Julia: Next Generation Language

SciPy 2019, Universidad de los Andes


There are dozens of us!

Let us understand the necessity: Scientific AI

Machine Learning + Domain Power modeling

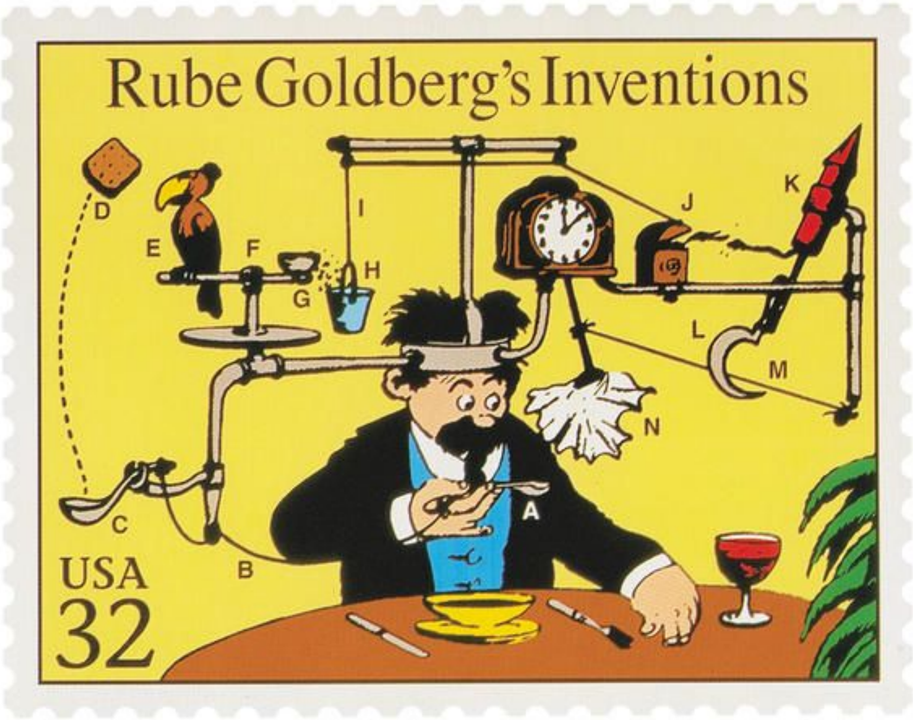
Do I have to learn another language?

Current situation: I'm here talking to experts in the scientific python community

At the moment we have the two language problem in Scientific AI

When I was TA in tools for computing:

- **NumPy** for numerical linear algebra.
- **R** for statistical analysis.
- **C** for the fast stuff
- **Bash** to tie all up



A.k.a "Ousterholt's Dichotomy"

"System languages"

- Static
- Compiled
- User types

"Scripting languages"

- Dynamic
- Interpreted
- Standard types

The Two Language Problem ?

Because of this dichotomy, a two-tier compromise is standard:

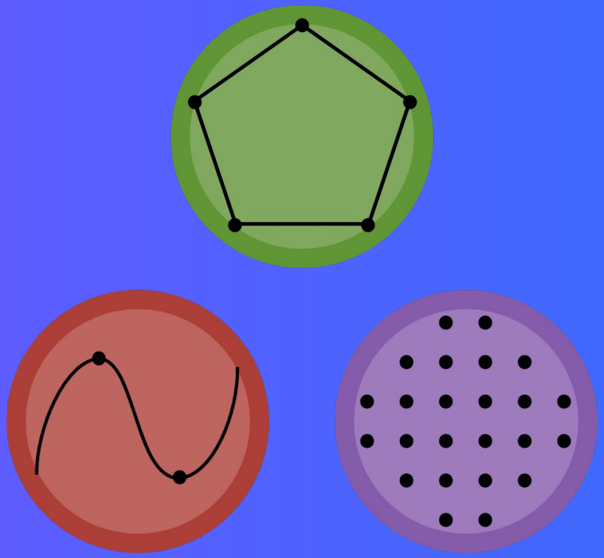
- For **convenience**, use a a scripting language (Matlab, R, Python)
- But do all the **hard stuff** in a system language (C, C++, Fortran)

Pragmatic for many applications, but has drawbacks

- Aren't the **hard parts** exactly where you need an **easier** language?
- Forces **vectorization** everywhere, even when awkward or wasterful
- Creates a **social barrier** - a wall between users and developers

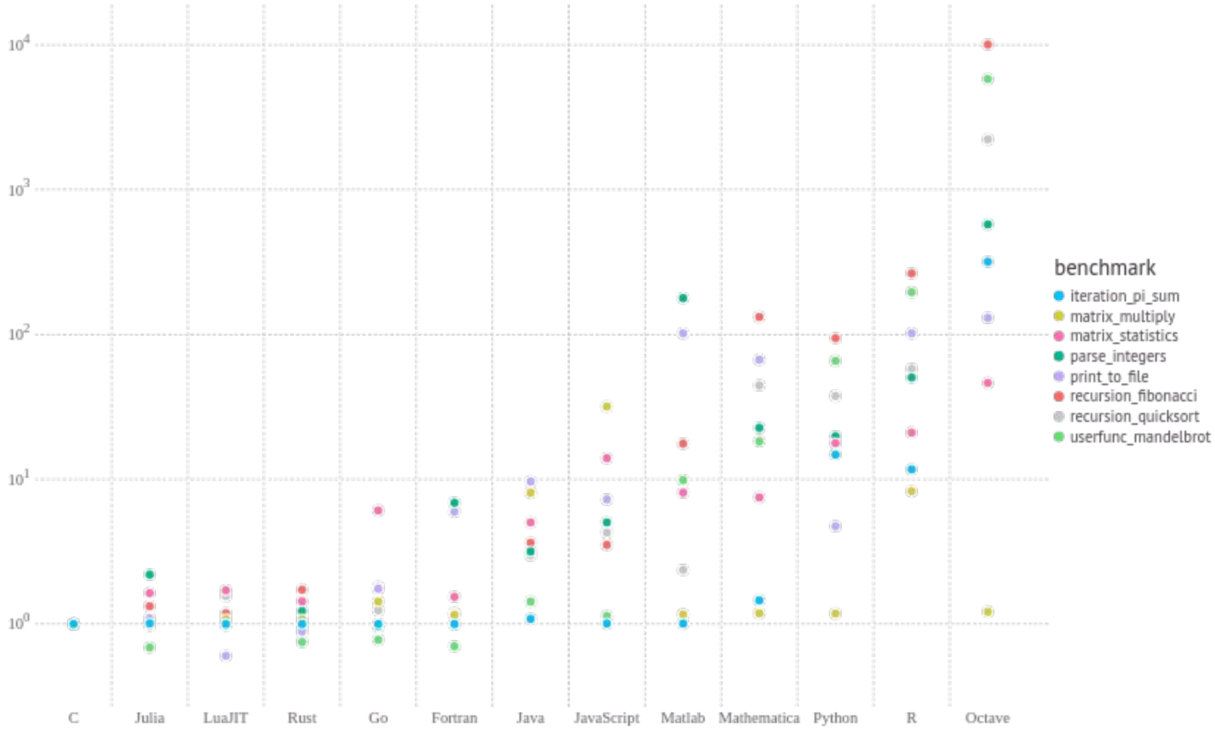
Evolution of Food.

