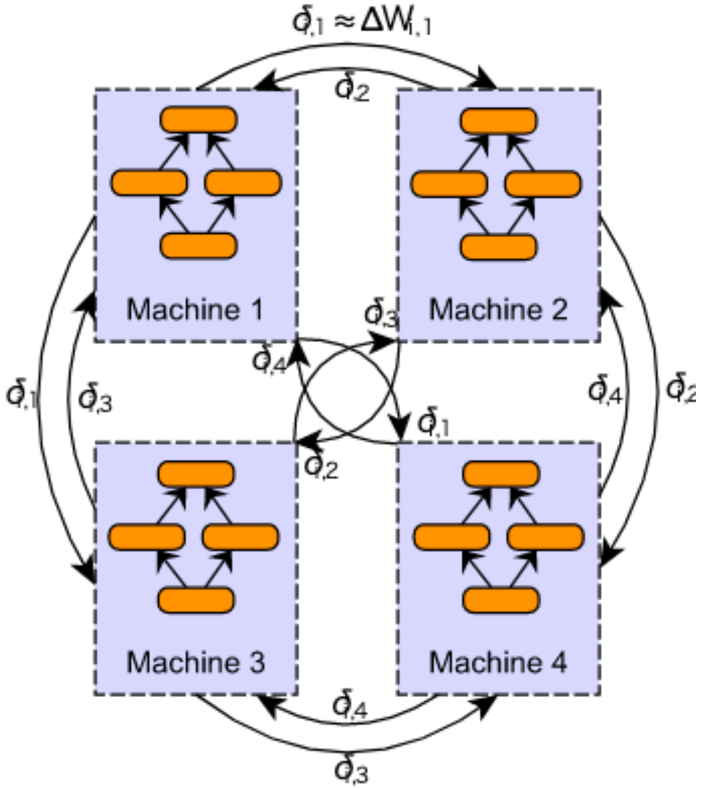
DL4Jにおける分散学習の実装

- パラメーター平均化アプローチ
  - Synchronous SGD
- 勾配ベースのアプローチ（**推奨**）
  - Asynchronous SGD with quantized gradients above a threshold
    - Spark + Aeron
    - Scalable Distributed DNN Training Using Commodity GPU Cloud Computing [Nikko Strom (Amazon), 2015]



Strom [2015] の概要

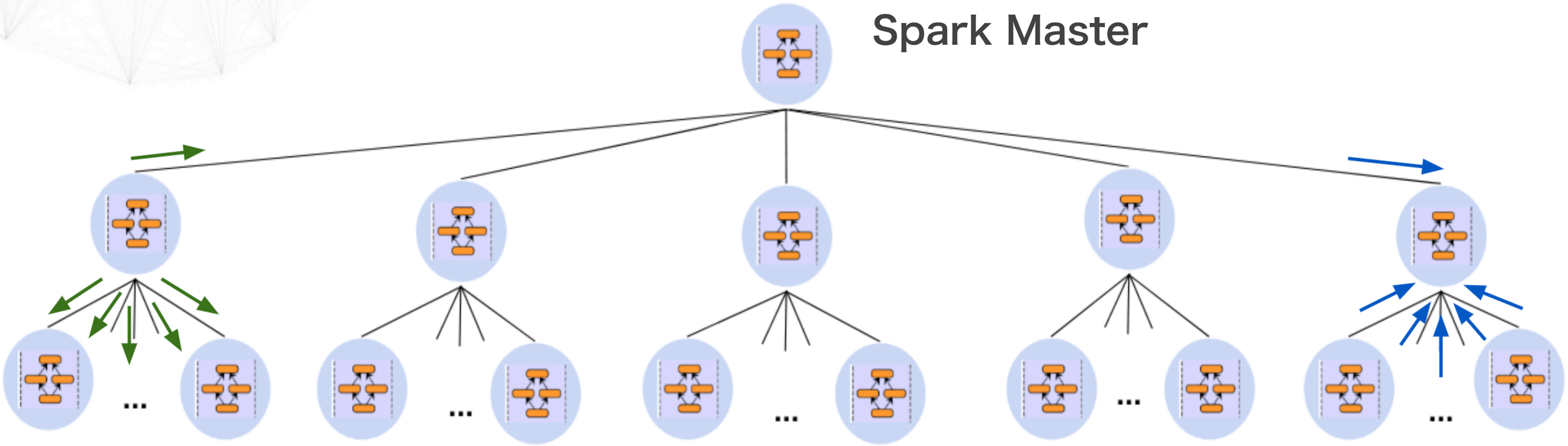
- 通信量削減のため全勾配ベクトルは送らない
  - 設定した閾値 ± τ を超えた勾配ベクトルのインデックスを送る
  - 重みの更新量も ± τ に量子化
  - ワーカー同士をP2P接続し、パラメーターサーバーなし
  - それぞれのワーカーの真の勾配と量子化した勾配の残差を蓄積
    - “勾配 + 残差” を量子化していく



