JULIA Programming

ISLab

Synthetic Intelligence Lab
@UMinho

Marco Gomes

Summary

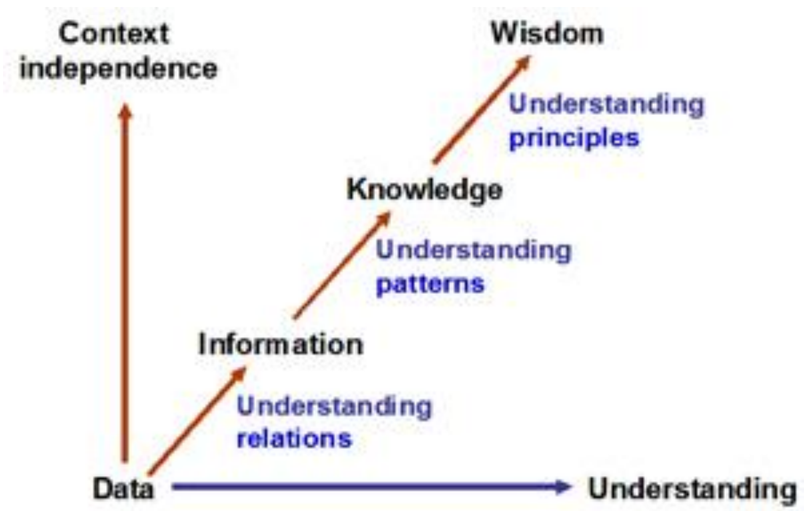
- **Introduction to Julia**

- **Artificial Neural Networks**
  - **Libraries**
  - **An example**
  - **XOR problem**

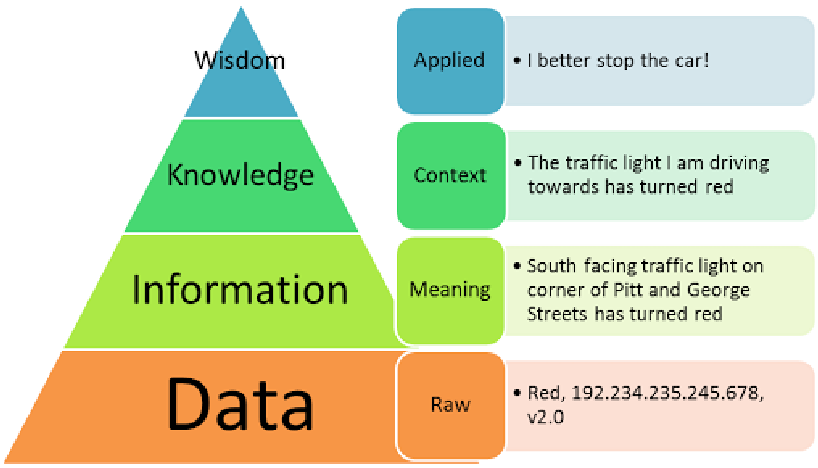
- **Deep Learning in Julia**

Out of scope

Out of scope



Context
independence
Wisdom
Understanding
principles
Knowledge
Understanding
patterns
Information
Understanding
relations
Data
Understanding



Wisdom
Knowledge
Information
Data
Applied
• I better stop the car!
Context
• The traffic light I am driving towards has turned red
Meaning
• South facing traffic light on corner of Pitt and George Streets has turned red
Raw
• Red, 192.234.235.245.678, v2.0

Introduction to Julia programming

Introduction: hi, Julia!

- First appeared: ~2012
- Paradigm: Multi-paradigm
- Website: http://julialang.org/
- Documentation: https://docs.julialang.org/en/v1/
- Wiki: https://en.wikipedia.org/wiki/Julia_(programming_language)

Introduction: hi, Julia!

«Who's» Julia?

A new programming language for *scientific computing, machine learning, data mining, large-scale linear algebra, distributed and parallel computing*:

- Developed by a group mostly from MIT (~2009)
- Fully open source
- It is a high-level and dynamic programming language

Community *motto*: «Walk like Phyton, run like C»!

The focus: learn how to use Julia for data analysis!

Introduction: hi, Julia!

2-language problem

Typical solutions involve using two languages:

- Phyton, matlab, mathematica, etc. for investigatory work and prototype development
- C, Fortan, OpenCL, specialized solutions (or even just recording in Cython) for performance implementations (to scale-up our code)

This is a huge development overhead

Introduction: hi, Julia!

Here are some of the ways Julia implements those aspirations (1/2):

- **Compiled, not interpreted, for speed**. Julia is just-in-time (JIT) compiled using the LLVM compiler framework. At its best, Julia can approach or match the speed of C.
- **Straightforward but useful syntax.** Julia's syntax is similar to Python's—terse, but also expressive and powerful.
- **Python, C, and Fortran libraries are just a call away.** Julia can interface directly with external libraries written in C and Fortran. It's also possible to interface with Python code by way of the PyCall library, and even share data between Python and Julia.

Introduction: hi, Julia!

Here are some of the ways Julia implements those aspirations (2/2):

- **Dynamic typing with static type benefits.** You can specify types for variables, like "unsigned 32-bit integer." But you can also create hierarchies of types to allow general cases for handling variables of specific types—for instance, to write a function that accepts integers generally without specifying the length or signing of the integer. And, finally, you can do without typing entirely if it isn't needed in a particular context.

- **Metaprogramming.** Julia programs can generate other Julia programs, and even modify their own code, in a way that is reminiscent of languages like Lisp.

Introduction: hi, Julia!