Enter The Julian Unification



- Dynamic
- Compiled
- User Types **and** standard types
- Standalone **or** glue

Julia is Fast!!



Let us understand the necessity: Scientific AI

Machine Learning + Domain Power modeling

Mechanistics vs Non-Mechanistics Models

Let's say we want to define a nonlinear model for the population of rabbits. There are three basic ways to do this:

Rabbits tomorrow = Model(Rabbits today)

- Analytical solutions require that someone explicitly define a nonlinear model. Clearly doesn't scale
- Differential equations describe mechanisms/structure and let the equations naturally evolve this description.
  - *Rabbits'(t) = K * Rabbits(t)* encodes "the rate at which the population is growing depends on the current number of rabbits".
- Machine learning models specify a learnable black box, where with the right parameters they can fit any nonlinear function.

Which is best?

Pros and Cons of Mechanistic Models

- Mechanistic models understand the structure, so they can extrapolate for beyond data
  - Einstein's equations described black holes, and area of parameter space with infinites, decades before we could ever get data on them!
- Mechanistic models are interpretable, can be extended, transferred, analytically understood.
- Mechanistic model require that you know mechanism
  - Data goes through a filter of experts. Scientists need to confirm every term.
- None-mechanistic models can give a very predictive model directly from data, but without the interpretability or extrapolatability of mechanistic models.

Neither is better than the other. Both have advantages.

Goal: Combine mechanistic and non-mechanistic models in ways that one receives the best of both worlds.

Convolutional Neural Networks Are Structure Assumptions

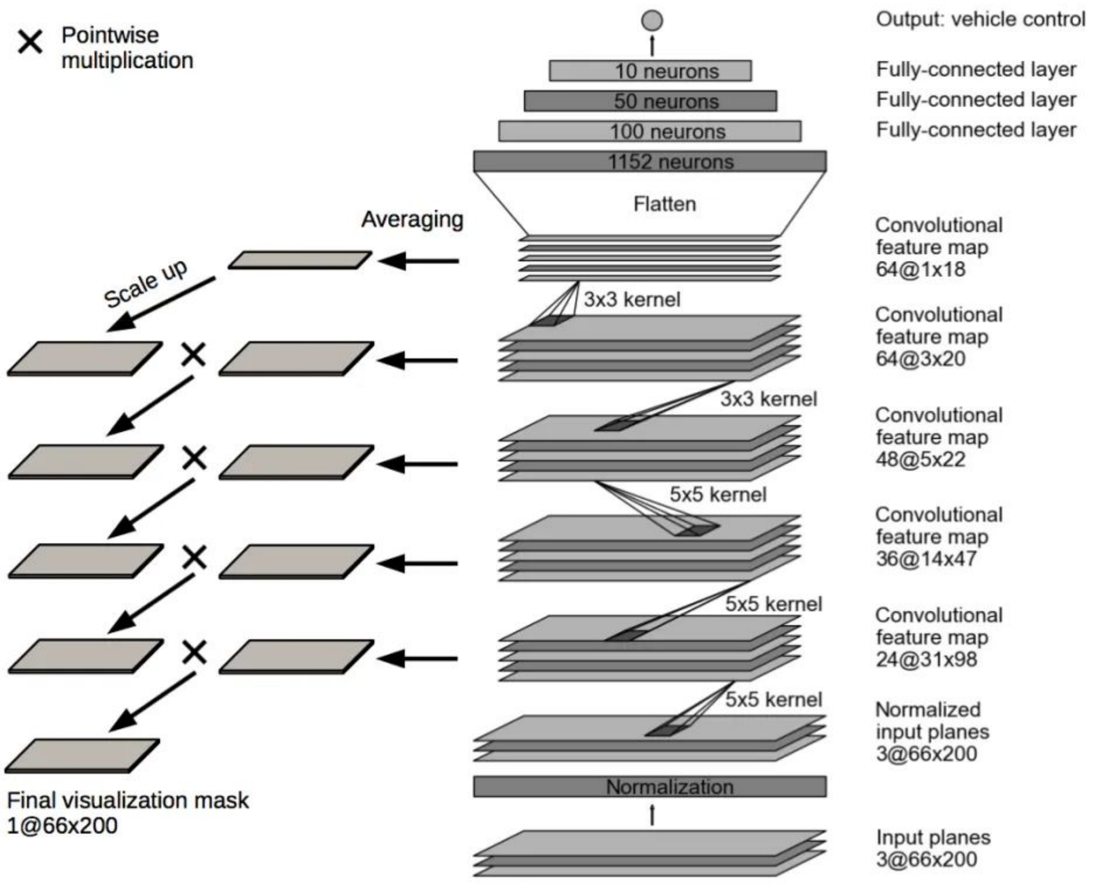


Figure 2: Block diagram of the visualization method that identifies the salient objects.

Learnable functions and UAT

- Universal approximation theorem (UAT):
  Neural Networks can get \epsilon close to any R^n -> R^m function.