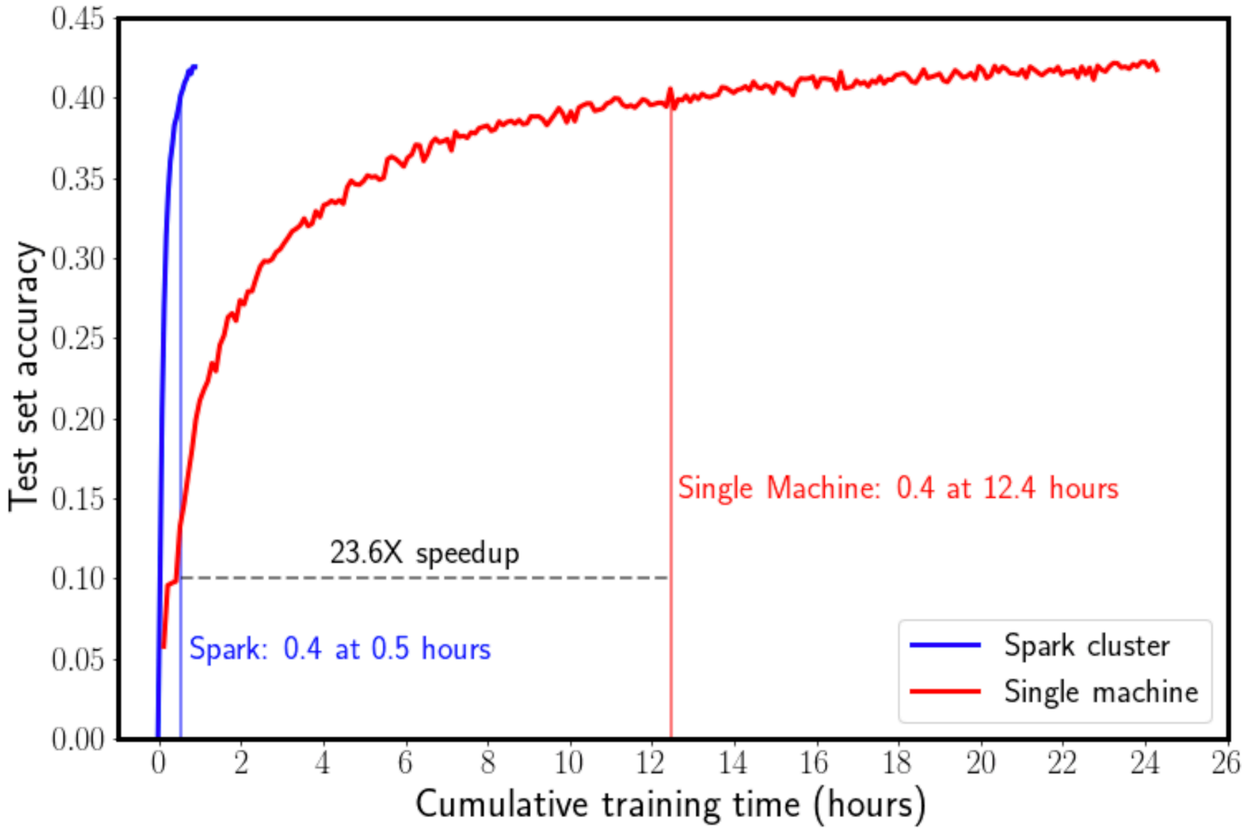
DL4JにおけるStrom論文の実装

P2P接続は実際にはツリー構造で代替している

（より帯域に優しいリング接続方式も将来のDL4Jで追加予定）



DL4JのStorm実装のベンチマークテスト結果



Linear SpeedUp

Runtime on single machine: 12.4 hours

Runtime on 24 node spark cluster: 30 minutes

