"

Training neural ODE

Neurales ODE in Anlehnung an eine einfache differentielle Gleichung (ODE). Wobei die differenzelle Änderungen in einem System beschreibt. Durch Eingabe der Startwerte kann so der tatsächliche Graph abgebildert werden. Anstelle eines Solvers, wo einzelne Parameter des ODE angepasst werden, soll ein neurales Netzwerk trainiert werden, dass die ODE abbildet.

Training eines NN (Dense).

1. 2->32, Aktivierungsfunktion tanh,
2. 32->2, Aktivierungsfunktion linear.

Eingabe in das Training: Zeitpunkte zu denen Gemessene Punkte mit trainierten verglichen werden sollen.

Die zwei Knoten in Eingabe und Ausgabe Layer entsprechen den Temperaturen an den gegeben Temperatursensoren des tclab, während die erste Heizquelle zyklisch aufgeheizt wird. Das heißt, beide Sensoren werden zyklisch erhitzt. Sensor 1, stärker als Sensor 2. Sensor 2 ist im Aufheizen und Abkühlen im Vergleich zu Sensor 1 leicht verzögert.

Es soll ein neurales ODE mit dem Netzwerk angelernt werden. Das Netz soll die Änderungen des Verlauf des Temperaturen Vorhersagen.

Für die Vorhersage wird daher die Starttemperatur benötigt.

```
begin
    using CSV
    using DiffEqFlux
    using Flux: train!, Chain, Dense
    using Flux
    using DataInterpolations
    using DataFrames
    using OrdinaryDiffEq
    using Plots
    using PlutoUI
    using BenchmarkTools
    using JLD2
end
```

Table of Contents


```
celsius2kelvin (generic function with 1 method)
```

```
celsius2kelvin(c) = c + 273.15
```

```
prepare_dataframe (generic function with 1 method)
```

```
function prepare_dataframe(data_frame_path)

    _df = DataFrame(CSV.File(data_frame_path))

    start_times_stat = unique(_df, :run)

    _df.start_time =  start_times_stat[trunc.(Int, _df.run).+1, :time]
    _df.duration = _df.time - _df.start_time
    _df.T1 = _df.T1 .|> celsius2kelvin
    _df.T2 = _df.T2 .|> celsius2kelvin
    _df.Column1 = _df.Column1 .+ 1
    _df.run = _df.run .+ 1

    return _df
end
```

	Column1	T1	T2	Q1	time	run	start_time	duration
1	1	294.693	295.499	100.0	1.68496e9	1.0	1.68496e9	0.0
2	2	294.693	295.338	100.0	1.68496e9	1.0	1.68496e9	1.35462
3	3	294.693	295.466	100.0	1.68496e9	1.0	1.68496e9	2.5741
4	4	294.693	295.466	100.0	1.68496e9	1.0	1.68496e9	3.794
5	5	294.693	295.466	100.0	1.68496e9	1.0	1.68496e9	5.01308
6	6	294.693	295.563	100.0	1.68496e9	1.0	1.68496e9	6.23228
7	7	295.015	295.563	100.0	1.68496e9	1.0	1.68496e9	7.45149
8	8	295.015	295.531	100.0	1.68496e9	1.0	1.68496e9	8.67103
9	9	295.338	295.563	100.0	1.68496e9	1.0	1.68496e9	9.8905
10	10	295.338	295.531	100.0	1.68496e9	1.0	1.68496e9	11.1097
more								
8640	8640	298.238	299.527	0.0	1.68497e9	12.0	1.68496e9	868.37

```
begin
    heating_df_path = joinpath(@__DIR__, "measurements_heating_and_cooling.csv")
    heating_df = prepare_dataframe(heating_df_path)
end
```

```
[295.499, 295.338, 295.466, 295.466, 295.466, 295.563, 295.563, 295.531, 295.563, 295.
```

```
begin
    # define data for the ODE

    end_point = 3600;
    tspan = (0.0, end_point);
    tsteps = range(tspan[1], tspan[2], length=80);

    # seconds since start
    t = heating_df[1:end_point, :time] .- heating_df.start_time[1];
    u = heating_df[1:end_point, :Q1];
    T₁ = heating_df[1:end_point, :T1];
    T₂ = heating_df[1:end_point, :T2];
end
```