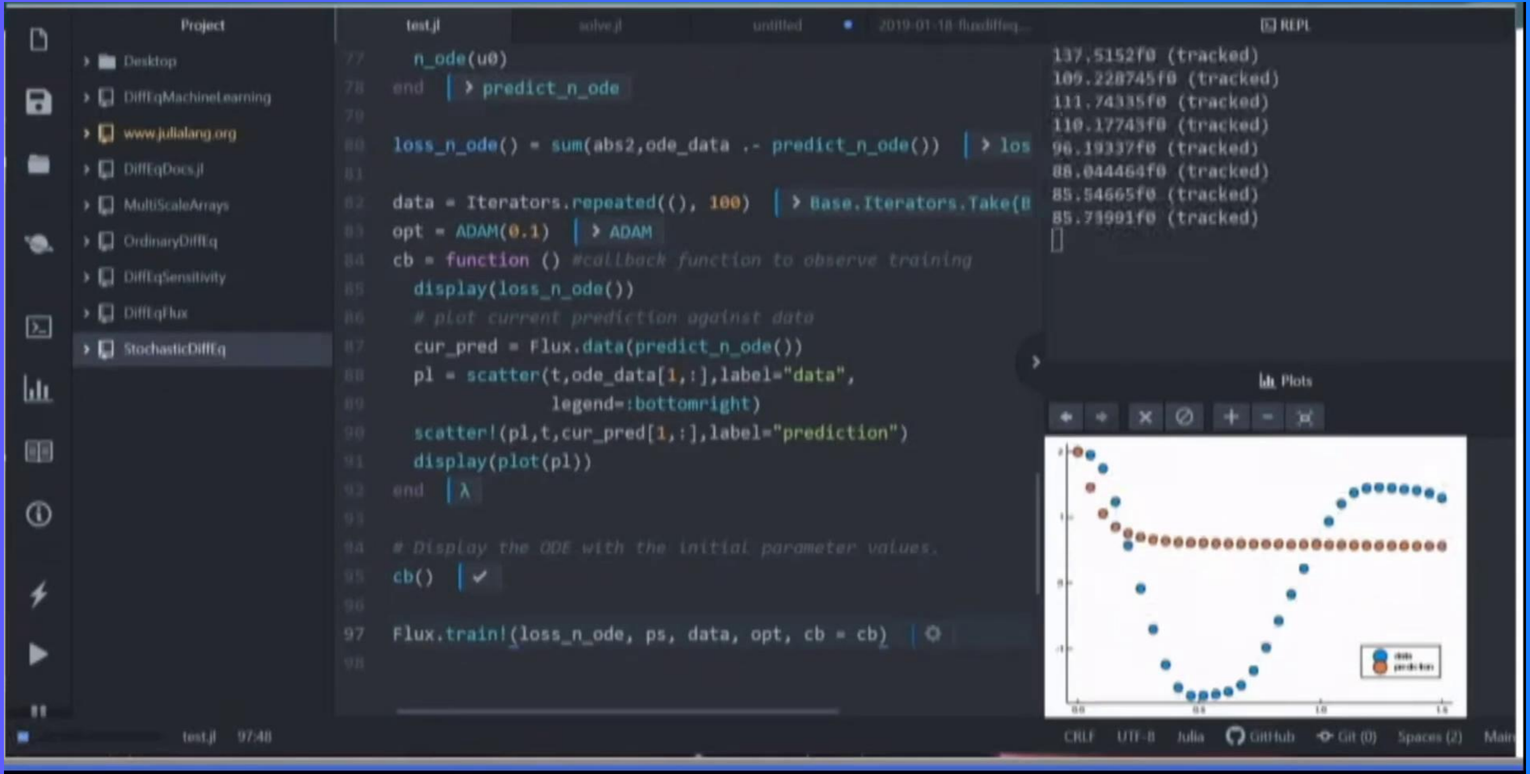
Neural Networks are just function expansions, fancy Taylor Series like things which are good for computing and bad for analysis

Latent (Neural) Differential Equations

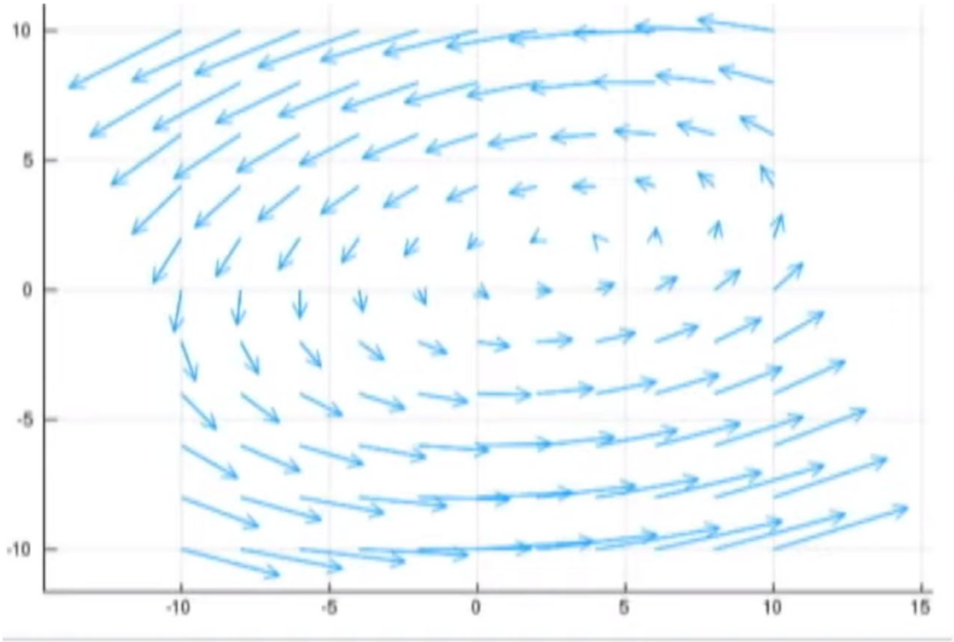
- Replace the user-defined structure with a neural network, and learn the nonlinear function for the structure.
- Neural ordinary differential equation: $u' = f(u, p, t)$ let f be a neural network.