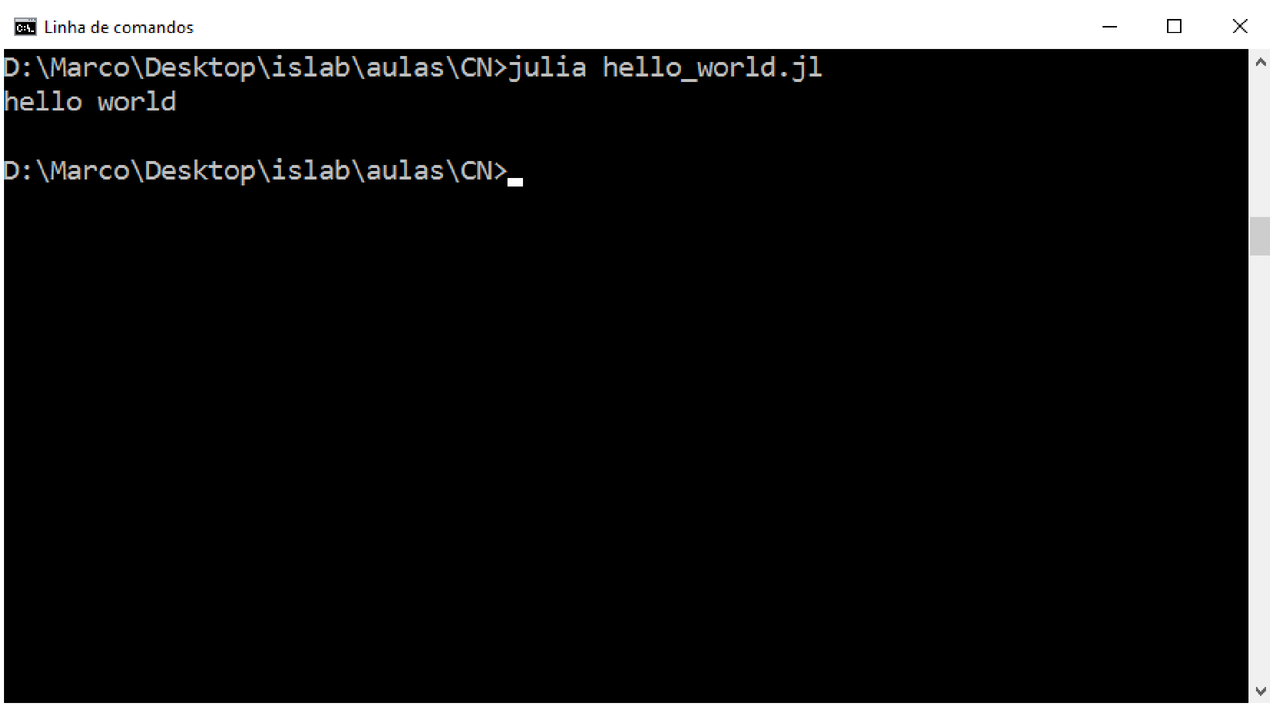
Interaction with Julia:

- Command-line invocation
- REPL (Read–Eval–Print Loop)
- Jupyter (IPhyton like notebooks)
- Juno + Atom

Introduction: hi, Julia!

Interaction with Julia:

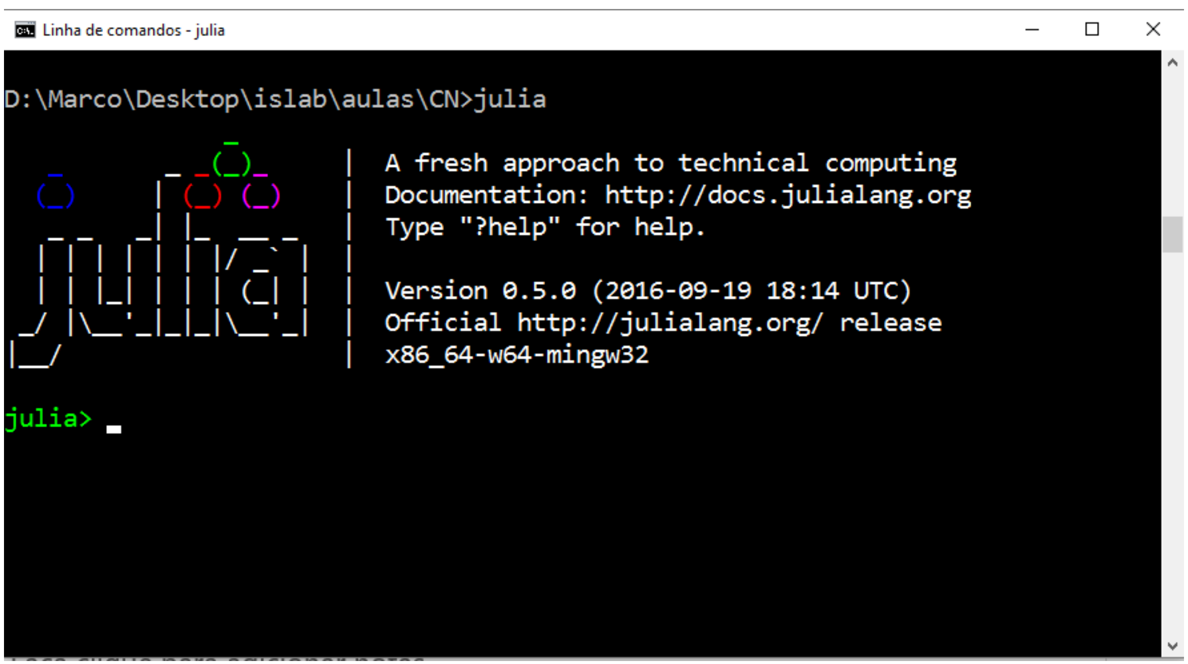
- Command-line invocation



Introduction: hi, Julia!

