Learning Stiff Atmospheric Chemical Kinetics: The Regional Atmospheric Chemistry Mechanism (RACM)

Levin Rug
Leibniz-Institute for Tropospheric Research, Leipzig, Germany
rug@tropos.de

What's the matter?

We want to train a neural network to predict the concentrations of chemical compounds in the atmosphere.

Chemical Mechanism (Example: Robertson Mechanism)

$$A \xrightarrow{k_1} B \qquad k_1 = 4 \times 10^{-2}$$
$$2B \xrightarrow{k_2} B + C \qquad k_2 = 3 \times 10^{7}$$
$$B + C \xrightarrow{k_3} A + C \qquad k_3 = 1 \times 10^{4}$$

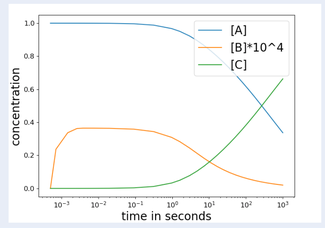
Chemical Mechanisms of the atmosphere describe, how the oxidation of Volatile Organic Compounds (VOCs) and their interaction with NO_x produces the air pollutant ozone, but also higher oxidised VOCs that are important for production of aerosol particle mass. Being able to consider accurate information about aerosols and chemical processes in the troposphere has direct impact on the quality of weather, climate and air quality predictions and research. While solvers remain slow, accurate mechanisms consistently gain in number of species and reactions (like the Master Chemical Mechanism: 6700 species, 17000 reactions, http://chmlin9.leeds.ac.uk/MCM/project.htt).

System of Ordinary Differential Equations

$$\frac{\partial}{\partial t}[A] = -k_1[A] + k_3[B][C]$$
$$\frac{\partial}{\partial t}[B] = k_1[A] - k_2[B]^2 - k_3[B][C]$$
$$\frac{\partial}{\partial t}[C] = k_2[B]^2$$

By simple transformations, we can write down a system of ordinary differential equations (ODEs) representing the mechanism. If we find a solution to this system, given initial conditions like measurements of chemical compounds in the troposphere, we can calculate future atmospheric conditions, which is done numerically in models. These equations are non-linear, with polynomial right hand sides. Together with reaction rates ($k_i, i = 1, 2, 3$) and concentrations in multiple magnitudes, this becomes a stiff system. In order to achieve accurate, stable solutions, we need to apply implicit solvers like Rosenbrock methods or LSODE. Those are sophisticated and efficient solvers, but still slow. They cause the chemistry bottleneck in big models.

Trajectories of Predicted Concentrations



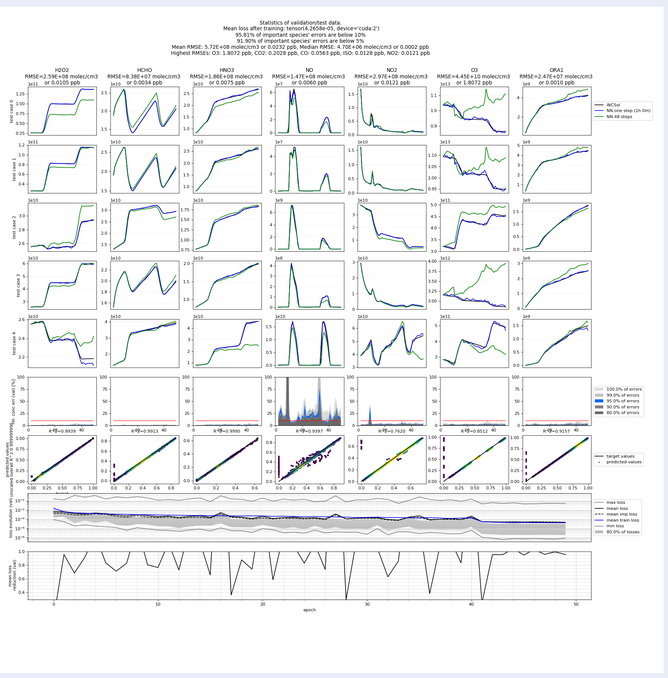
B has its relevant behaviour in a time span of the first milliseconds, while A and C change in time spans of tens to hundreds of seconds. This is hard for both numerical solvers, as the step size has to be chosen very carefully, and for neural networks, because they have to treat all of those species, where small absolute errors can have huge impact on the solution, also regarding the chaotic nature of atmospheric processes. Our neural network is supposed to recreate very similar trajectories, by taking as input concentrations and meteorological data and giving as output the next time step of concentrations, i.e. exactly what a numerical integrator does. But, neural networks can be evaluated much faster.