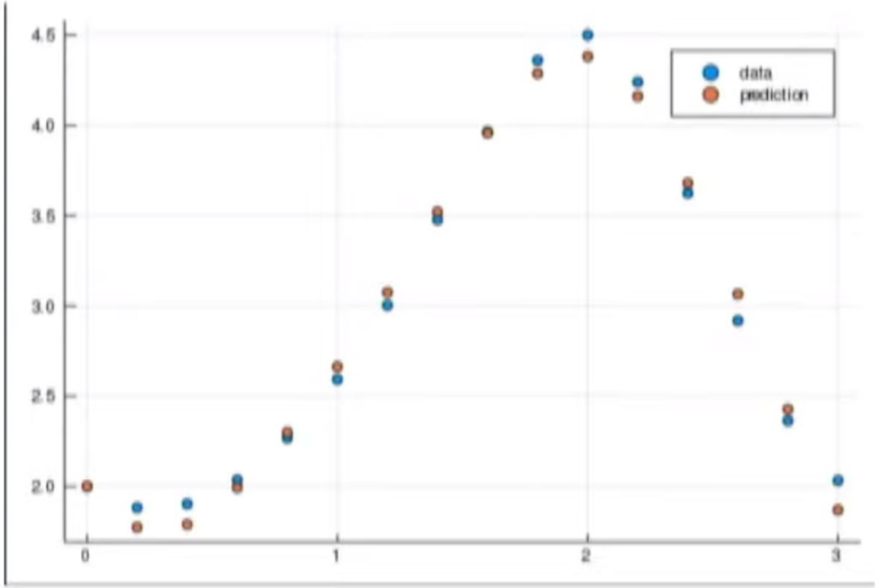
Why would we expect this to work? What is actually doing?

Neural ODE: Modeling Without Models



Direct Learning of ODEs from data:





Don't throw Away the Structure: Mix it!

Nonlinear Optimal Control is Neural ODEs

- Define an ODE where the first term's derivative is given by a neural network, the second term is a linear ODE dependent on itself and the first term.
- Use adjoint sensitivity analysis for computing fast derivatives

```
ann = Chain(Dense(2,10,tanh), Dense(10,1))

p1 = Flux.data(DiffEqFlux.destructure(ann))
p2 = Float32[-2.0,1.1]
p3 = param([p1;p2])
ps = Flux.params(p3,u0)

function dudt_(du,u,p,t)
    x, y = u
    du[1] = DiffEqFlux.restructure(ann,p[1:41])(u)[1]
    du[2] = p[end-1]*y + p[end]*x
end
prob = ODEProblem(dudt_,u0,tspan,p3)
diffeq_adjoint(p3,prob,Tsit5(),u0=u0,abstol=1e-8,reltol=1e-6)
```

Scaling the software to "real" problems

- Neural ODE with batching on the GPU (without internal data transfers) with high order adaptive implicit ODE solvers for stiff equations using matrix-free Newton-Krylov via preconditioned GMRES and trained using checkpointed adjoint equations.

```
using OrdinaryDiffEq, Flux, DiffEqFlux, DiffEqOperators, CuArrays
x = Float32[2.; 0.]|>gpu
tspan = Float32.((0.0f0,25.0f0))
dudt = Chain(Dense(2,50,tanh),Dense(50,2))|>gpu
p = DiffEqFlux.destructure(dudt)
dudt_(du,u::TrackedArray,p,t) = u .= DiffEqFlux.restructure(dudt,p)(u)
dudt_(du,u::AbstractArray,p,t) = u .= Flux.data(DiffEqFlux.restructure(dudt,p)(u))
ff = ODEFunction(dudt_,jac_prototype = JacVecOperator(dudt_,x))
prob = ODEProblem(ff,x,tspan,p)
diffeq_adjoint(p,prob,KenCarp4(linsolve=LinSolveGMRES());u0=x,
               saveat=0.0:0.1:25.0,backsolve=false)
```

Differentiable Programming Enables the full Differential Equation Solver Suite to "Neuralitize"

- Therefore, high order adaptive methods for stiff and non-stiff neural stochastic SDEs come for free

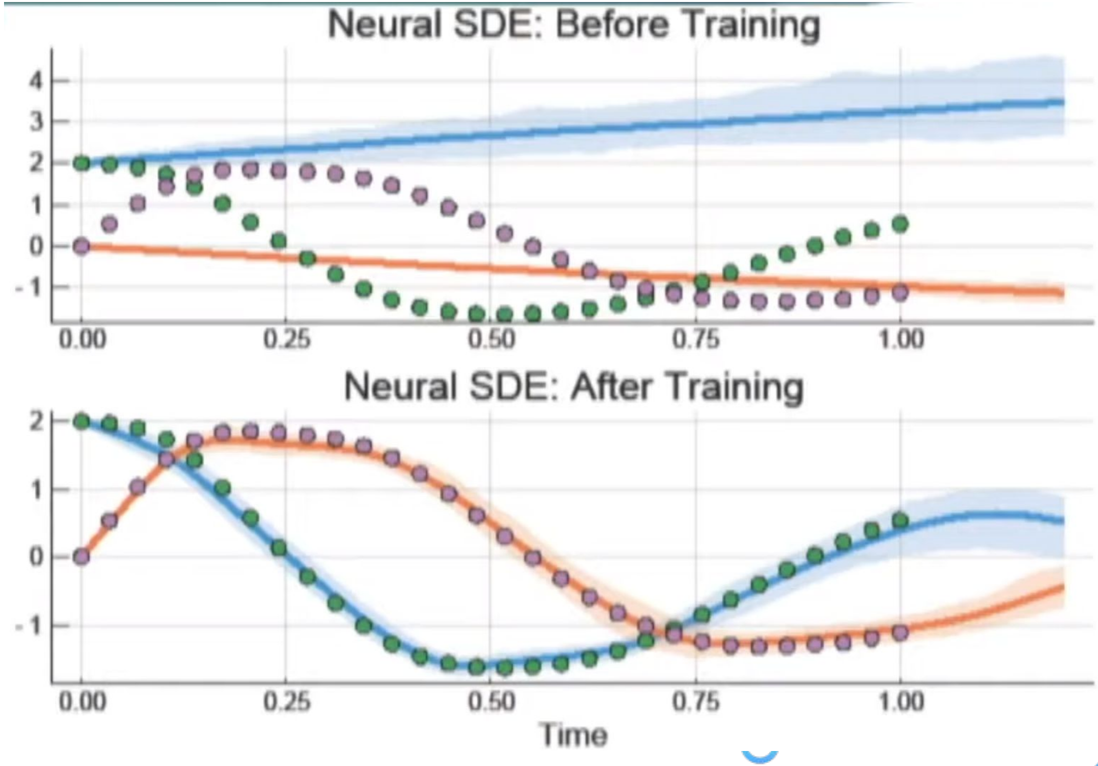
```
dudt = Chain(x -> x.^3,
             Dense(2,50,tanh),
             Dense(50,2)) |> gpu
ps = Flux.params(dudt)
n_sde = x -> neural_dmsde(dudt,x,mp,tspan,SOSRI(),
                          saveat=t,reltol=1e-1,abstol=1e-1)
```

Neural SDEs: Nonlinear Timeseries Learning and Extrapolation

DiffEqFlux.jl was the first software to be able to fit neural stochastic differential equations. These are neural differential equations with a deterministic and stochastic evolution:

$$du_t = f(u,p,t)dt + g(u,p,t)dW_t$$

Allows for direct training and discovery of financial "quant"models



Growing the equations and allowing discontinuities

- Jump stochastic differential equations allow for compound Poisson process to describe the stochastic behavior of discontinuities.
  - These jumps model regime changes in stock markets, discrete switches in biological organisms, etc

$$du_t = f(u,p,t)dt + g(u,p,t)dW_t + h(u,p,t)dN_t$$

Where dN_t is non-zero on a countable set

- Partial differential equations allow for specifying an ODE over spatial locations.