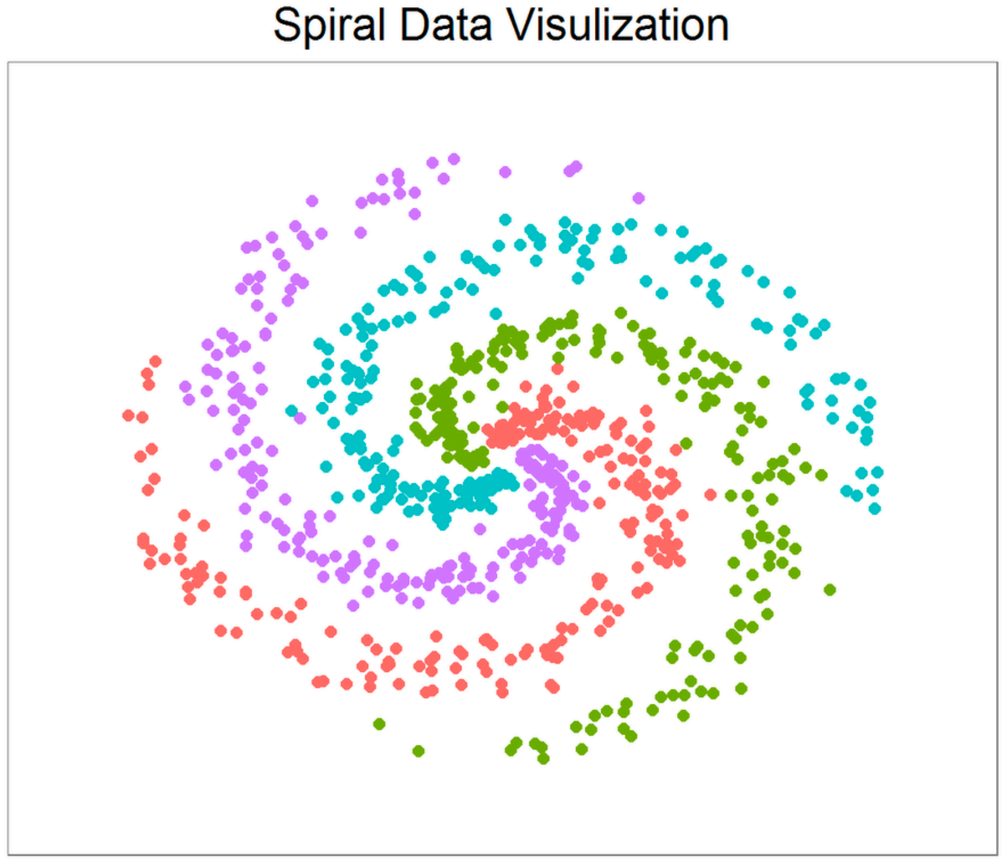
Minimal threshold tuning: 1e-4 vs. 1e-3

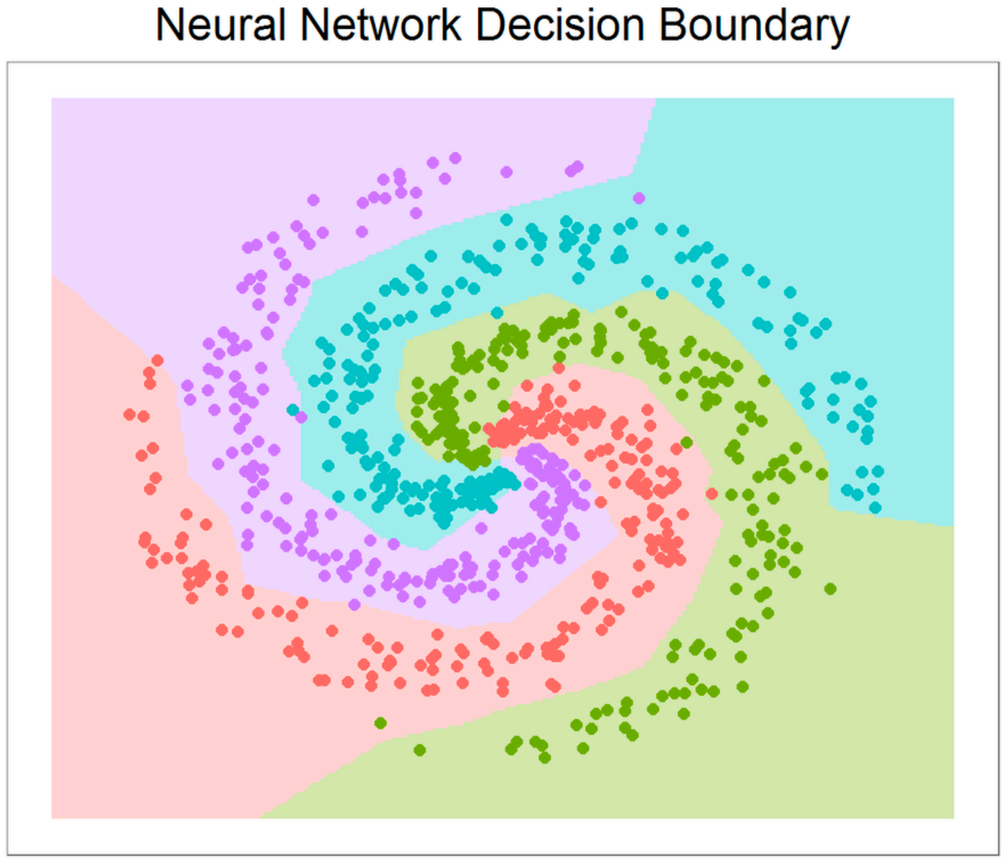
AIの性能は3つの要因で変化し続ける

Data	Model	Code
Schema	Algorithms	Business Needs
Sampling over Time	More Training	Bug Fixes
Volume	Experiments	Configuration