Interaction with Julia:

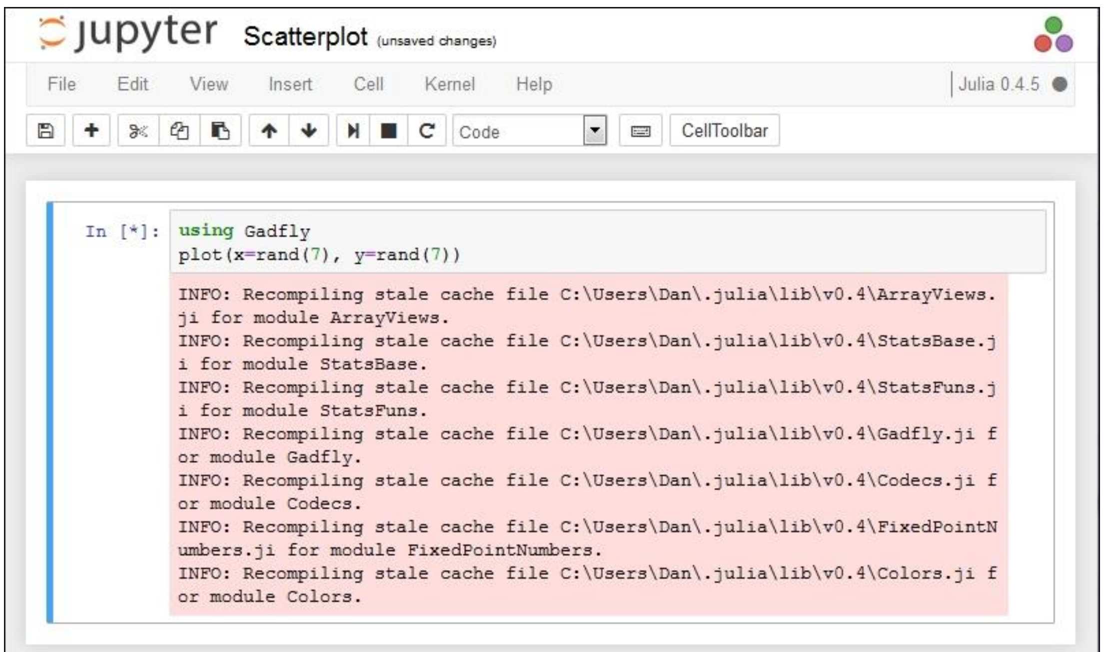
- REPL (Read–Eval–Print Loop)



Introduction: hi, Julia!

Interaction with Julia:

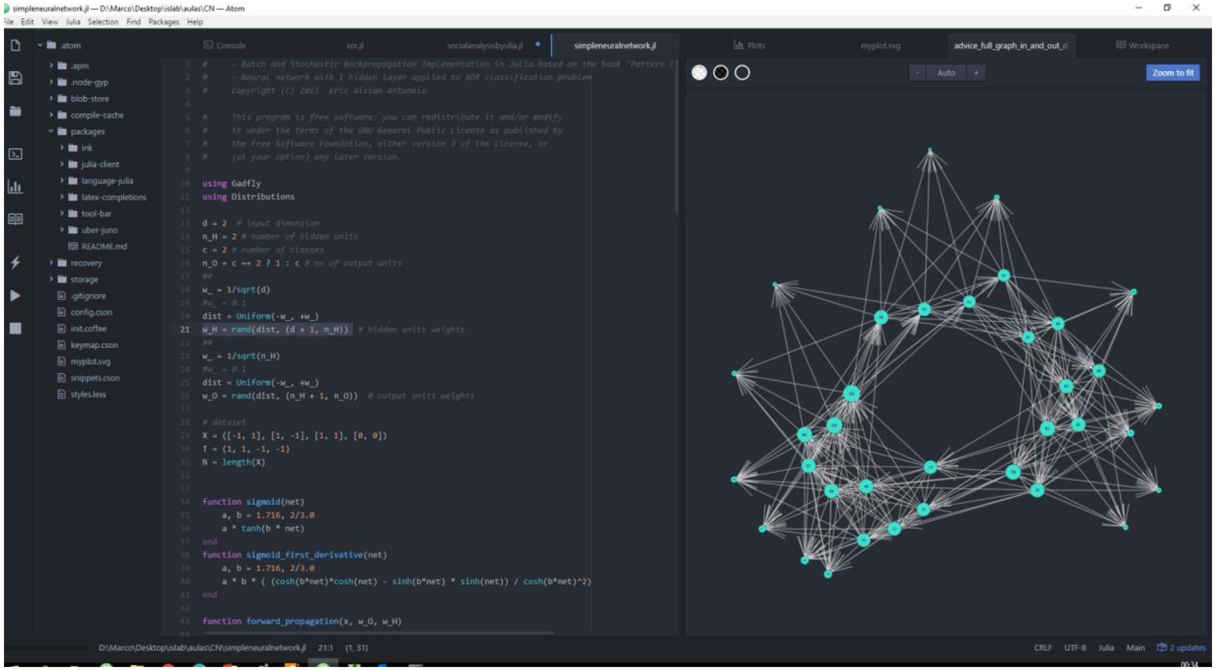
- Jupyter (IPhyton like notebooks)



Introduction: hi, Julia!

