Understanding Neural Networks: A From Scratch Implementation in Julia

2nd International Conference on Smart Industry, Technology and Environment

---

Abdelbacet Mhamdi
Dr.-Ing. in EE – MT @ ISET Bizerte

Presented on the 9th of November 2024 at Hammamet

Outlines

1. Context & Motivations
2. Train a Neural Network
3. Numerical Experiments
4. In Closing

Context & Motivations

Artificial Neural Network From Scratch In `Julia`

Why?

① understanding the architecture of ANN;

② grasp the mathematical foundations of ANN;

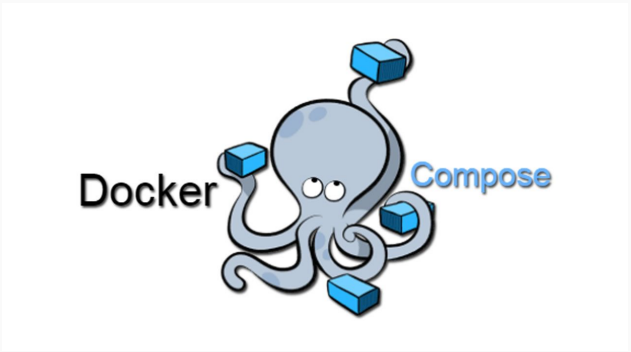
③ develop a practical tool for learning ANN;

④ tweaking: loss, activation functions, optimizers, etc.