([294.693, 305.386, 319.313, 329.699, 336.992, 342.39, 346.273, 349.113, 351.277, m

```
begin
    # Interpoliere the u, y
    u_t = ConstantInterpolation(u, t);
    T₁_t = LinearInterpolation(T₁, t);
    T₂_t = LinearInterpolation(T₂, t);
    T0 = [T₁_t(0),T₂_t(0)]

    ode_data = Array(T₁_t(tsteps)), Array(T₂_t(tsteps));
end
```

loss_n_ode (generic function with 1 method)

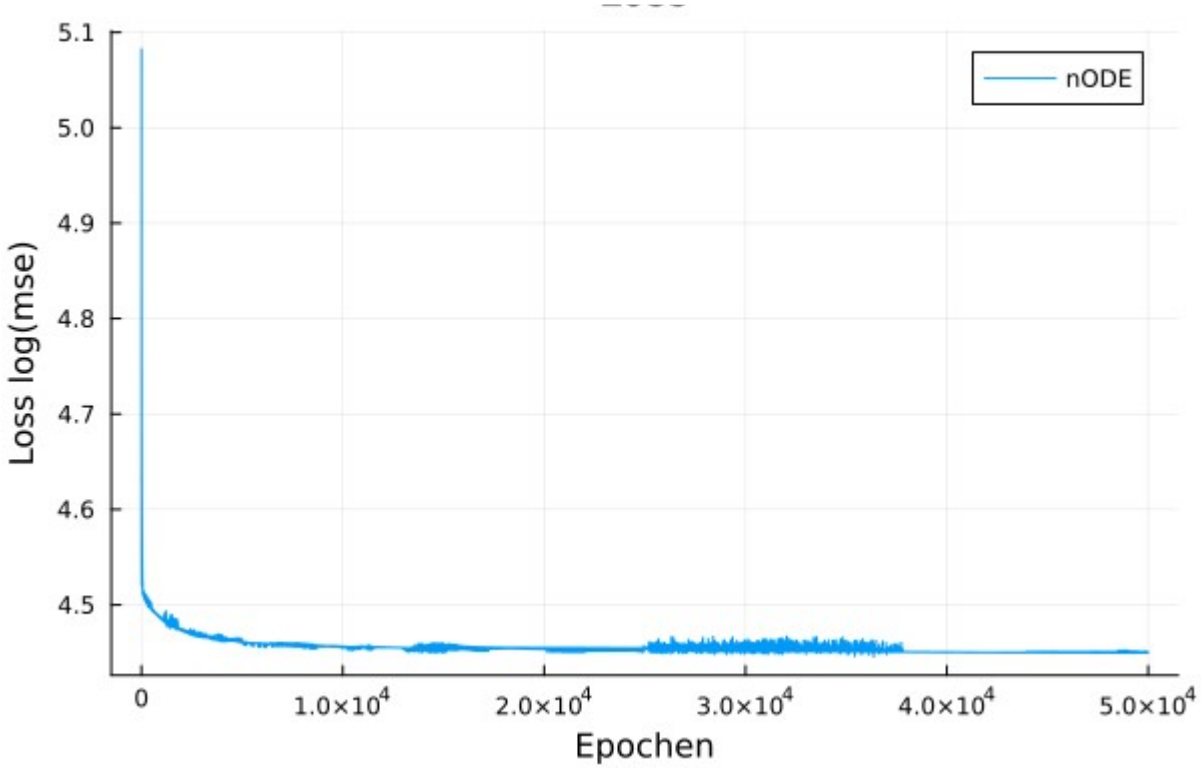
```
begin
    # durch Interpolation langsam im Training
    dudt = Flux.Chain(# x -> u_t.(x),  # wie Verallgemeinderung auf andere u_t
                      Flux.Dense(2=>32, tanh),
                      Flux.Dense(32=>2)) |> f64

    n_ode = NeuralODE(dudt, tspan, Tsit5(), saveat=tsteps, reltol=1e-3,
    abstol=1e-5)

    y_predict_before = n_ode(T0)
    ps = Flux.params(n_ode.p)

    function predict_n_ode()
          n_ode(T0)
    end

    loss_n_ode() = sum(abs2,ode_data[1] .- n_ode(T0)'[:, 1]) +
    sum(abs2,ode_data[2] .- n_ode(T0)'[:, 2])


end
```

```
begin

    η = 1e-4
    opt_n_ode = ADAM(η, (0.9, 0.8))

    loss_vector = Vector{Float64}()
    callback() = push!(loss_vector, loss_n_ode())

    epochs1 = 500
    data1 = Iterators.repeated((), epochs1)

    epochs2 = 4_500
    data2 = Iterators.repeated((), epochs2)

    epochs3 = 45_000
    data3 = Iterators.repeated((), epochs3)

    epochs = epochs1 + epochs2 + epochs3

    Flux.train!(loss_n_ode, ps, data1, opt_n_ode, cb=callback)
    opt_n_ode.eta = 5e-5
    Flux.train!(loss_n_ode, ps, data2, opt_n_ode, cb=callback)
    opt_n_ode.eta = 1e-5
    Flux.train!(loss_n_ode, ps, data3, opt_n_ode, cb=callback)
end
```

