Julia in Research

Albin Heimerson

What I do

PhD student doing ML for datacenter/cloud control

- Controlling large interconnected systems
- A lot of data
- Reinforcement learning for control
- Modelling to improve learning

Why I use Julia

Partly because I just enjoy the language, but also

- Automatic Differentiation for the whole language
- Composability between packages
- High level and high performance
- Transparency in library code, Julia all the way

Trade-offs in Automatic Differentiation

Julia

- ForwardDiff/ReverseDiff - Operator overloading
- Zygote - Source to source reverse mode
- Enzyme - LLVM source to source reverse mode, experimental

Flexible choices good for different things, usable with most packages.

Python

Does not handle dynamic structure of the language.

- Tensorflow - Essentially source to source, but make user write IR
- PyTorch - Operator overloading
- Jax - Non-standard interpretation to create IR, then TF

Fast with standard deep learning, not as fast or flexible outside.

http://www.stochasticlifestyle.com/engineering-trade-offs-in-automatic-differentiation-from-tensorflow-and-pytorch-to-jax-and-julia/

Trade-offs in Automatic Differentiation

Julia

- ForwardDiff/ReverseDiff - Operator overloading
- Zygote - Source to source reverse mode
- Enzyme - LLVM source to source reverse mode, experimental

Flexible choices good for different things, usable with most packages.

Python

Does not handle dynamic structure of the language.

- Tensorflow - Essentially source to source, but make user write IR
- PyTorch - Operator overloading
- Jax - Non-standard interpretation to create IR, then TF

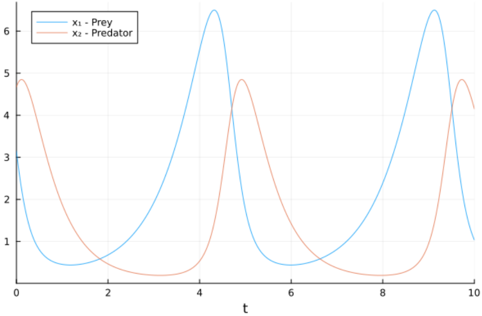
Fast with standard deep learning, not as fast or flexible outside.

http://www.stochasticlifestyle.com/engineering-trade-offs-in-automatic-differentiation-from-tensorflow-and-pytorch-to-jax-and-julia/

Embedding ML into differential equations

Example: Lotka-Volterra[1]

$$\frac{dx_1}{dt} = \alpha x_1 - \beta x_1 x_2$$
$$\frac{dx_2}{dt} = -\delta x_2 + \gamma x_1 x_2$$

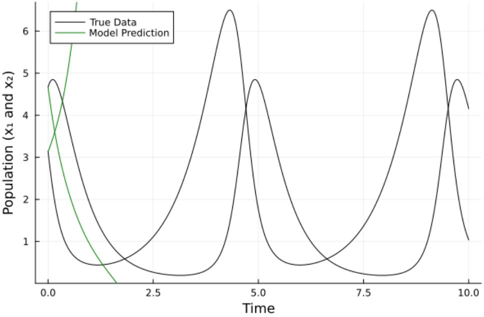


[1] https://docs.sciml.ai/Overview/stable/showcase/missing_physics/

Embedding ML into differential equations

Learn *unknown* dynamics with neural networks

$$\frac{dx_1}{dt} = \alpha x_1 + NN_1^{\boldsymbol{\theta}}(x_1, x_2)$$
$$\frac{dx_2}{dt} = -\delta x_2 + NN_2^{\boldsymbol{\theta}}(x_1, x_2)$$



Embedding ML into differential equations

Loss: difference between simulated $\boldsymbol{x}$ and real $\boldsymbol{x}_{data}$

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left[\boldsymbol{x}(t_i) - \boldsymbol{x}_{data}(t_i)\right]^2$$

```
function lotka_nn!(dx, x, p, t)
    x_nn = model(x, p)
    dx[1] = alpha * x[1] + x_nn[1]
    dx[2] = -delta * x[2] + x_nn[2]
end
```

```
function loss(p)
    prob = ODEProblem(lotka_nn!, x_train[:, 1], (0, 10), p)
    xhat = Array(solve(prob, saveat=t_train))
    mean(abs2, x_train .- xhat)
end
```

Embedding ML into differential equations

Loss: difference between simulated $\boldsymbol{x}$ and real $\boldsymbol{x}_{data}$

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left[\boldsymbol{x}(t_i) - \boldsymbol{x}_{data}(t_i)\right]^2$$

```
function lotka_nn!(dx, x, p, t)
    x_nn = model(x, p)
    dx[1] = alpha * x[1] + x_nn[1]
    dx[2] = -delta * x[2] + x_nn[2]
end
```

```
function loss(p)
    prob = ODEProblem(lotka_nn!, x_train[:, 1], (0, 10), p)
    xhat = Array(solve(prob, saveat=t_train))
    mean(abs2, x_train .- xhat)
end
```

Embedding ML into differential equations

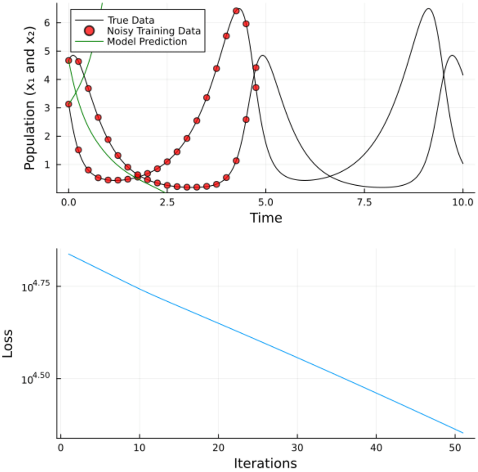
Loss: difference between simulated $\boldsymbol{x}$ and real $\boldsymbol{x}_{data}$