Programming Language

julialang.org/

Development Environments

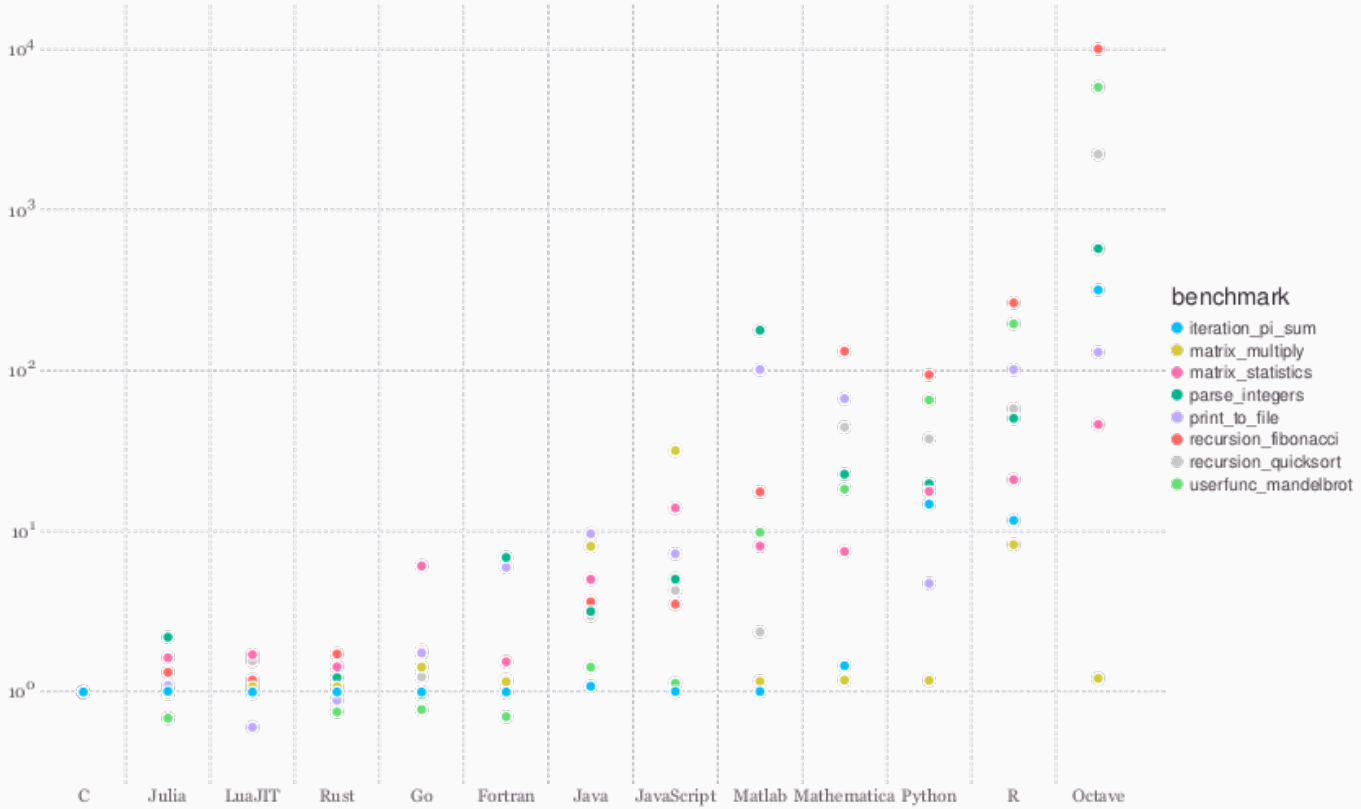


- ▲ $ docker compose up
- ▼ $ docker compose down

Julia in a Nutshell

- Fast
- Dynamic