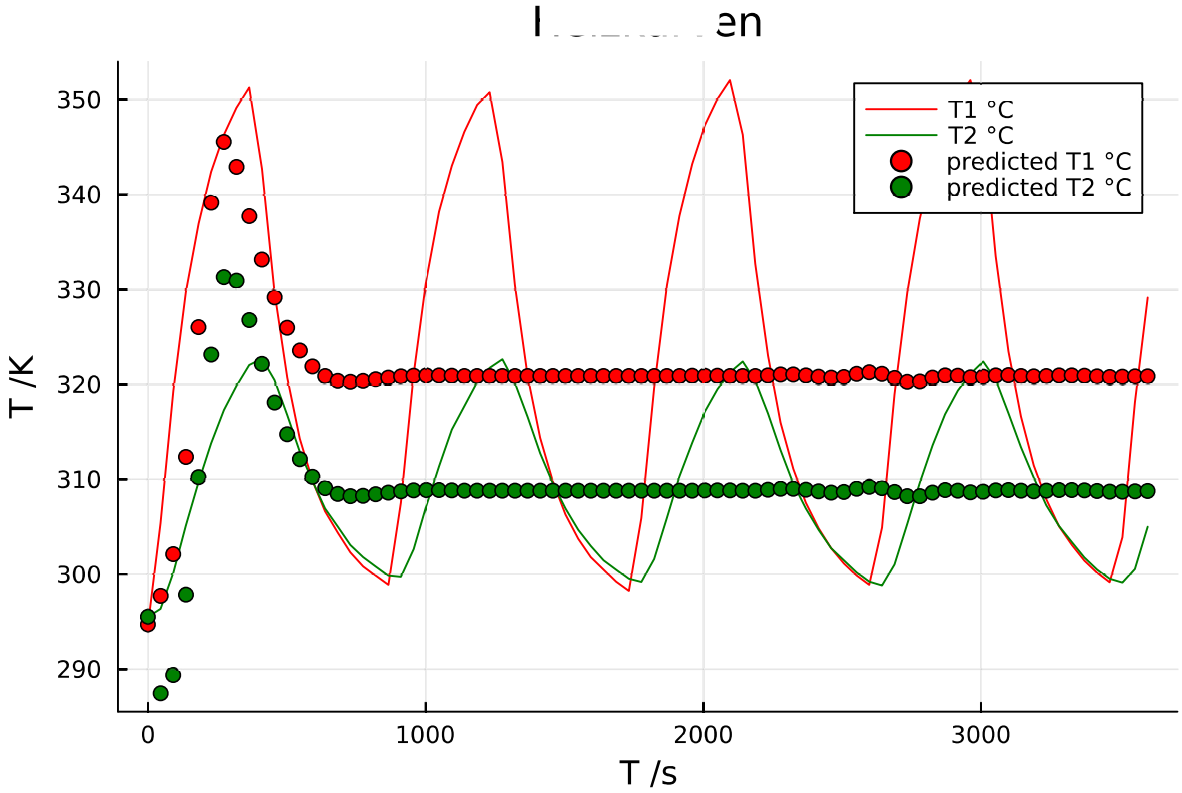
100%

100%

100%




```
plot(1:epochs, loss_vector .|> log10,
    label="nODE", title="Loss",
    xlabel = "Epochen", ylabel = "Loss log(mse)",)
```




```
begin
    y_predict_after = n_ode(T0)

    # plot result
    p_node = plot(tsteps, ode_data[1], label="T1 °C", lc=:red, xlabel = "T /s",
    ylabel = raw"T /K",  title="Heizkurven")
    plot!(p_node, tsteps, ode_data[2], label="T2 °C", lc=:green)
    scatter!(p_node, tsteps, y_predict_after[1, :], label="predicted T1 °C",
    mc=:red)
    scatter!(p_node, tsteps, y_predict_after[2, :], label="predicted T2 °C",
    mc=:green)
end
```

Saving the Model

```
begin
    model_state = Flux.state(n_ode);
    jldsave("nODE_model.jld2"; n_ode)
end
```

Änderungen und Fazit

Der Loss nimmt bis zu 1000 Zyklen ab und stagniert hiernach.

