Building a Deep Learning Model for Atmospheric Transport of Gases

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Garvit Agarwal

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

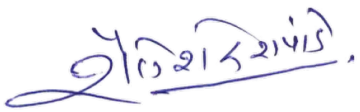
May, 2024

Supervisor: Dr. Shailesh Deshpande
© Garvit Agarwal 2024

All rights reserved

Certificate

This is to certify that this dissertation entitled **Building a Deep Learning Model for Atmospheric Transport of Gases** towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Garvit Agarwal at Tata Consultancy Services (TCS) Research (TRDDC), Pune under the supervision of Dr. Shailesh Deshpande, Principal Scientist, during the academic year 2023-2024.



Dr. Shailesh Deshpande

Committee:

Dr. Shailesh Deshpande

Prof Amit Apte

This thesis is dedicated to my sister

Declaration

I hereby declare that the matter embodied in the report entitled **Building a Deep Learning Model for Atmospheric Transport of Gases** are the results of the work carried out by me at Tata Consultancy Services (TCS) Research, Pune under the supervision of Dr. Shailesh Deshpande , and the same has not been submitted elsewhere for any other degree. Wherever others contribute, every effort is made to indicate this clearly, with due reference to the literature and acknowledgement of collaborative research and discussions.

Garvit Agarwal

IISER Roll no: 20191041

Date: 14th March 2024

Acknowledgments

First and foremost, I would like to express my gratitude towards the Career Development Cell (CDC) at IISER Pune, for making the opportunity to work in an industrial environment available to me.

I am most grateful to my supervisor, Dr. Shailesh Deshpande at TCS Research, Pune who offered me the freedom and space to lead the work, make mistakes, learn from them and navigate my way around constraints. Thanks to the colleagues in the research group, I had a comfortable experience working at TCS.

I would also like to thank Prof Amit Apte for being the expert for my thesis and for his suggestions during the mid-year and final evaluation.
I also appreciate all the discussions I had with Soorya Narayan about deep learning and the challenges I faced.
Finally, I would like to thank my friends and family for being a big source of motivation.

Abstract

Obtaining accurate estimates of the distribution of surface emissions of gases like CO_2, NO_2, CO, etc is a technically challenging as well as environmentally relevant problem. Many data assimilation-based algorithms have been developed to tackle this problem over decades. While these algorithms have their own strengths, they also face some prohibitive challenges, like high computational cost, deep mathematical and computational expertise, and errors arising from various numerical simplifications and parameterizations. The rapidly evolving field of deep learning can help address these limitations due to their GPU-powered fast computations and relative ease of application. In this work, we propose a deep learning algorithm to estimate the emission distribution of a given gas, using future information of its concentrations in the atmosphere. The algorithm involves building a deep learning surrogate of an existing numerical transport model and then using the automatic differentiability of neural networks to perform emission estimation. This thesis focuses on the first part, ie building a deep learning model that is trained to predict future concentrations of a gas (CO_2) given its initial concentrations, the geographical distribution of its emissions/sinks, and the relevant meteorological conditions (eg winds, pressure etc). The numerous model-related choices are described, various alternatives are tested and the model with relatively the best performance is analysed. Our model learns atmospheric transport to a significant degree, having an average error in predicted concentrations of about 0.104 ppm. This thesis is based on the work done at TCS Research, Pune.

Contents

Chapter 1

Introduction

Two of the major problems of the 21st century, air pollution and global warming, are the results of the excess emissions of certain harmful gases, like CO_2, CH_4, CO, SO_2, NO and NO_2 into the atmosphere due to human activities. International treaties and agreements have been established that aim to set emission reduction targets for countries, like the Kyoto Protocol [1] and the Paris Agreement [2] in the context of global warming. Hence, to assess the effectiveness as well as the adherence by countries to these policies, governments and industries alike are very interested in estimating the distribution of emissions[1] of these gases on the surface of the Earth.