- Reproducible
- Composable
- General
- Open Source

Julia Micro-Benchmarks (1/2)



https://julialang.org/benchmarks

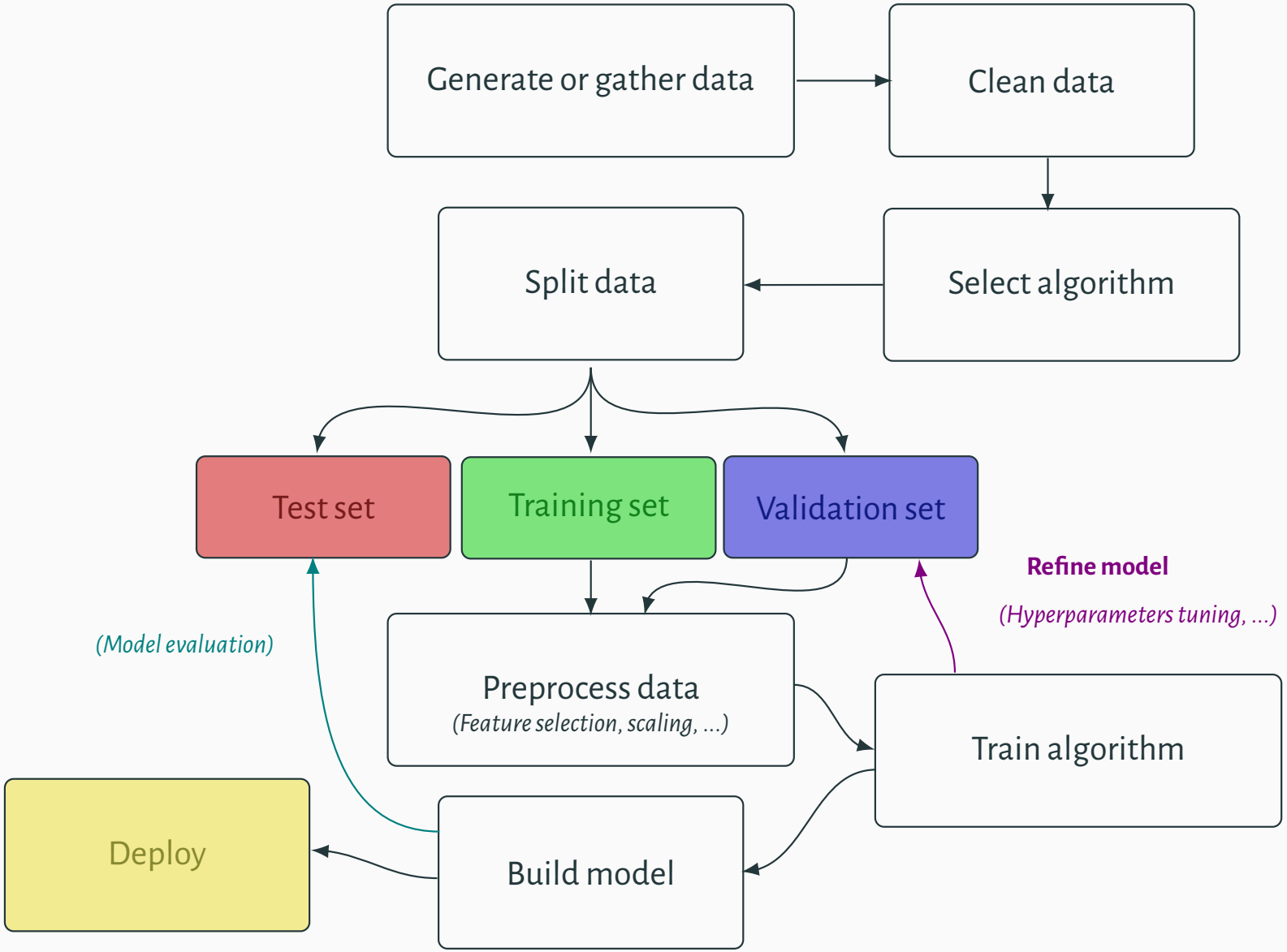
Julia Micro-Benchmarks (2/2)