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left[\boldsymbol{x}(t_i) - \boldsymbol{x}_{data}(t_i)\right]^2$$

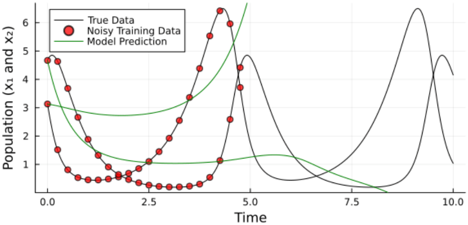
```
function lotka_nn!(dx, x, p, t)
    x_nn = model(x, p)
    dx[1] = alpha * x[1] + x_nn[1]
    dx[2] = -delta * x[2] + x_nn[2]
end
```

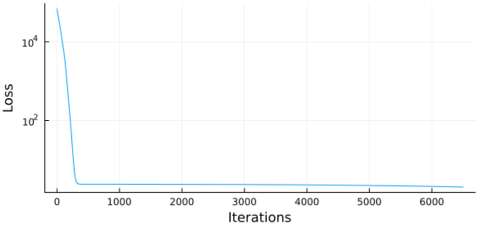
```
function loss(p)
    prob = ODEProblem(lotka_nn!, x_train[:, 1], (0, 10), p)
    xhat = Array(solve(prob, saveat=t_train))
    mean(abs2, x_train .- xhat)
end
```

Embedding ML into differential equations

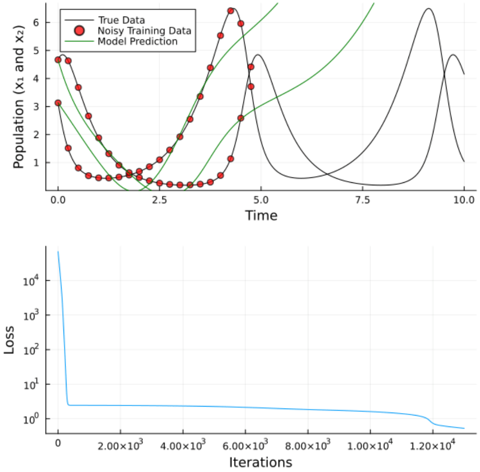


Embedding ML into differential equations

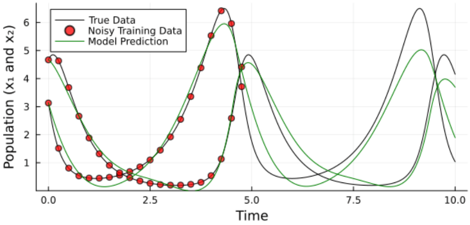


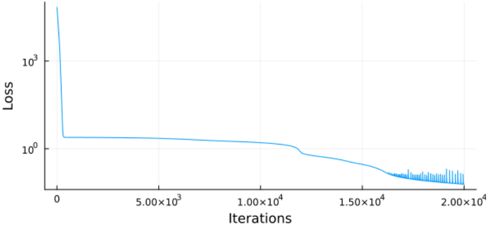


Embedding ML into differential equations

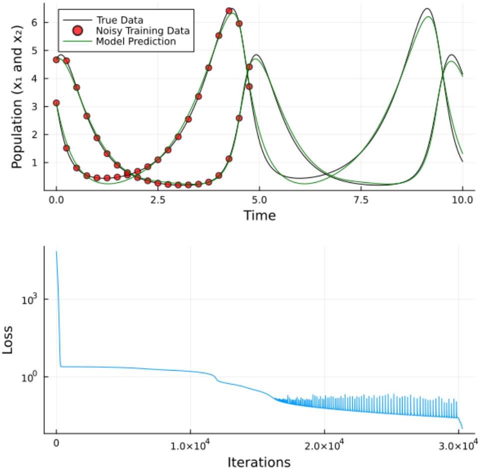


Embedding ML into differential equations





Embedding ML into differential equations



Embedding ML into differential equations

Remarks on the example:

- Found missing dynamics with NN
- Symbolic regression
- More effect in more complex models

Composability and Specialization

Performant composability, specialized code if types are inferred

Some interesting types

- Extended and arbitrary precision numbers
- Dual numbers for AD
- Intervals for interval arithmetic
- Distributions represented by number types
- GPU-arrays and distributed arrays
- Matrices with structure that can be exploited for performance, e.g. diagonal/banded/sparse

Performance

A few of the nice points

- Fast specialized code when type-stable
- Vectorized code vs for loops
- Built in parallelism

Some possible pitfalls

- Type instabilities
- Allocations
- Global scope

Performance

A few of the nice points

- Fast specialized code when type-stable
- Vectorized code vs for loops
- Built in parallelism

Some possible pitfalls

- Type instabilities
- Allocations
- Global scope

Introspection

- Open source language and package ecosystem
- One language all the way down
- Useful tools (debugging, profiling) and macros
  - `@show func(a, b, c)`
  - `@edit func(a, b, c)`
  - `@code_llvm func(a, b, c)`

Summary

Julia is nice for many reasons, some of them are

- Flexible AD
- Composability
- High level and high performance
- Transparency, Julia all the way

Thanks for listening!

Questions?

Summary

Julia is nice for many reasons, some of them are

- Flexible AD
- Composability
- High level and high performance
- Transparency, Julia all the way

Thanks for listening!

Questions?