Interaction with Julia:

- Juno IDE (built of Atom)



Introduction: hi, Julia!

JuliaBox

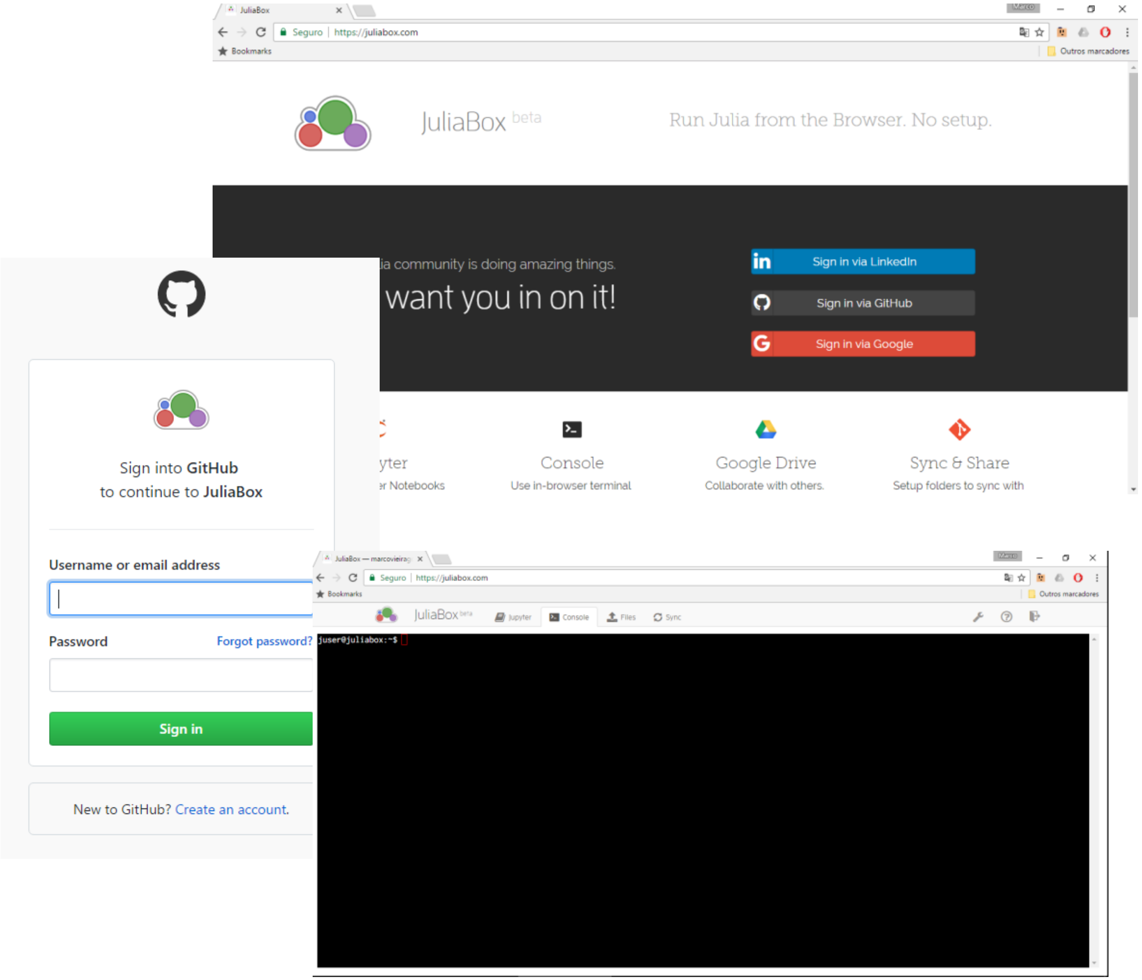
- You don't even need to install Julia! JuliaBox* is an online server that allows you to run your Julia codes on a remote machine hosted by the Amazon WebServices.
- Once you are in, there will be an iJulia notebook interface running on Jupiter (iPython). You can easily upload, download and edit your codes. Moreover, you can sync your files with Google Drive or GIT.

* JuliaBox: http://juliabox.org

Introduction: hi, Julia!

JuliaBox

Using a github account



Julia & IJulia Cheat-sheet (for 18.xxx at MIT)

Basics:

`julialang.org` — documentation; `juliabox.com` — run Julia online
`github.com/stevengj/julia-mit` installation & tutorial
`using IJulia; IJulia.notebook()` start IJulia browser
shift-return execute input cell in IJulia

Defining/changing variables:

`x = 3` define variable x to be 3
`x = [1,2,3]` array/"column"-vector (1,2,3)
`y = [1 2 3]` 1×3 matrix (1,2,3)
`A = [1 2 3 4; 5 6 7 8; 9 10 11 12]`
—set A to 3×4 matrix with rows 1,2,3,4 etc.
`x[2] = 7` change x from (1,2,3) to (1,7,3)
`A[2,1] = 0` change $A_{2,1}$ from 5 to 0
`u, v = (15.03, 1.2e-27)` set u=15.03, v=1.2×10^{-27}
`f(x) = 3x` define a function f(**x**)
`x -> 3x` an "anonymous" function

Constructing a few simple matrices:

`rand(12), rand(12,4)` random length-12 vector or 12×4 matrix with uniform random numbers in [0,1)
`randn(12)` Gaussian random numbers (mean 0, std. dev. 1)
`eye(5)` 5×5 identity matrix I
`linspace(1.2,4.7,100)` 100 equally spaced points from 1.2 to 4.7
`diagm(x)` matrix whose diagonal is the entries of x