https://martinfowler.com/articles/cd4ml.html

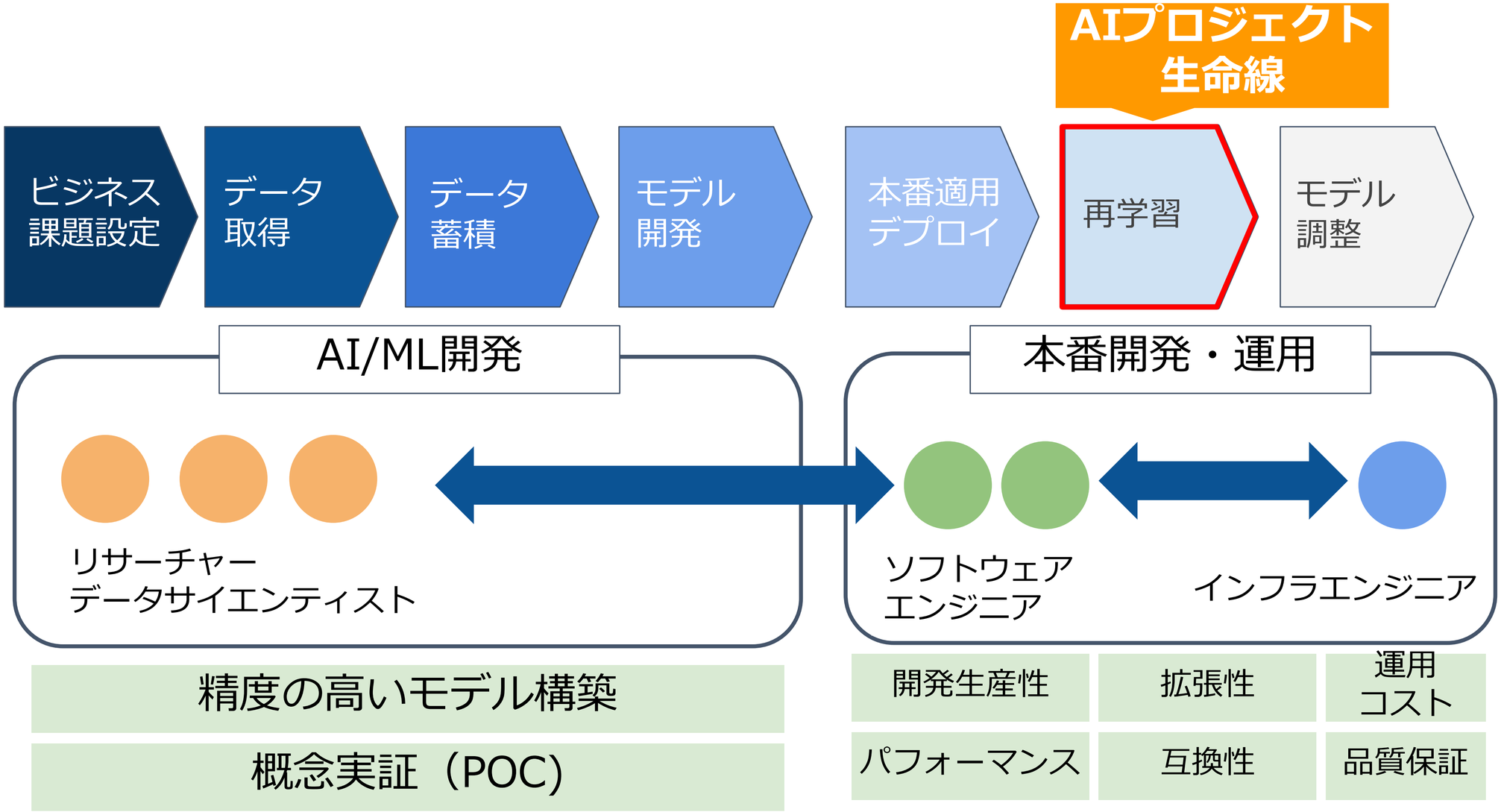
ニューラルネットワークはデータのダイナミクスを理解しない



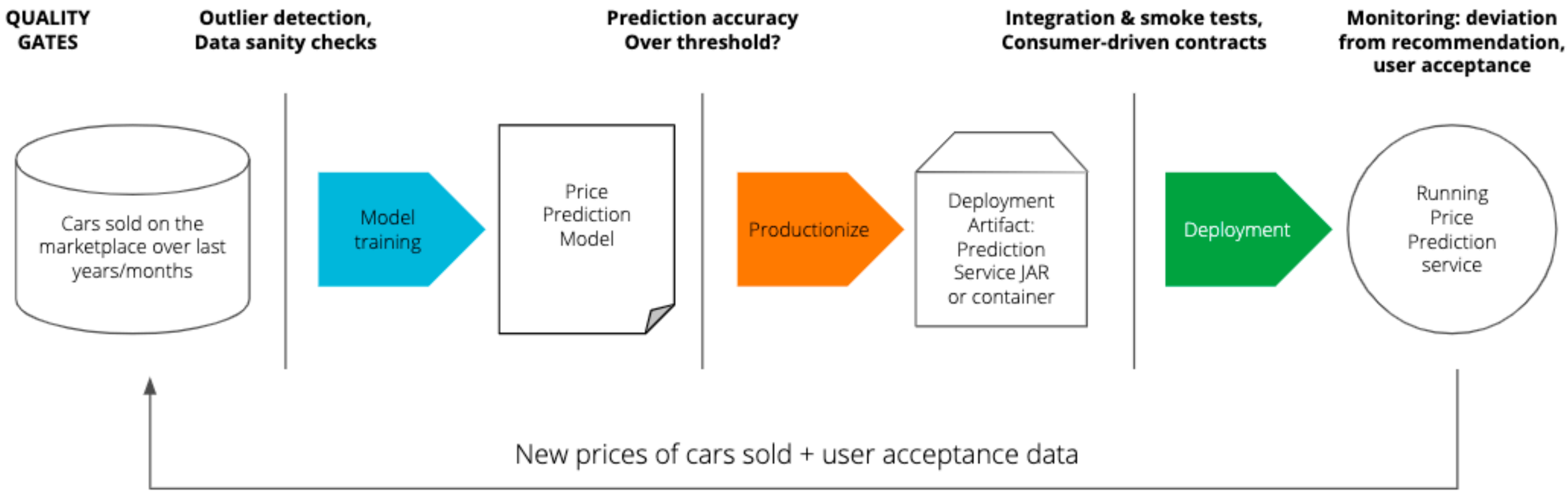


https://towardsdatascience.com/build-your-own-neural-network-classifier-in-r-b7f1f183261d