To do this, broadly there are two classes of methods [3]. One of them, called **bottom-up** methods involve gathering detailed data on individual sources, activities, and technologies to estimate the emissions at different locations. These methods rely on:

- comprehensive inventory of sources from a wide range, including industrial facilities, power plants, transportation, agriculture, and residential activities.
- the associated emission factors (coefficients that relate the quantity of pollutants/-gas emitted to a specific activity level) determined through field measurements, laboratory experiments, or published literature.
- activity data, that represents the level of a particular activity that is associated with pollutant or greenhouse gas emissions

[1] in this document emissions imply the fluxes of a gas from the surface in mass(kg) per unit area (m^2) per unit time (sec)

These methods can provide detailed information, sector-specific insights and granular understanding of the emissions. However, collecting and processing detailed data from various sources can be a complex and costly undertaking; on the other hand they may not capture all sources of emissions. There can also be potentially high uncertainties associated with emission factors or activity data. There is heavy dependence on assumptions, and crucially, they have limited applicability in remote areas or developing countries where data collection infrastructure may be lacking.

The other class of methods, called **top-down** methods involve measuring concentrations of the gas/pollutant in the atmosphere and inferring the sources of those concentrations with the help of *transport models*. A large number of atmospheric transport and chemistry models have been developed; these models predict the future concentrations of a gas/pollutant species given its past surface emission distribution, past concentrations and meteorological conditions. They model various processes like advection, convection, planetary boundary layer turbulence, surface deposition, cloud convection, chemical reactions, etc numerically. Using certain data assimilation algorithms, it is possible to 'invert' the transport models, ie estimate the emission distribution using measurements of atmospheric concentrations of the gas. These methods can exploit the global coverage granted when we use satellites, they takes into account all sources of emission, known or unknown, and they open the possibility of near-real-time monitoring.

Emission estimation using Transport Models

Transport models take into account various non-linear physical processes some of them governed by well-known equations so numerical methods have been used to implement them for decades. The goal is to essentially estimate the input of these models (emissions) given the output (future concentrations) but we cant directly achieve that, since there is no analytical solution to this problem when you take into account transport over large spatial and temporal scales. This problem has additional difficulties; the solution to the inverse problem is not unique (since the problem is mathematically underdetermined) and the measurements of concentrations that are used always have uncertainties. Hence we need to give a prior estimate of the emissions to deal with the non-uniqueness and the algorithm should be able to deal with the uncertainties. 2 major algorithms that have been widely used to achieve this are the Ensemble Kalman filter (EnKF) [4][5][6] and

the 4-D Variational (4D Var) method [7][8][9]. The idea behind both algorithms is to estimate a state vector using a time-series of observations. For our problem, the 'state' is the surface emission flux distribution.

1. Ensemble Kalman Filter (EnKF) [10]: The idea behind the Kalman filter is essentially repetitive application of a Bayesian update step, ie given a *prior* estimate of the 'state' and its uncertainties, it *updates* the prior (and its uncertainties) using the observations and the new estimate is called the *posterior*. The Kalman filter assumes that all probability distributions involved are Gaussian. This Bayesian update step is followed by advancing the model in time (using an atmospheric transport model), then incorporating new data in a Bayesian update step and so on. However, maintaining the covariance matrix is not feasible computationally for high-dimensional systems. For this reason, EnKFs were developed. EnKFs represent the distribution of the system state using a collection of state vectors, called an ensemble, and replace the covariance matrix with the sample covariance computed from the ensemble.

2. 4-D Variational method [11]: In this algorithm, the state is estimated by minimizing a cost function that is a function of the prior state, a transport model, and the observations within a fixed time-duration called the 'assimilation window'. Typically the cost function takes into account the state's deviation from the prior estimate as well as the deviation of the predicted concentrations based on the current state from new observations and these 2 terms are weighted by the uncertainties in the prior and the measurements respectively. The first term is to make sure that the state does not drift too far away from the prior that are known to usually be reliable as initial estimates. The minimization is done using a gradient descent algorithm.

The observational concentration data that are used for the above can come from measurements from ground-based stations or from satellites. In the latter case, we get total-column measurements of the concentrations through remote sensing. While ground-based measurements are cost-effective (for a relatively small area), can provide us high spatial and temporal resolution and are easier to calibrate and monitor, satellites offer a much superior spatial coverage providing us data from remote and inaccessible areas.