Portions of matrices and vectors:

`x[2:12]` the 2nd to 12th elements of x
`x[2:end]` the 2nd to the last elements of x
`A[5,1:3]` row vector of 1st 3 elements in 5th row of A
`A[5,:]` row vector of 5th row of A
`diag(A)` vector of diagonals of A

Arithmetic and functions of numbers:

`3*4, 7+4, 2-6, 8/3` mult., add, sub., divide numbers
`3^7, 3^(8+2im)` compute 3^7 or 3^{8+2i} power
`sqrt(-5+0im)` $\sqrt{-5}$ as a complex number
`exp(12)` e^{12}
`log(3), log10(100)` natural log (ln), base-10 log ($\log_{10}$)
`abs(-5), abs(2+3im)` absolute value $|-5|$ or $|2+3i|$
`sin(5pi/3)` compute $\sin(5\pi/3)$
`besselj(2,6)` compute Bessel function $J_2(6)$

Arithmetic and functions of vectors and matrices:

`x * 3, x + 3` multiply/add every element of x by 3
`x + y` element-wise addition of two vectors x and y
`A*y, A*B` product of matrix A and vector y or matrix B
`x * y` not defined for two vectors!
`x .* y` element-wise product of vectors x and y
`x .^ 3` every element of x is cubed
`cos.(x), cos.(A)` cosine of every element of x or A
`exp.(A), expm(A)` exp of each element of A, matrix exp e^A
`x', A'` conjugate-transpose of vector or matrix
`x'*y, dot(x,y), sum(conj(x).*y)` three ways to compute $x \cdot y$
`A \ b, inv(A)` return solution to Ax=b, or the matrix A^{-1}
`λ, V = eig(A)` eigenvals λ and eigenvectors (columns of V) of A

Plotting (type `using PyPlot` first)

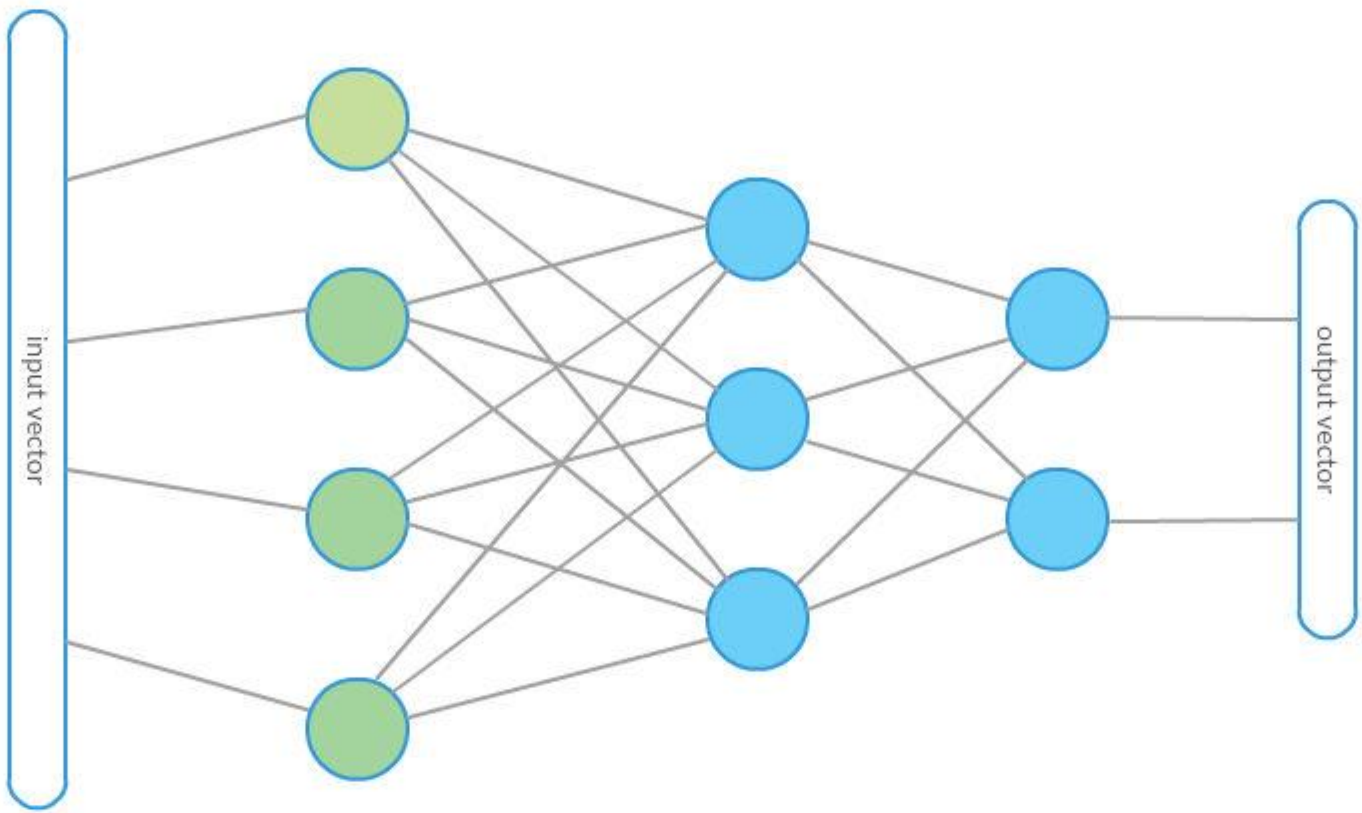
`plot(y), plot(x,y)` plot y vs. 0,1,2,3,… or versus x
`loglog(x,y), semilogx(x,y), semilogy(x,y)` log-scale plots
`title("A title"), xlabel("x-axis"), ylabel("foo")` set labels
`legend(["curve 1", "curve 2"], "northwest")` legend at upper-left
`grid(), axis("equal")` add grid lines, use equal x and y scaling
`title(L"the curve $e^\sqrt{x}$")` title with LaTeX equation
`savefig("fig.png"), savefig("fig.pdf")` save PNG or PDF image

Introduction: hi, Julia!