DifferentialEquations.jl can solve these equations, therefore differentiable programming means we can neuralitize them.

Try it yourself, open: docs.juliadiffeq.org

- Look at how to define a jump diffusion, put neural networks in there.
- Does it work?

This works and it trains against simulated data!

```
dudt = Chain(x -> x.^3,
             Dense(2,50,tanh),
             Dense(50,2)) |> gpu
dudt2 = Chain(Dense(2,50,tanh),
              Dense(50,2)) |> gpu
ps = Flux.params(dudt,dudt2)

g(u,p,t) = mp.*u
n_sde = function (x)
    dudt_(u,p,t) = dudt(u)
    rate(u,p,t) = 2.0
    affect!(integrator) = (integrator.u = dudt2(integrator.u))
    jump = ConstantRateJump(rate,affect!)
    prob = SDEProblem(dudt_,g,param(x),tspan,nothing)
    jump_prob = JumpProblem(prob,Direct(),jump,save_positions=(false,false))
    solve(jump_prob, SOSRI(); saveat=t ,abstol = 0.1, reltol = 0.1) |> Tracker.collect
```

Now let's define a neural semilinear PDE

- Define a method of lines discretization of the semilinear heat equation, and replace the nonlinearity with a neural network

$$du_t = \Delta u + f(u)$$

2-dimensional GPU-accelerated Neural PDE

```
ann = Chain(Dense(3,50,tanh), Dense(50,3)) |> gpu
p1 = DiffEqFlux.destructure(ann)
ps = Flux.params(ann)

_ann = (u,p) -> reshape(p[3*50+51 : 2*3*50+50],3,50)*
                tanh.(reshape(p[1:3*50],50,3)*u + p[3*50+1:3*50+50]) +
                      p[2*3*50+51:end]

function dudt_(_u,p,t)
  u = reshape(_u,N,N,3)
  A = u[:,:,1]
  DA = D .* (A*Mx + My*A)
  _du = mapslices(x -&gt; _ann(x,p),u,dims=3) |> gpu
  du = reshape(_du,N,N,3)
  x = vec(cat(du[:,:,1]+DA,du[:,:,2],du[:,:,3],dims=3))
end
```

Train it with a high order adaptive methods for semi-stiff ODEs

```
prob = ODEProblem(dudt_,vec(Flux.data(u0)),tspan,Flux.data(p1))
@time diffeq_fd(p1,Array,length(u0)*length(0.0f0:5.0f0:100.0f0),
                prob,ROCK2(),progress=true,
                saveat=0.0f0:5.0f0:100.0f0)
```

Current state

- We can train neural ODEs, neural SDEs, neural jump SDEs, neural PDEs, neural DDEs, neural DAEs, and neural Gillespie equations.
  - DiffEqFlux.jl is the first library to handle stiff neural ODEs, and any form of all of the other equations@
  - DiffEqFlux.jl is GPU-accelerated and allows for forward-mode AD, reverse-mode AD, and the use of O(1) memory adjoint representations (backsolve and checkpointing)
- A few issues were spotted in here:
  - PDEs are still at the cusp: Maplslices and GMRES could be faster on the GPU, changes need to be "packaged"
  - Low rate jumps are a bear to fit

Different form of combinations

- Neural networks can be defined where the "activations"are nonlinear functions described by differential equations
- Neural networks can be defined where some layers are ODE solves.
- Cost functions on ODEs can define neural networks.

All have different use cases.

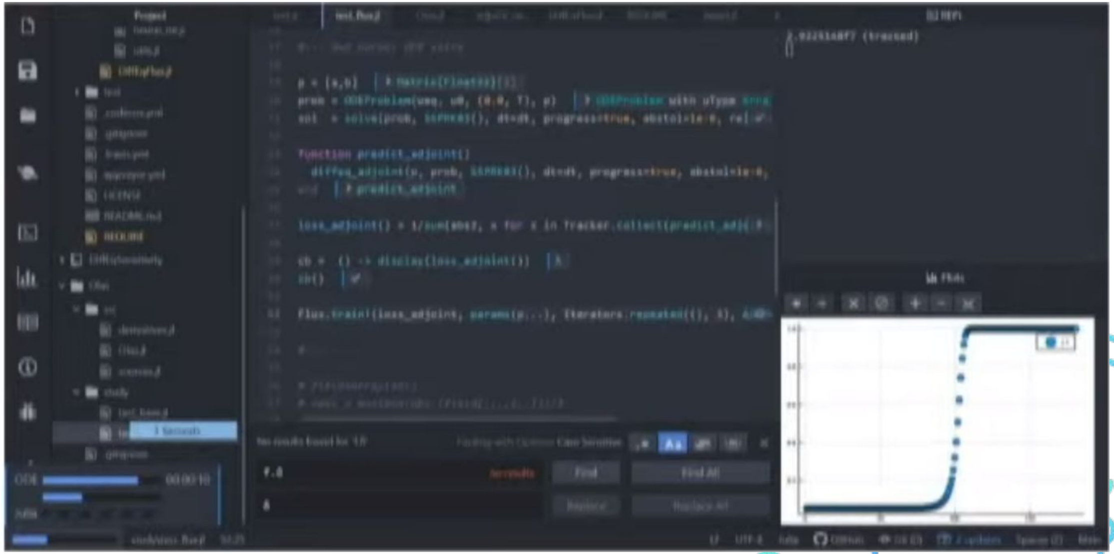
Neural PDEs for Acceleration: Automated Quasilinear Approx.

$$(u \cdot \nabla)u \approx a(u,p,t)\nabla u$$

- Navier-Stokes is HARD!