Geometric Means of Micro-Benchmarks by Language

1	C	1.0
2	Julia	1.17006
3	LuaJIT	1.02931
4	Rust	1.0999
5	Go	1.49917
6	Fortran	1.67022
7	Java	3.46773
8	JavaScript	4.79602
9	Matlab	9.57235
10	Mathematica	14.6387
11	Python	16.9262
12	R	48.5796
13	Octave	338.704

Train a Neural Network

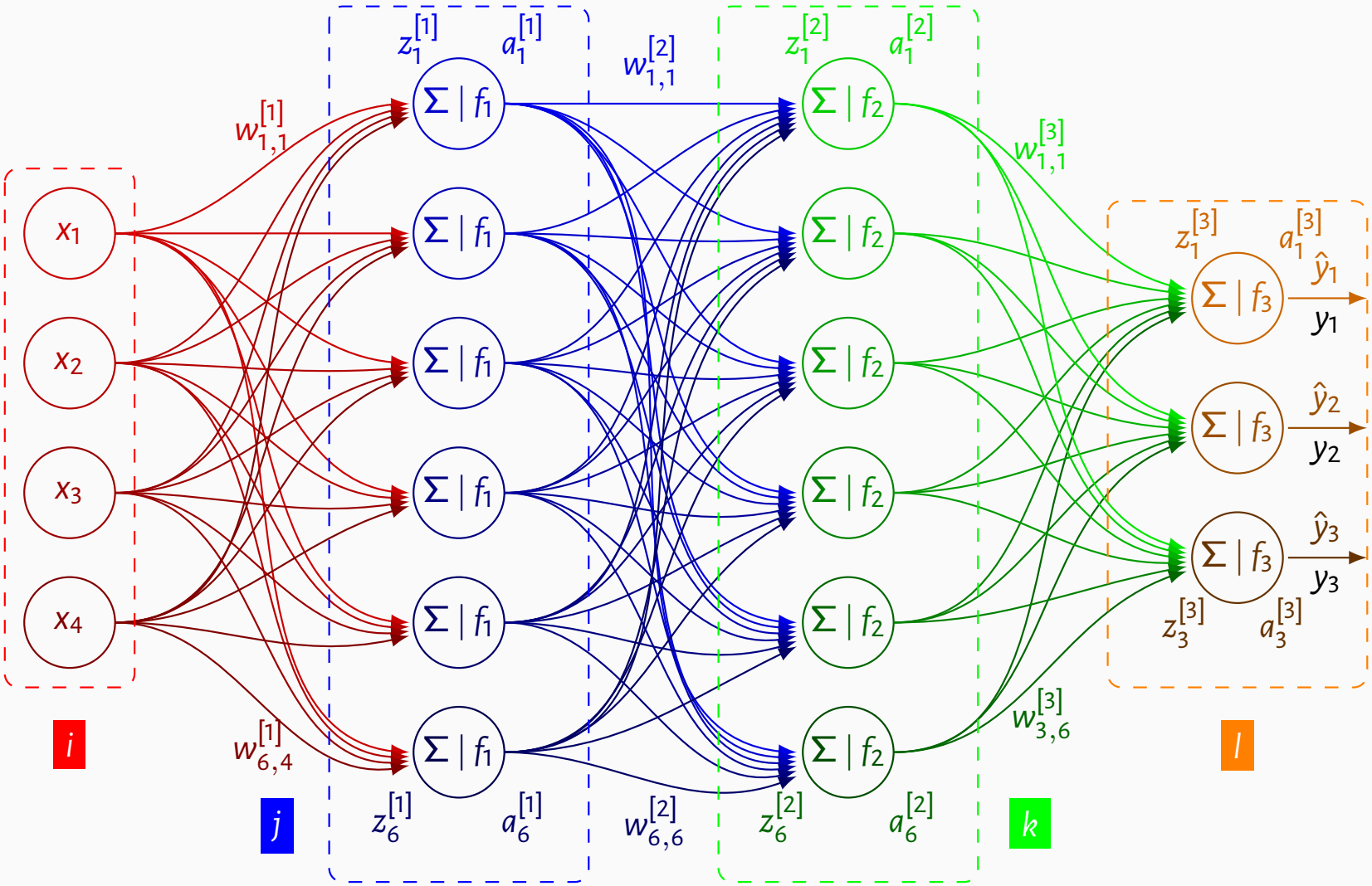
Workflow


Generate or gather data
Clean data
Split data
Select algorithm
Test set
Training set
Validation set
Refine model
(Hyperparameters tuning, ...)
(Model evaluation)
Preprocess data
(Feature selection, scaling, ...)
Train algorithm
Deploy
Build model

Core Components

- Layer Struct
- Activation Function
- Loss Function
- Regularization
- Optimizer
- Forward and backward propagation

Architecture



Multivariate chain rule

Output layer $\longrightarrow$ hidden layer #2

$$\frac{\partial \hat{y}_l}{\partial w_{l,k}^{[3]}} = \underbrace{\frac{\partial \hat{y}_l}{\partial z_l^{[3]}}}_{\dot{f}_3\left(z_l^{[3]}\right)} \underbrace{\frac{\partial z_l^{[3]}}{\partial w_{l,k}^{[3]}}}_{a_k^{[2]}}$$

Output layer $\longrightarrow$ hidden layer #1

$$\frac{\partial \hat{y}_l}{\partial w_{k,j}^{[2]}} = \underbrace{\frac{\partial \hat{y}_l}{\partial z_l^{[3]}}}_{\dot{f}_3\left(z_l^{[3]}\right)} \underbrace{\frac{\partial z_l^{[3]}}{\partial a_k^{[2]}}}_{w_{l,k}^{[3]}} \underbrace{\frac{\partial a_k^{[2]}}{\partial z_k^{[2]}}}_{\dot{f}_2\left(z_k^{[2]}\right)} \underbrace{\frac{\partial z_k^{[2]}}{\partial w_{k,j}^{[2]}}}_{a_j^{[1]}}$$

Output layer $\longrightarrow$ input layer

$$\frac{\partial \hat{y}_l}{\partial w_{j,i}^{[1]}} = \underbrace{\frac{\partial \hat{y}_l}{\partial z_l^{[3]}}}_{\dot{f}_3\left(z_l^{[3]}\right)} \underbrace{\frac{\partial z_l^{[3]}}{\partial a_k^{[2]}}}_{w_{l,k}^{[3]}} \underbrace{\frac{\partial a_k^{[2]}}{\partial z_k^{[2]}}}_{\dot{f}_2\left(z_k^{[2]}\right)} \underbrace{\frac{\partial z_k^{[2]}}{\partial a_j^{[1]}}}_{w_{k,j}^{[2]}} \underbrace{\frac{\partial a_j^{[1]}}{\partial z_j^{[1]}}}_{\dot{f}_1\left(z_j^{[1]}\right)} \underbrace{\frac{\partial z_j^{[1]}}{\partial w_{j,i}^{[1]}}}_{x_i}$$