AI/MLシステム開発の流れ

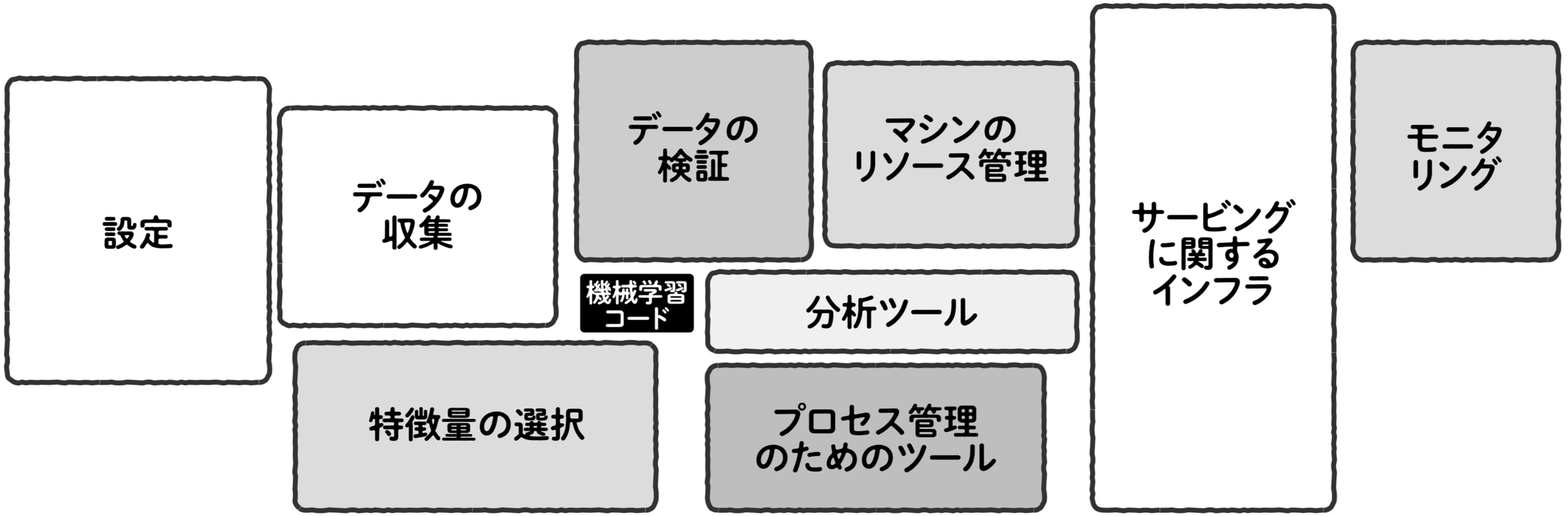


例: 中古車の適正買取価格予測



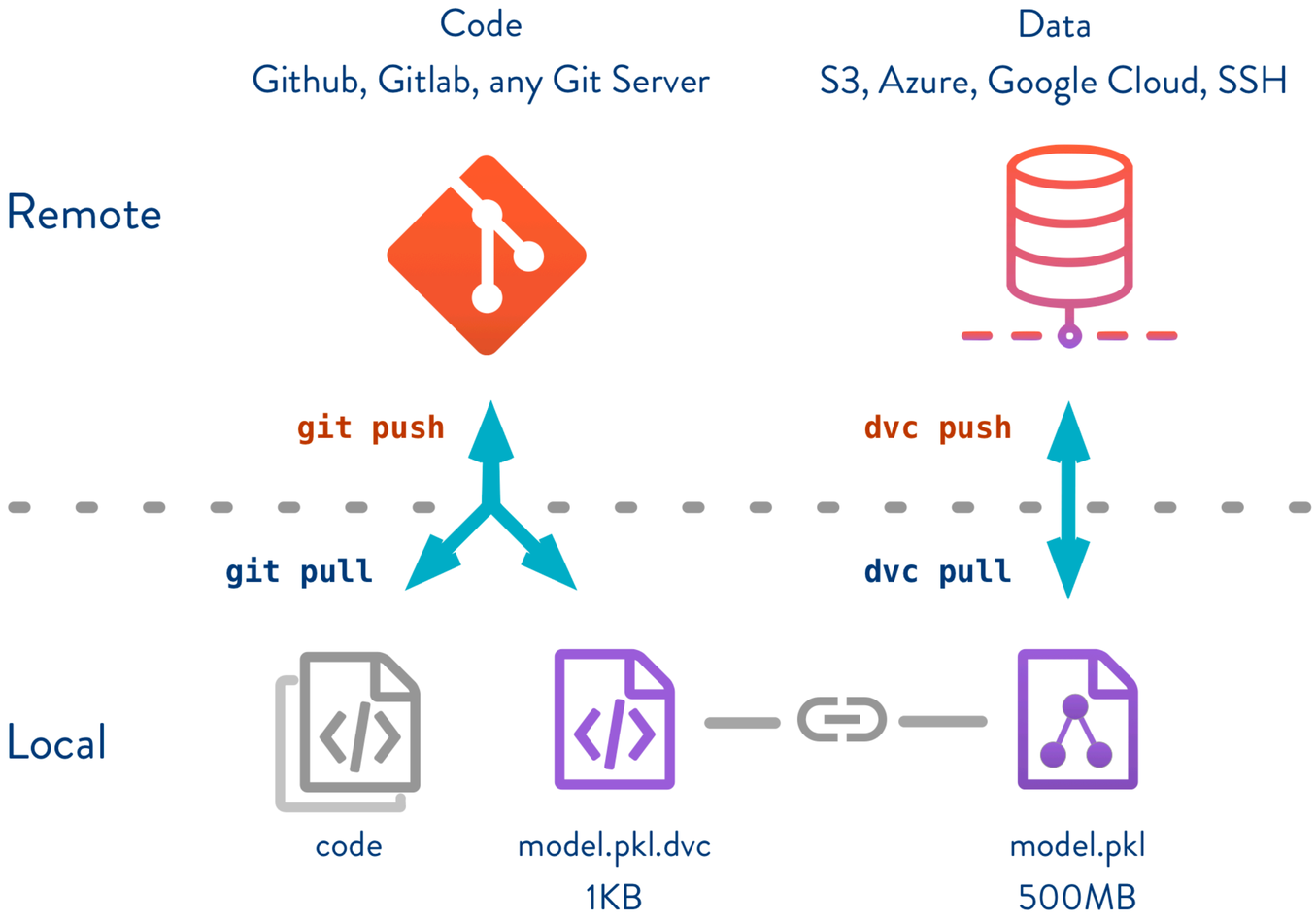
https://www.thoughtworks.com/insights/articles/intelligent-enterprise-series-cd4ml