Die Oszilation der Temperaturen beim Heizen und Abkühlen konnte nicht nachgebildet werden.

Der Prozentsatz der Intensität der Heizung wird, testweise mit Interpolation als erste Layer hinzugefügt. Hierfür wurde eine ConstantInterpolation gewählt, da diese leicht schneller ist, als die Linear und bei den Wertebereich [0, 100] geringere Fehler machen sollte.

Verwendung von zwei Temperaturen

Beim Training mit zwei Temperaturkurven ist der Verlauf des Losses glatter, als wenn nur ein Sensor angelernt wird.

Tolleranz vs Rechenzeit

Es wurden drei Parametersätze für die Auflösung des ODE getestet.

reltol	abstol	Rechenzeit pro Epoche \s
1e-3	1e-5	1.94
1e-5	1e-7	2.83
1e-7	1e-9	7.92

Der Verlauf Prognose ist zwischen den Parametersätzen nicht wirklich unterschiedlich. Zusätzlich wurde eine Lauf mit einer relativen Toleranz von 1e-2 durchgeführt, hier hatte das System den Anschein, als ob zu schwingen beginnen würde. Allerdings kann es sich hier um ein Artefakt der hohen erlaubten Toleranz handeln, zusätlich war der aufgezeichnete Loss sehr verrauscht, eine Optimierung gelang hier nicht. Der Test mit der Toleranz von 1e-3 ein Optimum zu finden, wo das Rauschen verringt und dennoch Schwingung erhalten bleibt war nicht erfolgreich. Der Unterschied in den reichenzeit sind bei hoher und Mittlerer Toleranz nicht wesentlich im Vergleich zur niedrigen Toleranz erst hier werden anscheinend wesentlich mehr Zwischenschritte für die Berechnung benötigt. Die Rechenzeit bezoglich auf zwei Layer mit 32 versteckten Knoten mit einem AMD Ryzen 5800X bei 120 Messpunkten. Aktuell wird zur Beschleunigung mit 80 Punkten gerechnet.

Optimierer

Tsit5 gegen DP5 getauscht. Der Verlauf Prognose ist zwischen den beiden Optimierern nicht wesentlich unterschiedlich. Tsit5 war aber bei reltol=1e-3 , abstol=1e-5 mit einer Rechenzeit von 2.5 s pro Epoche langsamer.

Anzahl der Punkte

Die Anzahl festgelegten Zwischenschritte (tsave) wurde erhöht, damit mehr Trainingsbeispiele

des Ausgabevektor des Netzes mit (interpolierten) Messwerten verglichen werden können, um so die Anzahl der Beispiele zu erhöhen. Die Anzahl wurde erst verdoppelt und dann verdreifacht. Dies führe jedoch nicht zu besseren Optimierungen. Daher wurde die Anzahl wieder auf Aktuell 80 Punkte reduziert.

Anzahl der Heizzyklen

Desweiteren wurde getestet, ob mehr Heizzyklen als Beispiel in das Training überführt werden können. Dies führte zu keiner verbessung der Optimierung.

Optimierung des Aufgezeichneten Loss

Die Funktion aufgezeichnete Loss war zum Teil sehr rauch. Es wurde vermutet, dieses dies insbesondere, dass weitere lernen verhindert. Deswegen unterschiedliche Schritte mit unterschiedlichen Lernraten getestet. Niedrige Lernraten führten zwar zur klatteren Kurven, aber nicht zur besseren Optimierungen.

Variation der Hiddenlayer und Anzahl der Knoten

Es wurde regelmäßig die Anzahl Layer von zwei bis zu fünf getestet. Gleichzeigig wurde die Anzahl der Knoten von 16, 32, 64 geprüft. Beide Veränderungen führen zu kleiner wesentlichen Verbessung im Loss. Das Modell fängt nicht an zu schwingen.

Erhöhung der Anzahl der Epochen

Durch all die Optimierungen der Rechenzeit und der Reduzierung des Rauschens im Loss gelang, dass bei der Erhöhung der Anzahl der Epochen ein geringer Fortschritt beim Lernen möglich wurde. Nun gelingt es durch die Erhöhung der Epochen von *1000 auf 50000* für die erste Heizphase eine Abweichung von der Mitteltemperatur zu erreichen.