Optimization

Here is a list of some optimizers for artificial neural networks:

$$\Delta\hat{\mathcal{W}} \triangleq \mathcal{F}\left(\nabla \underbrace{\mathcal{J}\left(\hat{\mathcal{W}}\right)}_{\text{Loss Function}}\right) \equiv \hat{\mathcal{W}} \triangleq \hat{\mathcal{W}} + \mathcal{F}\left(\nabla\mathcal{J}\left(\hat{\mathcal{W}}\right)\right)$$

$$\nabla\mathcal{J}\left(\hat{\mathcal{W}}\right) = \begin{bmatrix} \frac{\partial\mathcal{J}}{\partial\hat{w}_0} \\ \vdots \\ \frac{\partial\mathcal{J}}{\partial\hat{w}_n} \end{bmatrix}$$

Stochastic Gradient Descent (SGD)

$$\hat{\mathcal{W}} \triangleq \hat{\mathcal{W}} - \eta\nabla\mathcal{J}\left(\hat{\mathcal{W}}\right)$$

Mini-batch Gradient Descent

$$\hat{\mathcal{W}} \triangleq \hat{\mathcal{W}} - \frac{\eta}{m}\nabla\sum_{i=1}^{m}\mathcal{J}\left(\hat{\mathcal{W}}\right) \quad \longleftarrow \; m \text{ denotes the size of the mini-batch}$$