The Network

Training

Training a neural network in our case can achieve some satisfying accuracy for single time steps if it is trained long enough. More complicated is the long-term prediction performance, i.e. when erroneous output is fed through the network again and again. In practice, this is important, because we don't want to look one time step into the future, but hours or days or even more. Below, performance of single time steps is indicated by blue lines, long-term predictions by green lines.

Input: vector of concentrations at a time (C(t)), daytime and vector of emissions (E(t)) → 95+1+6=102 values
Output: vector of concentrations at next time: $C_{t+\Delta t}$ → 95 values
Loss Function: Mean Squared Error, i.e. $||C(t + \Delta t) - C_{t+\Delta t}||^2$
Optimizer: Stochastic Gradient Descent (batch size = 1)
Network Size: 3 Layers á 3000 neurons
Data Set Size: 36000 hourly concentrations of 750 simulations with individual initial concentrations and emission profiles
Training Time: 1h 14min on RTX8000 GPU (48GB memory)

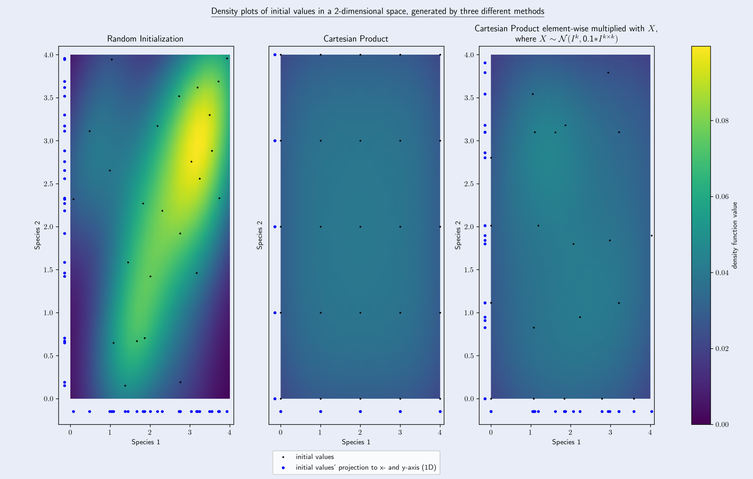


The above figure shows several evaluations of the trained network.
The top five rows show test data, i.e. data which was never used in training. We see some important species' trajectories for each of the test simulations. Blue lines ("NN hourly") indicate, what the neural network predicts, if the previous time step is fed into the network. Green lines ("NN full") show the result for feeding the initial condition into the network and then put the output into the network again 47 times. We get more or less accurate 48 hour predictions from the neural network. Black lines are the reference solutions.
The sixth row displays relative error of the above species in dependence of daytime. Red lines indicate 10% relative error.
The seventh row has a plot of all values occuring in the validation data set for each species and their predictions, with color indicating density of values from blue=low density to yellow=high density.
The eighth and ninth row show how some error measures decreased during training. The jagged line in row nine is an indicator for the low-regularity behaviour of our loss function.