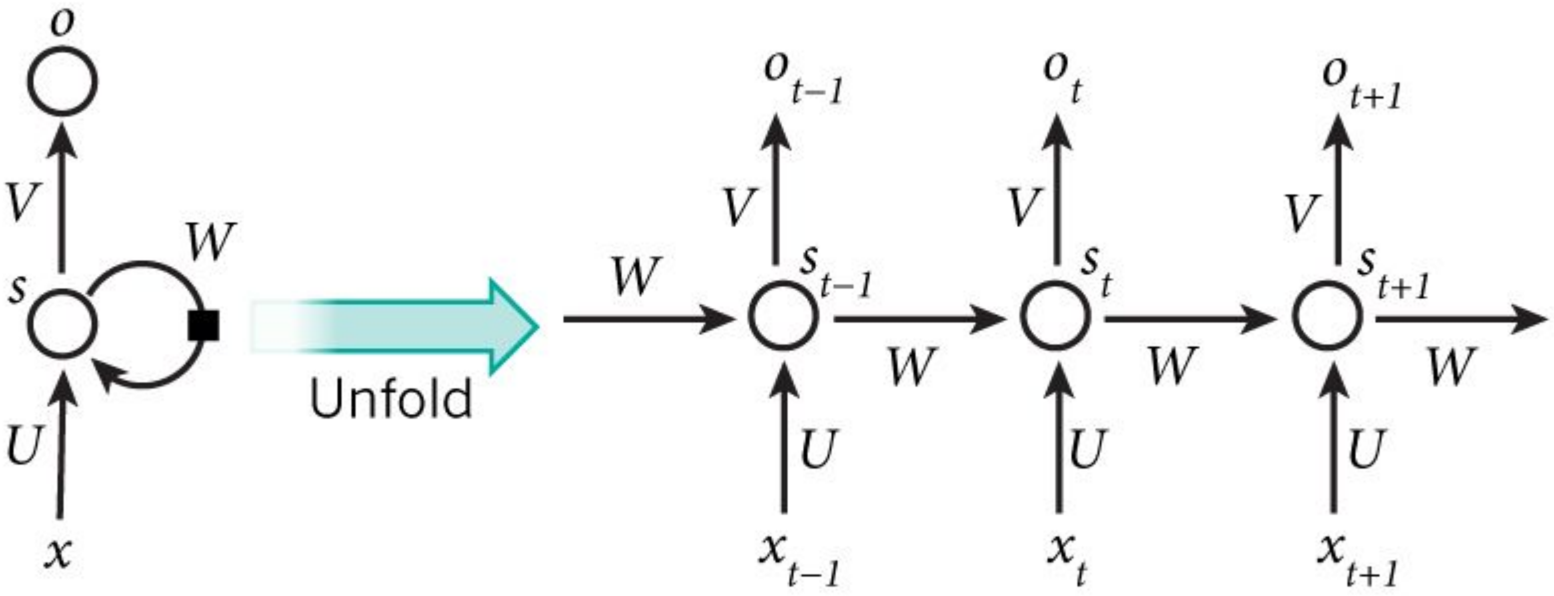
$$\overbrace{\underbrace{\frac{\partial \mathbf{u}}{\partial t}}_{\text{Variation}} + \underbrace{(\mathbf{u}\cdot\nabla)\mathbf{u}}_{\text{Convection}}}^{\text{Inertia (per volume)}} \overbrace{-\underbrace{\nu\nabla^2\mathbf{u}}_{\text{Diffusion}} = \underbrace{-\nabla w}_{\text{Internal source}}}^{\text{Divergence of stress}} + \underbrace{\mathbf{g}}_{\text{External source}}$$

- People attempt to solve Navier-Stokes by quasilinearization of the convection term, making it:
- Instead of picking a form for the a, replace it with a neural network and learn it form small scale simulations!