Hands on!

Artificial Neural Networks

ANN



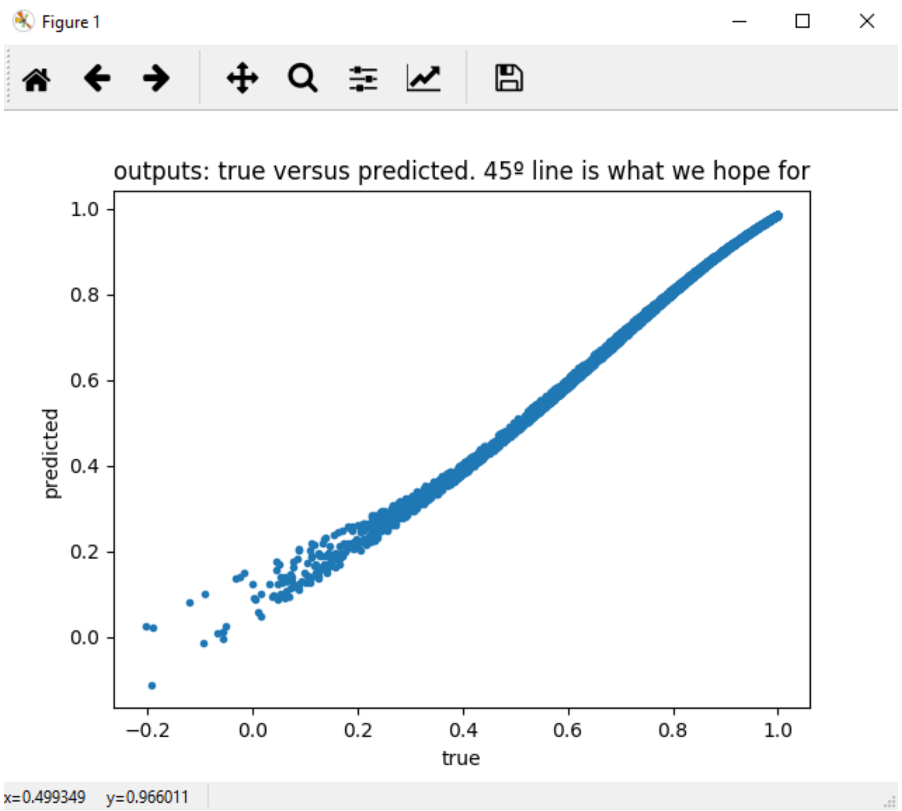
ANN

Julia libraries for Neural Networks:

- Flux.jl
- Mocha.jl
- MXNet.jl
- TensorFlow.jl
- Knet.jl
- … (https://github.com/svaksha/Julia.jl/blob/master/AI.md#neural-networks)

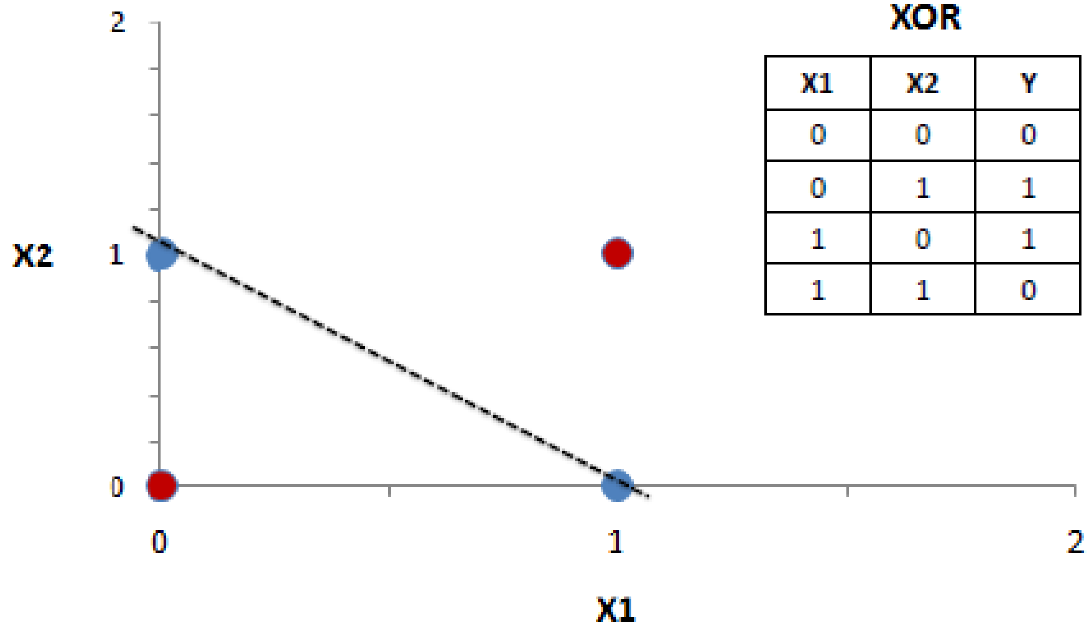
ANN

Simple regression example using MXNet.jl



ANN

Exercise 1: training a ANN to «solve» XOR problem



XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Deep Learning in julia

Deep Learning in Julia

The big picture

Tribe	Origins	Master Algorithm
Symbolists	Logic, philosophy	Inverse deduction
Connectionists	Neuroscience	Backpropagation
Evolutionaries	Evolutionary biology	Genetic programming
Bayesians	Statistics	Probabilistic inference
Analogizers	Psychology	Kernel machines