The Data

The Curse of Dimensionality

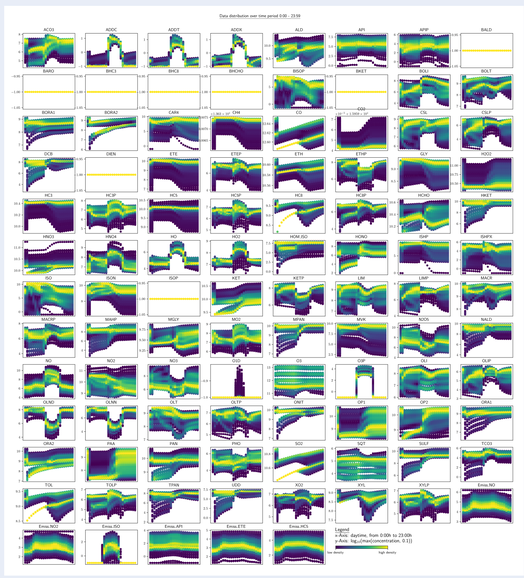
The most common way to generate data is random sampling. Enough samples form a good covering of the input space and followingly, the network will be trained on most of the possible scenarios.
But, a sufficiently large number of samples is needed. Otherwise there are random gaps, where the network is doomed to perform more poorly than elsewhere. So, what is a sufficiently large number?
This is no trivial question, but simplified, we can assume to have a similar absolute amount of prediction quality when we have n input values sampled randomly as we have when we choose to create a grid of n points as input values, say a cartesian product. (Note, that the random samples are better for large n, because the prediction quality is more distributed.)
The curse of dimensionality strikes now: A fine grid would be the best to cover the space thoroughly. But, only imagining an n-dimensional cube, i.e. a cartesian product of 2 values per species, we would have to train on $2^{\text{dim(input)}} = 2^{102} \approx 5 \times 10^{30}$ values. Beside that being completely out of reach, a cube is still an extremely poor grid! Random samples are simply infeasible, because a random sampled equivalent of a cube always has huge, random gaps in its space.

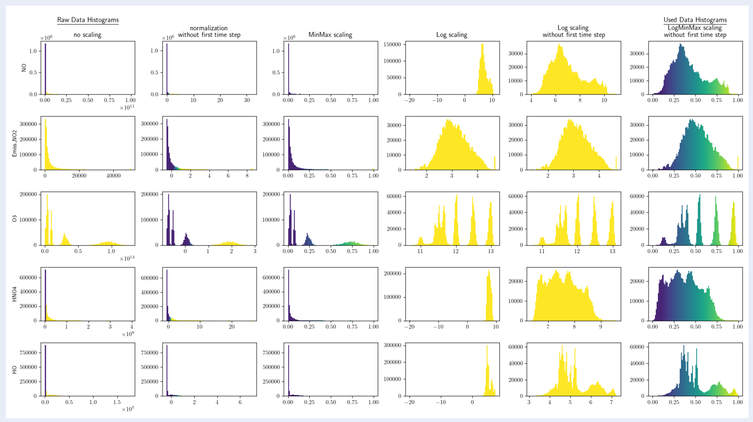


The left hand side figure should work as an example of the curse of dimensionality. We see three methods of creating samples in a 2-dimensional space: completely random, uniform grid and distorted uniform grid. The coloured background indicates density of the tuples, which we want to be constant at best. For a low number of samples, the second method performs best. The reason why it should still be avoided can be seen in the projections to one dimension (1D), the blue dots outside the color plot: the whole training would only provide 5 seperate values per species! The network is supposed to learn the complete behaviour, not 5 values over and over. In 1D projections, random looks best, but it is infeasible - the third method is a good combination, taking advantage of both randomness and structure.

Individual Data

The data for each problem is obviously very individual. Since we aim to use the network for atmospheric conditions, we can further individualize the data up to daytime-specific ranges for every single species. This is beneficial, since it tightens the, as outlined above, exceedingly large input space. Additionally, training on irrelevant data is prohibitive in multiple ways.
Luckily, we can use the mechanism to tell us the specific ranges for the species. We vary the most valuable influences on the system, i.e. emissions and our observable O_3 and in this way, possible and relevant simulations are generated. The right hand side figure shows the specific ranges for each species in dependence of daytime, with colour depending on density of values.





The last step until our data is ready to train the network successfully is scaling. As mentioned before, the individual species can have concentrations in very different magnitudes and also, some change their magnitude very rapidly. But still, every little change in concentration can be of arbitrary importance for the stiff system. A neural network has a hard time learning one input value being sensitive in regions of 10^{13} and the other in 10^3 and the next from 10^6 to 10^{14}, especially when the species' behaviour is indeed logarithmic. The figure to the left shows six common ways of scaling data. The plots are histograms of the whole data generated for various species, colour indicates magnitude of values. The rightmost column shows, in comparison, very similarly distributed data for all species, which is why we use this way of scaling.