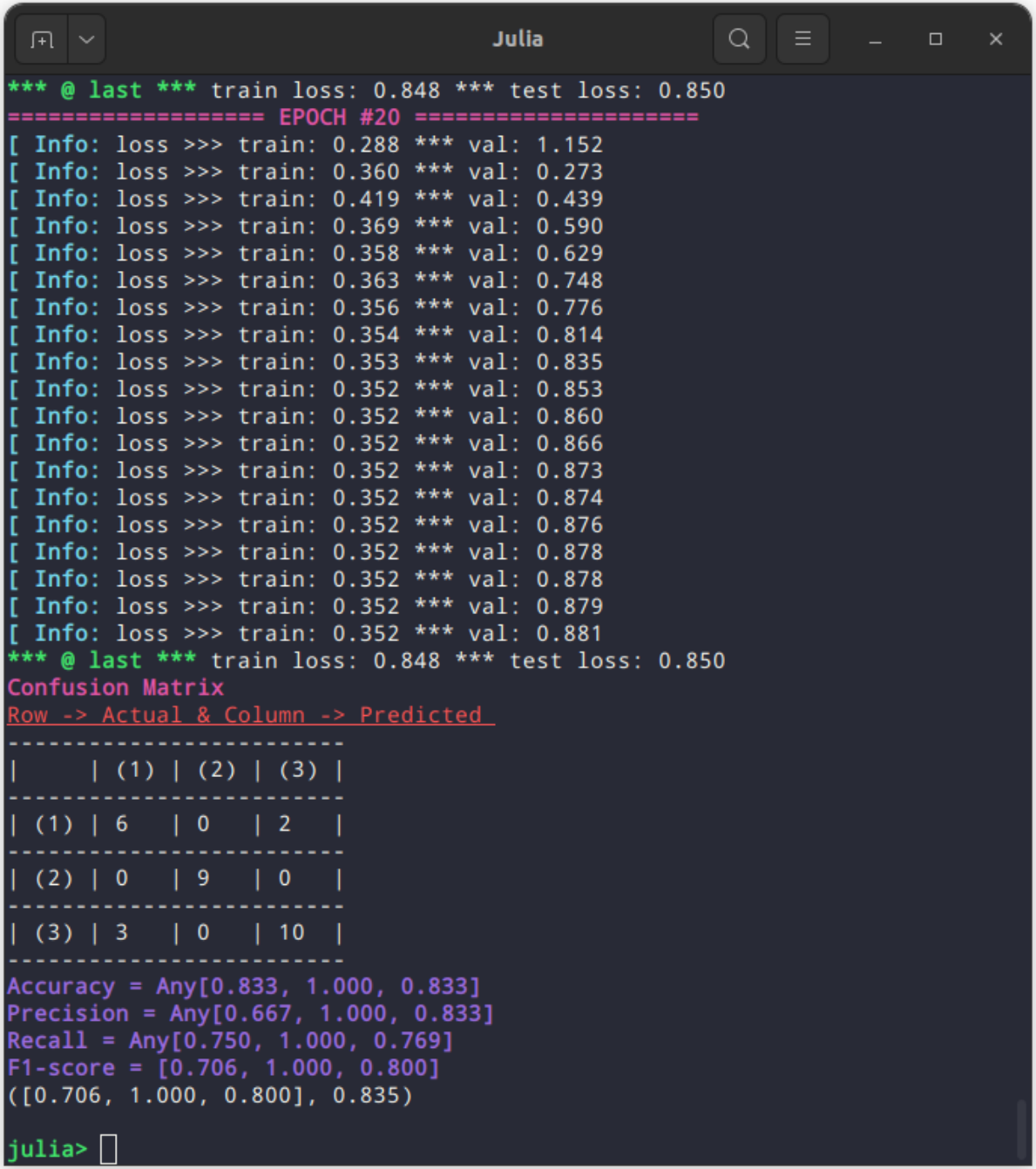
Code Snippet

```
if (isa(reg.method, Symbol) && reg.method == :l1) \
  || (isa(reg.method, String) && lowercase(reg.method) == "l1") # LASSO
    for (ix, l) in enumerate(layers)
        ∇W[ix] .+= reg.λ .* sign.(l.W)
    end
elseif (isa(reg.method, Symbol) && reg.method == :l2) \
  || (isa(reg.method, String) && lowercase(reg.method) == "l2") # RIDGE
    for (ix, l) in enumerate(layers)
        ∇W[ix] .+= reg.λ .* l.W
    end
elseif (isa(reg.method, Symbol) && reg.method == :elasticnet) \
  || (isa(reg.method, String) && lowercase(reg.method) == "elasticnet")
    for (ix, l) in enumerate(layers)
        ∇W[ix] .+= reg.r * reg.λ .* sign.(l.W) .+ (1-reg.r) * reg.λ .* l.W
    end
end
layers[i].W -= solver.η .* ∇W[i]
layers[i].b -= solver.η .* ∇b[i]
```

Numerical Experiments


Julia
julia> versioninfo()
Julia Version 1.10.4
Commit 48d4fd48430 (2024-06-04 10:41 UTC)
Build Info:
  Official https://julialang.org/ release
Platform Info:
  OS: Linux (x86_64-linux-gnu)
  CPU: 8 × Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-15.0.7 (ORCJIT, skylake)
Threads: 1 default, 0 interactive, 1 GC (on 8 virtual cores)
Environment:
  DYLD_LIBRARY_PATH = /home/mhamdi/torch/install/lib:
  LD_LIBRARY_PATH = /home/mhamdi/torch/install/lib:
(neural-network-from-scratch...) pkg> st
Status `~/Work/git-repos/AI-ML-DL/neural-network-from-scratch-in-Julia/Project.toml`
  [91a5bcdd] Plots v1.40.5
  [ce6b1742] RDatasets v0.7.7
  [295af30f] Revise v3.5.18
(neural-network-from-scratch...) pkg>

```julia
for epoch in 1:hp.epochs
    for (data_in, data_out) in zip(data_x, data_y)
        TrainNN(model, data_in, data_out, x_val, y_val; solver)
    end
    ### TRAIN LOSS
    ŷ_train = Predict(model, x_train)
    loss = loss_fct(y_train, ŷ_train; loss=solver.loss)
    push!(ltrn, loss)
    ### TEST LOSS
    ŷ_test = Predict(model, x_test)
    loss = loss_fct(y_test, ŷ_test; loss=solver.loss)
    push!(ltst, loss)
end
ŷ_tst = Predict(model, x_test)
ŷ_tst = Int.(ŷ_tst .== maximum(ŷ_tst, dims=2))
cm(y_test, ŷ_tst), accuracy_score(y_test, ŷ_tst), f1_score(y_test, ŷ_tst);
```