Neural ODEs for memory-efficient ML

Jesse Bettencourt UofT (Deepmind) RNN are low-memory representations of RNN

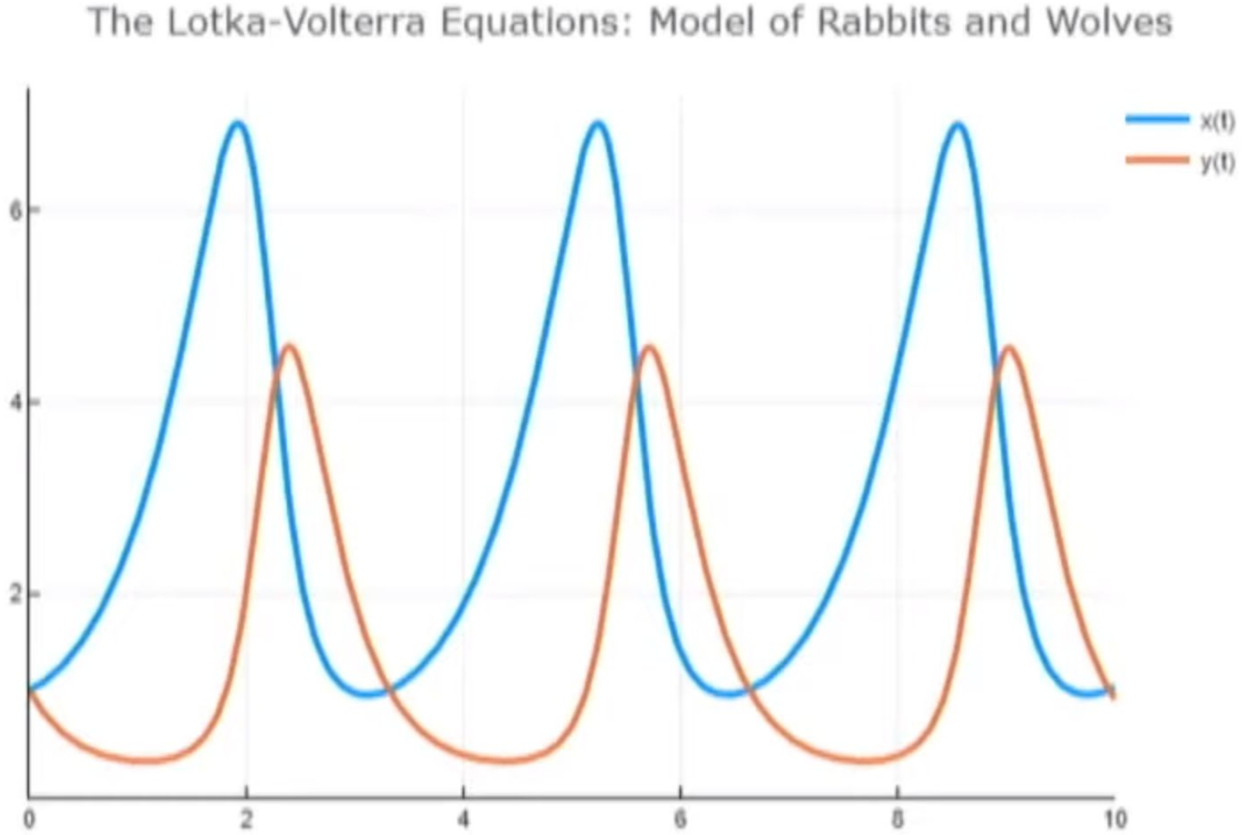


Neural Network Surrogates for Real-Time Nonlinear Approx. c

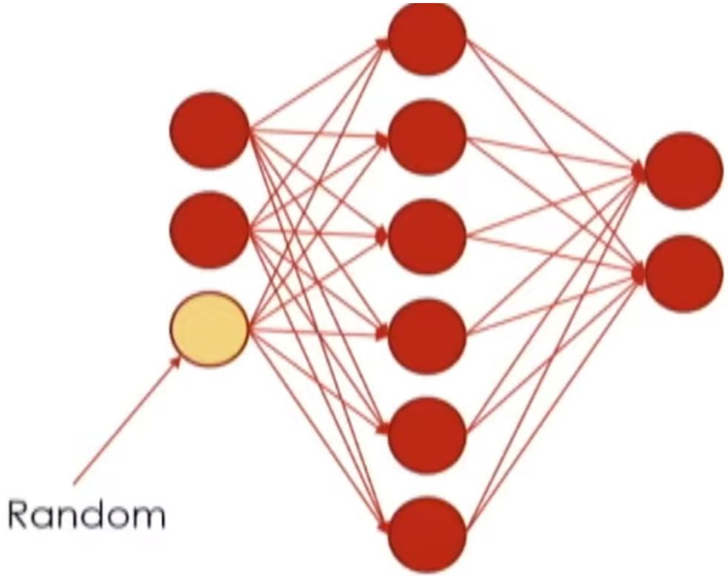
$$\frac{dx}{dt} = \alpha x - \beta xy$$

$$\frac{dy}{dt} = \delta xy - \gamma y$$

Problem: Given alpha and delta, give me beta and gamma s.t solution stays in [0, 6]



A Complimentary Inversion Network



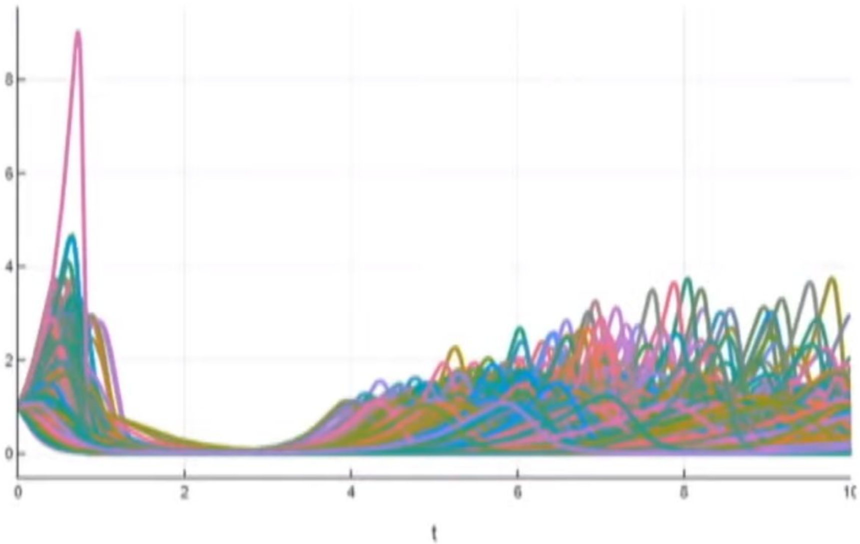
Cost function is evaluated against the previous network

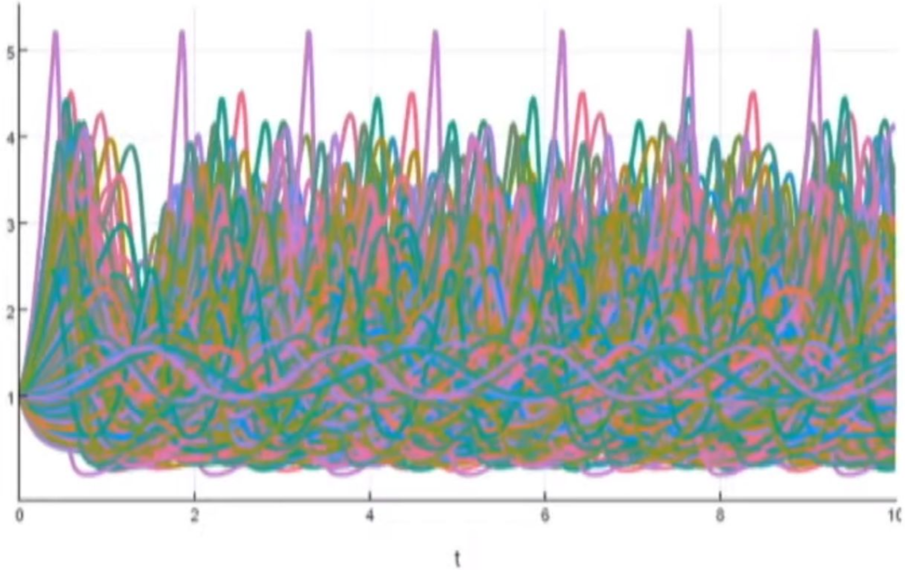
Trained neural net procedure:

- Take N random numbers
- Throw each through the neural net
- Predict the chance that the result satisfies the condition from the neural net 1
- Choose the choice which has the highest chance

Inversion is accurate and independent of simulation time

1999/2000 correct ... but adapt data (infinite data!) ... 2000/2000 correct



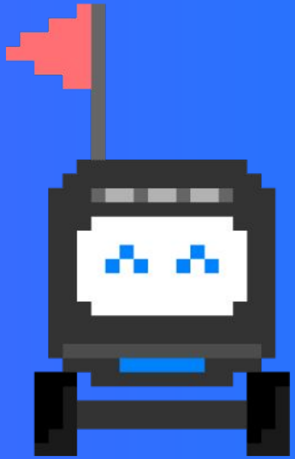


How to try Julia

Resources

- https://julialang.org/
- https://hub.docker.com/_/julia
- https://juliabox.com/
- https://julialang.org/learning/