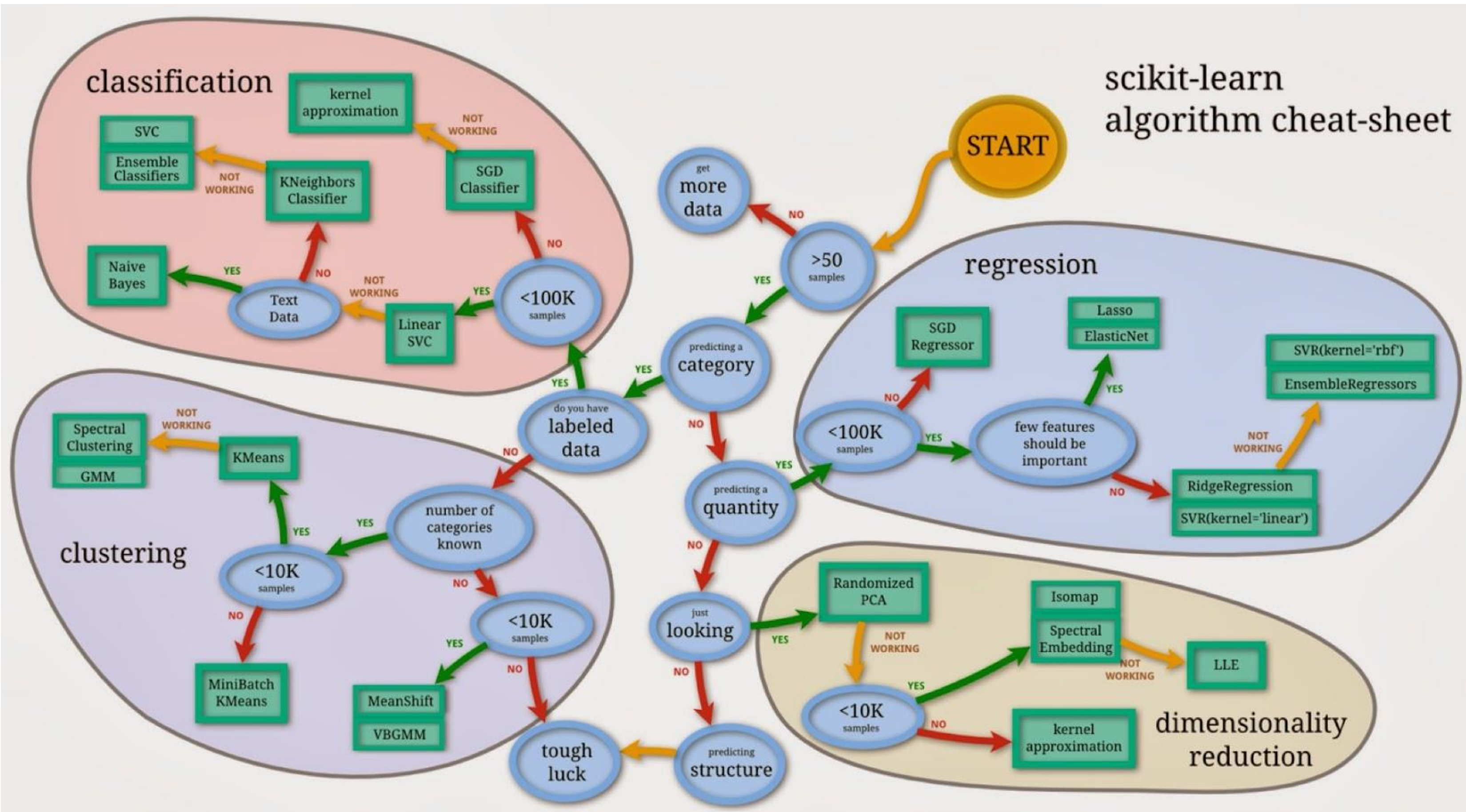
Symbolists: problem of learning knowledge that can be compose in many different ways (filling the gaps of knowledge)
Connectionists: problem of (blame) credit assignment using back propagation
Evolutionaries: the problem of learning structure
Bayesians: the problem of uncertainty
Analogizers: learn by analogy, it can generalize using just one or two examples

Deep Learning in Julia

Putting the pieces together (what all tribes shares):

- **Representation (which mathematic function the learner represents what is learning e.g. first order logic, linear regression, differential equations)**
    1. Probabilistic logic (e.g. Markov logic networks)
    2. Weighted formulas -> Distribution over states
- **Evaluation (how well my candidate fits the data) should be provide by the user**
    1. Posterior probability
    2. User-defined objective function
- **Optimization (find the model that maximize the objective function)**
    1. Formula discovery: genetic programming
    2. Weight learning: backpropagation

Deep Learning in Julia



Deep Learning in Julia

What is Deep Learning then?

- Depending on the quality of the features, the learning problem might become easy or difficult.
- What features to look at when the input are complicated or unintuitive?
  - E.g. for image input, looking at the raw pixels directly is usually not very helpful
- Feature designing / engineering used to be a very important part of machine learning applications.
  - SIFT in computer vision
  - MFCC in speech recognition
- Deep Learning: **learning both the representations and the model parameters automatically and jointly from the data**.
  - Recently become possible with huge amount of data (credit: internet, mobile devices, Mechanic Turk, …) and highly efficient computing devices (GPUs, ...)