Julia
*** @ last *** train loss: 0.848 *** test loss: 0.850
================== EPOCH #20 ====================
[Info: loss >>> train: 0.288 *** val: 1.152
[Info: loss >>> train: 0.360 *** val: 0.273
[Info: loss >>> train: 0.419 *** val: 0.439
[Info: loss >>> train: 0.369 *** val: 0.590
[Info: loss >>> train: 0.358 *** val: 0.629
[Info: loss >>> train: 0.363 *** val: 0.748
[Info: loss >>> train: 0.356 *** val: 0.776
[Info: loss >>> train: 0.354 *** val: 0.814
[Info: loss >>> train: 0.353 *** val: 0.835
[Info: loss >>> train: 0.352 *** val: 0.853
[Info: loss >>> train: 0.352 *** val: 0.860
[Info: loss >>> train: 0.352 *** val: 0.866
[Info: loss >>> train: 0.352 *** val: 0.873
[Info: loss >>> train: 0.352 *** val: 0.874
[Info: loss >>> train: 0.352 *** val: 0.876
[Info: loss >>> train: 0.352 *** val: 0.878
[Info: loss >>> train: 0.352 *** val: 0.878
[Info: loss >>> train: 0.352 *** val: 0.879
[Info: loss >>> train: 0.352 *** val: 0.881
*** @ last *** train loss: 0.848 *** test loss: 0.850
Confusion Matrix
Row -> Actual & Column -> Predicted
	(1)	(2)	(3)
(1)	6	0	2
(2)	0	9	0
(3)	3	0	10
Accuracy = Any[0.833, 1.000, 0.833]
Precision = Any[0.667, 1.000, 0.833]
Recall = Any[0.750, 1.000, 0.769]
F1-score = [0.706, 1.000, 0.800]
([0.706, 1.000, 0.800], 0.835)
julia>

In Closing

Summary

- ✓ Valuable insights into the artificial neural network architecture;
- ✓ Practical tool for learning ANN;
- ✓ Intuitive and easy to use.

- ▶ Parallelization on batch of data;
- ▶ Optimizers: `SGD-Momentum`, `SGD-Nesterov`, `RMSprop`, `Adam`, Adagrad and `Adadelta`.

Summary

- ✓ Valuable insights into the artificial neural network architecture;
- ✓ Practical tool for learning ANN;
- ✓ Intuitive and easy to use.