GoogleのTechnical Debt論文の図



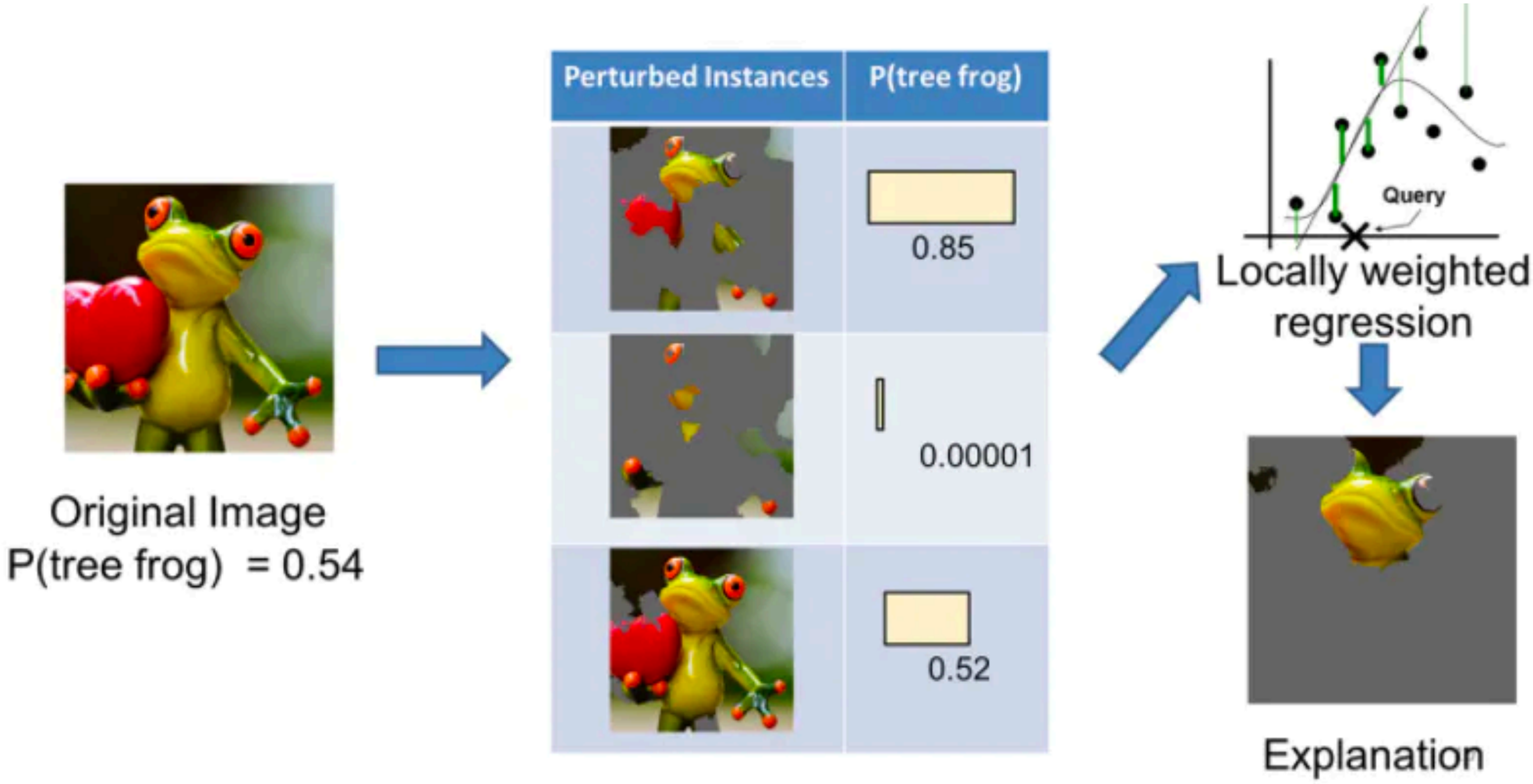
▶ 図 3.1　機械学習のモデリングのコードの割合は少ない（[2] 中の図を翻訳して引用）

DVC: Data science Version Control



https://dvc.org/doc/use-cases/data-and-model-files-versioning

LIME: Local Interpretable Model-Agnostic Explanations



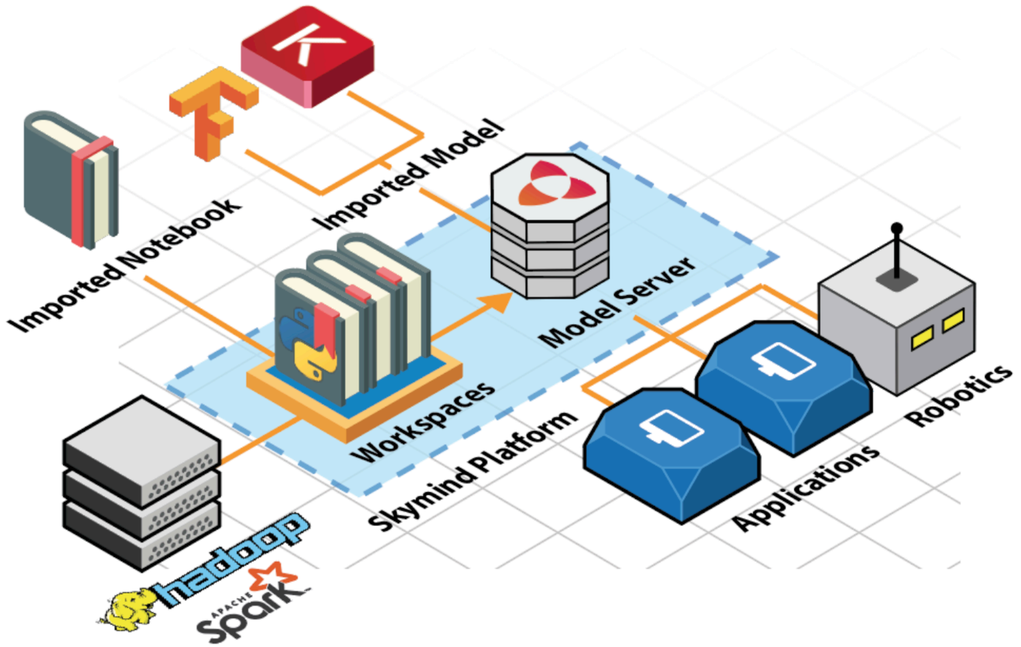
https://www.oreilly.com/learning/introduction-to-local-interpretable-model-agnostic-explanations-lime

Prometheus (モニタリング) + grafana (可視化)

https://dasalog.hatenablog.jp/entry/promehteus-grafana-qnap

SKIL: Skymind Intelligent Layer

- **Workspaces** to manage modeling experiments
- **Model server** with **REST-based API** to serve models and predictions
- **Integration** with multiple compute platforms (Spark, cloud services, etc.)



https://skymind.ai/platform

まとめ

- 「詳説 Deep Learning ー 実務者のためのアプローチ」発売中です
- Deeplearning4j (DL4J) は Keras-like な API
  - DataVec で前処理
  - Spark で分散処理
  - TensorFlow, Keras などPython系で学習したモデルをDL4Jにインポート可能
- モデル学習して終わりというわけでもない
  - CI/CD的発想が必要
  - 周辺ツールがオープンソースで出揃ってきている
  - パッケージングされたソフトウェア/SaaSを使うのもあり

Appendix

skymind

DL4J Ecosystem

Deeplearning4j, ScalNet
Build, train, and deploy neural networks on JVM and in Spark.