Deep Learning in Julia

Several facts

- Many machine learning and deep learning algorithms fits nicely with GPU parallelization models: simple logic but massive parallel computation.
- Training time large deep neural networks:
  - From ∞ (or probably finite, but takes years, nobody was able to do it in pre-GPU age)
  - To weeks or even days, with optimally designed models, computation kernels, IO, and multi-GPU parallizations

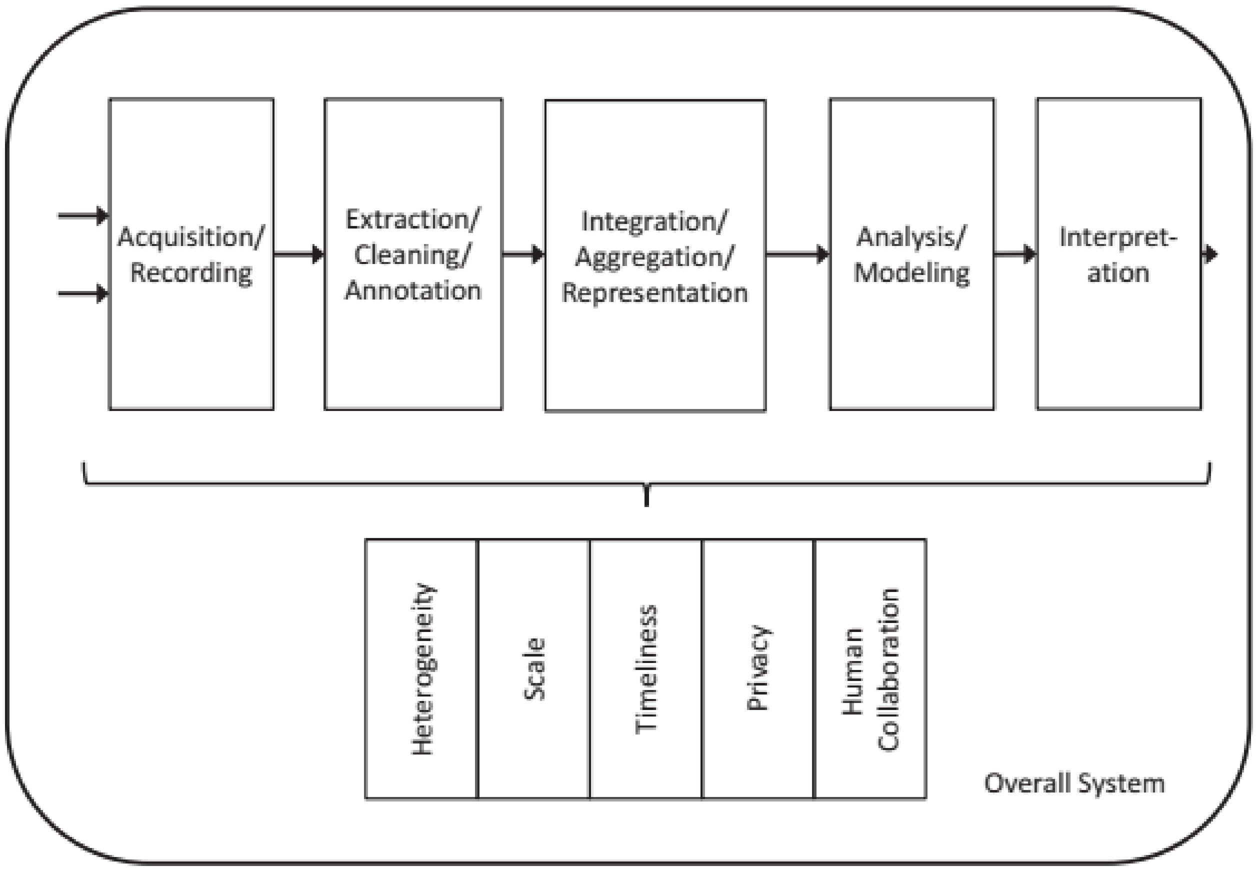
- Julia is primarily designed for CPU parallelization and distributed computing, but GPU computing in Julia is gradually getting there
  - https://github.com/JuliaGPU

Deep Learning in Julia

Now there are several packages available in Julia with GPU supports:

- **Flux.jl:** https://github.com/FluxML/Flux.jl
  It is an elegant approach to machine learning. It's a 100% pure-Julia stack.
- **Mocha.jl**: https://github.com/pluskid/Mocha.jl Currently the most feature complete one. Design and architecture borrowed from the Caffe deep learning library.
- **MXNet.jl:** https://github.com/dmlc/MXNet.jl
  A successor of Mocha.jl. Different design, with a language-agnostic C++ backend dmlc/libmxnet. Relatively new but very promising, with flexible symbolic API and efficient multi-GPU training support.
- **Knet.jl:** https://github.com/denizyuret/Knet.jl
  Experimental symbolic neural network building script compilation.

Deep Learning in Julia



Deep Learning in Julia

Hands on: a simple example using Flux.jl library

- Run the examples from here: https://github.com/pluskid/Mocha.jl/tree/master/examples
- Adapt your code using different learning approaches

Deep Learning in Julia

Exercise 2:

Use the RDatasets packages to train and test Deep Neural Networks

JULIA Programming

ISLab

Synthetic Intelligence Lab
@UMinho

Marco Gomes