Thank you



Cristian Garcia

Email: **cgarcia.e88@gmail.com**

Twitter: **@cgarciae88**

LinkedIn: **@cgarciae**

The AI & Robotics team

kiwibot
-Food delivery-

German David Cardozo
david@kiwicampus.com
AI & Robotics Lead

Juan David Galvis
juangalvis@kiwicampus.com
Robotics and Control Lead Engineer

Camilo Alvis
camiloalvis@kiwicampus.com
Robotics software engineer

Robin Deuber
robindeuber@kiwicampus.com
Robotics engineer

Marcela Gomez
marcelagomez@kiwicampus.com
Project Manager

https://www.kiwicampus.com/team

Juan Pablo Ramirez
juanramirez@kiwicampus.com
Full-stack Developer

Juan Francisco Jurado P.
jj@kiwicampus.com
Embedded Systems Engineer

Juan David Rios
juanrios@kiwicampus.com
Machine Learning Engineer

John A. Betancourt G.
john@kiwicampus.com
Computer Vision Engineer

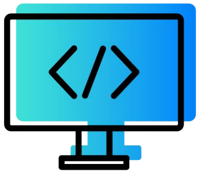
Name a section
Title of section

Title of slide

You can also add and show different types of bullets or lists along with icons or small images. There's an icon folder you can find different kiwi icons that can be used. If you need another one please contact Alexandra.

Ut enim ad minim veniam, quis nostrud exercitation ullamco

Ut enim ad minim veniam, quis nostrud exercitation ullamco



Ut enim ad minim veniam, quis nostrud exercitation ullamco

Comparison 1

You can add a small text and/or an image or graph

Restaurant	Status
Boba Ninja	Great
Bongo Burger (Center)	Great
Sumo Roll	to be improved
Gypsy's Trattoria Italiana	Great
El Burro Picante	Lack of communication
Bongo Burger (Dwight)	Great
Seniores Pizza	Lack of communication
Cheese n' Stuff	Great
Dumpling Express	Lack of communication

Comparison 2

You can add a small text and/or an image or graph

Dumpling Express	Lack of communication
Poke Parlor	Great
McDonald's (Shattuck)	Lack of communication
Cancun Sabor Mexicano	to be improved
The Halal Guys Berkeley	Great
Sushinista	to be improved
Ma Lai Zui Tasty Bowl	Lack of communication
Crave Subs	Great
Bag O' Crab	Lack of communication
PInky & Reds	Lack of communication

Title

Name	#	#	#	#	#	#	#
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Total	**000**	**$000**	**$000**	**$000**	**$000**	**$000**	**$000**

Another way of showing graphs

Name	#	#	#	#	#	#	#
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Name	0	0	0	0	0	0	0
Total	**000**	**$000**	**$000**	**$000**	**$000**	**$000**	**$000**

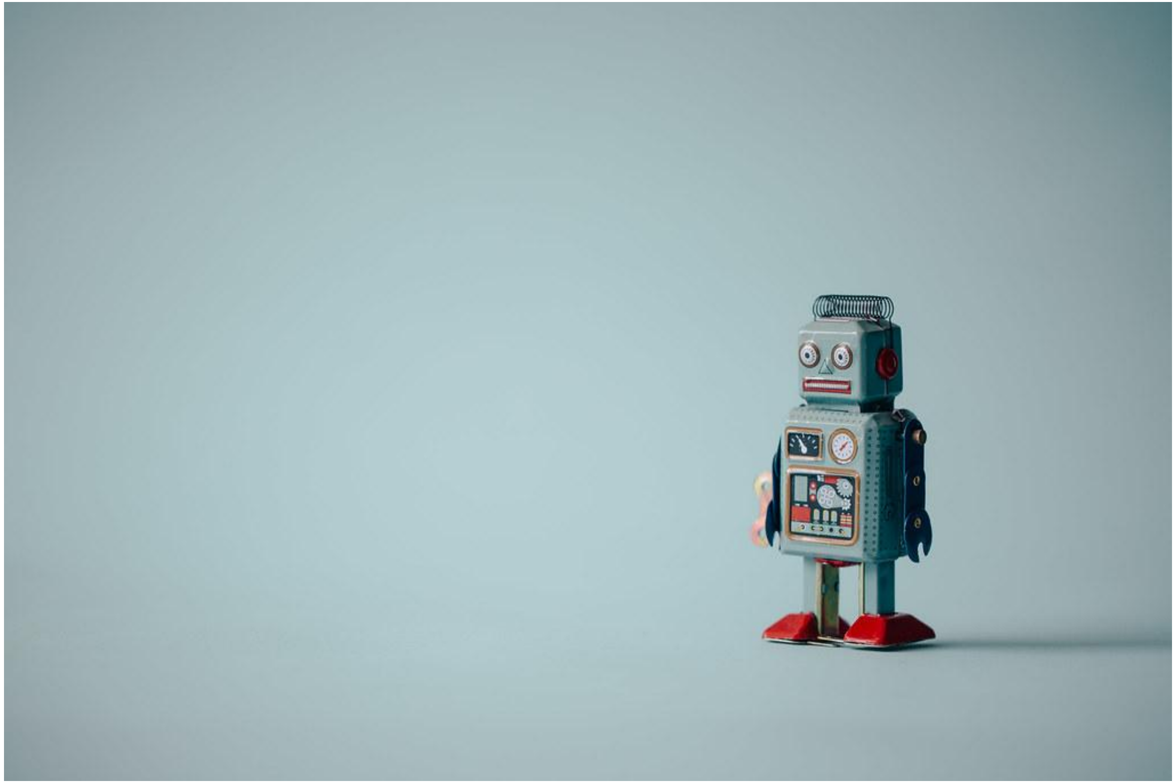
Another way of showing images/videos

Title

Here you can add a small text and or bullet points. On the white part, add image / graph to complement information.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

- Ut enim ad minim veniam, quis nostrud exercitation ullamco
- laboris nisi ut aliquip ex ea commodo consequat.
- Duis aute irure dolor in reprehenderit in voluptate.



Name a section
Title of section