ND4J / libND4J
High performance linear algebra on GPU/CPU. Numpy for JVM.

DataVec
Data ingestion, normalization, and vectorization. Pandas integration.

SameDiff
Symbolic differentiation and computation graphs.

Arbiter
Hyperparameter search for optimizing neural networks.

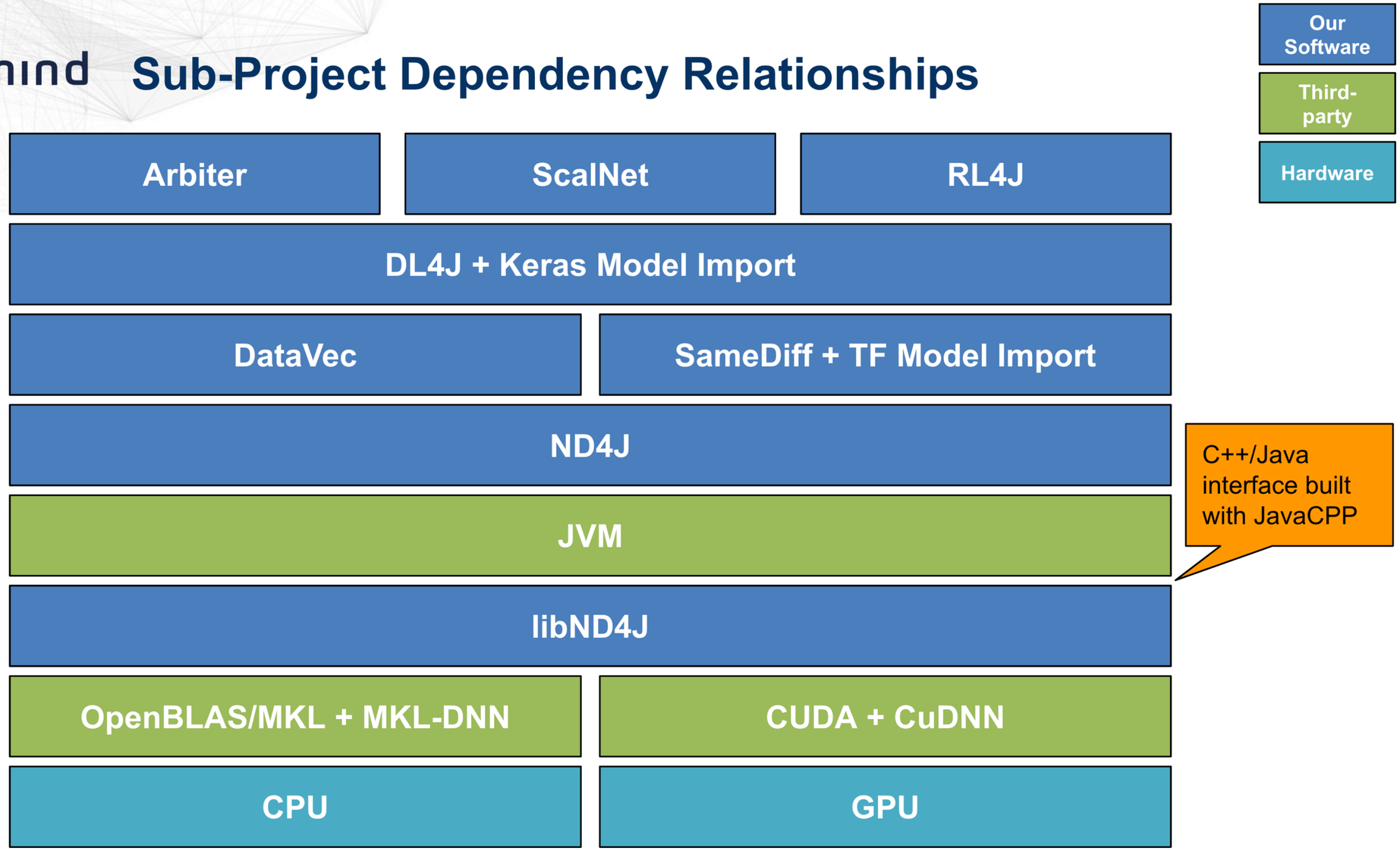
RL4J
Reinforcement learning on JVM.

Model Import
Import neural nets from ONNX, TensorFlow, Keras (Theano, Caffe).

Jumpy
Python API for ND4J.

skymind

Sub-Project Dependency Relationships



skymind

Common Use Cases

Anomaly Detection	Fraud, Cybersecurity, Infrastructure
Computer Vision	Image Classification, Detection, Segmentation, OCR, Medical
Natural Language Processing	Sentiment Analysis, Translation, Chatbots
Predictive Analytics	Forecasting, Churn Prediction
Recommender Systems	Marketing, Content Recommendation