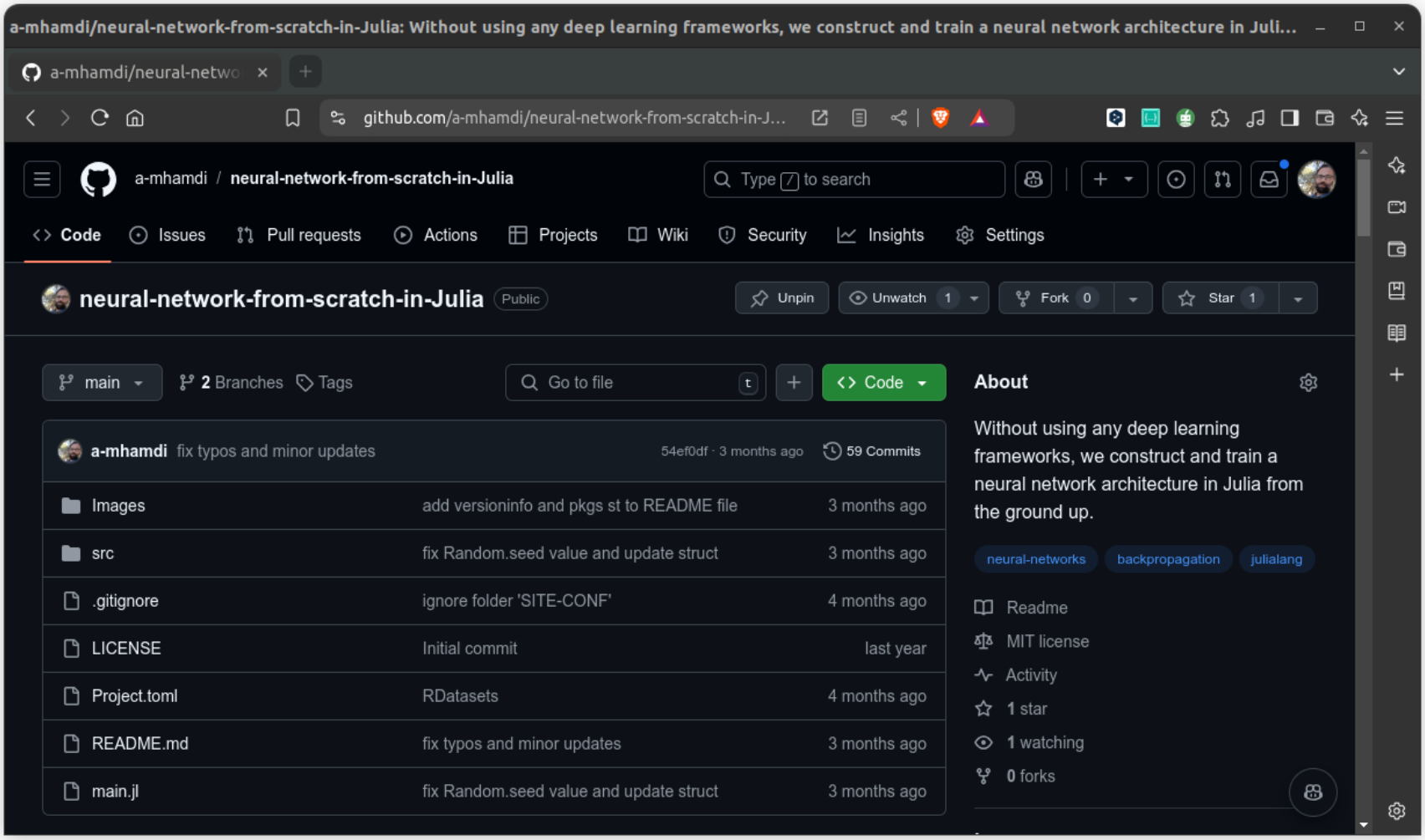
- ▶ Parallelization on batch of data;
- ▶ Optimizers: `SGD-Momentum`, `SGD-Nesterov`, `RMSprop`, `Adam`, Adagrad and `Adadelta`.

Summary

- ✓ Valuable insights into the artificial neural network architecture;
- ✓ Practical tool for learning ANN;
- ✓ Intuitive and easy to use.

- ▶ Parallelization on batch of data;
- ▶ Optimizers: `SGD-Momentum`, `SGD-Nesterov`, `RMSprop`, `Adam`, Adagrad and `Adadelta`.

Neural Network from scratch



https://github.com/a-mhamdi/neural-network-from-scratch-in-Julia

Thanks for your attention

Understanding Neural Networks: A From Scratch Implementation in Julia

2nd International Conference on Smart Industry, Technology and Environment

---

Abdelbacet Mhamdi
Dr.-Ing. in EE – MT @ ISET Bizerte

Presented on the 9th of November 2024 at Hammamet