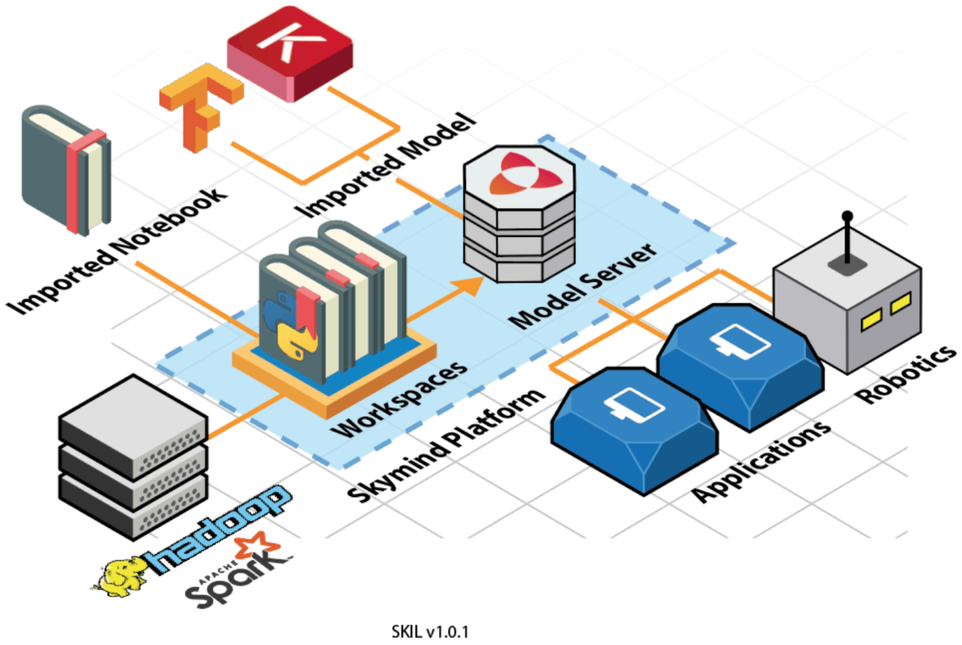
skymind

Skymind Intelligence Layer

SKIL bridges the gap between the Python ecosystem and the JVM with a cross-team platform for Data Scientists, Data Engineers, and DevOps/IT.

Key Features

- Model Import via TF and Keras
- Integrations with JVM stack
- Platform Agnostic
- Deployment On-Prem, Cloud, or Hybrid
- Distributed Across Multi-GPU
- Real-time or Batch




H2O.ai
DataRobot
skymind
aws
Google Cloud Platform
ANACONDA
Powered by Continuum Analytics
DOMINO
DATA LAB
hadoop
APACHE
Spark

skymind

Who is SKIL for?

Data Scientist

Wants: Flexibility for experimentation and data viz.
Concern: Model accuracy.

Enterprise Developers

Wants: Integrate systems using standard technologies.
Concern: Bugs in interfacing code. Performance for complex data.

DevOps

Wants: Systems built with familiar technologies.
Concern: SLA monitoring and management.

C-Level

Wants: Reduce time to market.
Concern: Costs and time.

skymind

DataVec Transform Process

Basic Transform Example found here

- Filter rows by column value
- Handle invalid values with replacement (-ve $ amt)
- Handle datetime, extract hour of day etc
- Operate on columns in place
- Derive new columns from existing columns
- Join multiple sources of data
- AND much more...

Serialize to JSON!!

https://gist.github.com/eraly/3b15d35eb4285acd444f2f18976dd226

Import into DL4J

Importing the model and weights files separately:

```
MultiLayerNetwork network =
    KerasModelImport.importKerasSequentialModelAndWeights(
          "<JSON_FILE>",
          "<H5_FILE>")
```

OR

```
ComputationGraph network =
    KerasModelImport.importKerasModelAndWeights("<JSON_FILE>","<H5_FILE>")
```

Import into DL4J

Importing only the Model Architecture JSON File

```
MultiLayerNetworkConfiguration modelConfig =
    KerasModelImport.importKerasSequentialConfiguration("<JSON_FILE>”)
```

OR

```
ComputationGraphConfiguration modelConfig =
    KerasModelImport.importModelConfiguration("<JSON_FILE>”)
```

skymind

Core Layers

- Dense
- Activation
- Dropout
- Flatten
- Reshape
- Merge
- Permute
- RepeatVector
- Lambda
- ~~ActivityRegularization~~
- Masking
- SpatialDropout1D
- SpatialDropout2D
- SpatialDropout3D

Convolutional Layers

- Conv1D
- Conv2D
- Conv3D
- AtrousConvolution1D
- AtrousConvolution2D
- ~~SeparableConv1D~~
- SeparableConv2D
- Conv2DTranspose
- ~~Conv3DTranspose~~
- Cropping1D
- Cropping2D
- Cropping3D
- UpSampling1D
- UpSampling2D
- UpSampling3D
- ZeroPadding1D
- ZeroPadding2D
- ZeroPadding3D

skymind

Pooling Layers

- MaxPooling1D
- MaxPooling2D
- MaxPooling3D
- AveragePooling1D
- AveragePooling2D
- AveragePooling3D
- GlobalMaxPooling1D
- GlobalMaxPooling2D
- GlobalMaxPooling3D
- GlobalAveragePooling1D
- GlobalAveragePooling2D
- GlobalAveragePooling3D

Embedded, Locally Connected

- LocallyConnected1D
- LocallyConnected2D
- Embedding

Recurrent Layers

- SimpleRNN
- ~~GRU~~
- LSTM
- ~~ConvLSTM2D~~

Merge, Advanced Activations

Merge Layers

- Add / add
- Multiply / multiply
- Subtract / subtract
- Average / average
- Maximum / maximum
- Concatenate / concatenate
- Dot / dot

Advanced Activation Layers

- LeakyReLU
- PReLU
- ELU
- ThresholdedReLU

Normalization, Noise, Layer Wrappers

Normalization Layers

- BatchNormalization

Noise Layers

- GaussianNoise
- GaussianDropout
- AlphaDropout

Layer Wrappers

- ~~TimeDistributed~~
- Bidirectional

skymind

Keras Applications

Keras model import now imports ***every*** Keras application

https://keras.io/applications/

- Xception
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet
- DenseNet
- NASNet
- MobileNetV2

End to End tests:

https://github.com/deeplearning4j/deeplearning4j/blob/master/deeplearning4j/deeplearning4j-modelimport/src/test/java/org/deeplearning4j/nn/modelimport/keras/e2e/KerasModelEndToEndTest.java