While the above methods have been applied to the emission estimation problem to yield good results, they face the following problems. Performing high-dimensional optimization (eg in 4D-Var) or running large ensembles of model simulations (eg in ensem-

ble Kalman filter) can be computationally expensive, especially when we need to work with fine resolution. Moreover these algorithms involve complex mathematical formulations and integrating them with atmospheric transport models involves coupling complex numerical codes. Implementing these algorithms requires a deep understanding of the underlying mathematics and computational methods, and hence poses a big barrier. Furthermore, atmospheric transport models contain uncertainties and errors arising from simplifications in model physics, parameterizations (of processes like surface-layer turbulence, chemistry etc), and numerical approximations. Assimilating observations of concentrations into such models may propagate and amplify these uncertainties, leading to inaccurate estimates of emission distributions.

Deep learning is finding a growing use in the atmospheric and geoscience community [12][13][14] owing to their demonstrated great ability in capturing non-linearities in a system (that are a major part of atmospheric transport and estimation of emissions). Deep learning models automatically learn relevant features and representations from the data, eliminating the need for knowing the dependence among inputs or other domain-specific knowledge. They are also adept at capturing complex spatio-temporal patterns, are known to be scalable, and are highly parallelizable using GPUs, making them much faster than the above data assimilation methods. Moreover their ability to capture complex patterns and ease of fine-tuning makes it possible for them to overcome the model errors and uncertainties of the numerical transport models.

Goal of the Project

In light of the above, the broader goal is to **use Deep Learning & an atmospheric transport model to implement an algorithm for the estimation/correction of emissions of a given gas using its future concentrations** .

Broadly the algorithm proposed in this work involves:

1. First training a Deep Learning surrogate/twin of the transport model that learns to emulate the latter's output, then
2. Keeping the surrogate fixed, iteratively update the prior emissions using the error between the predictions and the measurements of the concentrations of the gas.

This is an ambitious and long-term work being done at TCS Research, Pune and building an accurate deep learning surrogate will be strictly required for us to do reliable emission estimation. Hence in the current thesis we focus on-

Building a Deep Learning Surrogate of an atmospheric transport model that can predict future concentrations of a gas, given its initial concentrations, its emission information and relevant meteorological data.

We have used a grid of resolution $4^\circ \times 5^\circ$ (450 x 560 km) for a region of size about 7200 km $\times$ 9000 km over Asia, so the horizontal grid size is 16×16. For this work our transport surrogate model is trained to predict concentrations for CO_2 in particular. The choice was determined by i) the ease of availability of good quality concentration data (from satellites for example), (ii) the fact that CO_2 does not react chemically in the atmosphere, especially on the scale of a few days, and (iii) because of its huge relevance to global warming. Furthermore, we have trained the model to consider only wind-driven advection and ignore other processes (molecular diffusion is considered negligible because of *numerical diffusion*, see section 2.2). The requirement for these many simplifications is explained in section 4.1.

In section 2.1, we describe the process of choosing a good transport model for our work and details of the chosen model. That is followed by a short overview of the basic concepts in numerical transport modeling that are relevant for us (section 2.2). Then in chapter 3, we provide some basics of Deep Learning/Neural Networks (section 3.1) that are necessary to understand how we trained our transport surrogate. The methodology that we propose to perform emission correction is described in section 3.2. Next in chapter 4 we go into the details of the surrogate model we built and how we trained it, followed by analyses of its performance in chapter 5, where it performs well and where it doesnt etc. Finally, we will discuss how we can improve the surrogate's accuracy before we can perform emission correction.

Chapter 2

Transport Model

As mentioned earlier, many atmospheric transport models have been developed by different research organisations across the globe. Since the broad goal and physics is the same, all of these models are similar when it comes to the input data they take and their output. They might differ in the exact numerical schemes they use to model various processes like advection, convection, planetary boundary layer turbulence etc. More importantly for us, some of these models are easier to implement than the others, depending on how customizable they are, how good their documentation is, whether they have good user support etc.

Based on these factors we performed a literature survey of 10 some of the most well-known transport models [15][16][17][18][19][20][21][22][23] (for TM3, the description and documentation was obtained directly from the developers); they are presented in the table below.

Transport Model Name	Developers(s)
1) GEOS-Chem	Main Institutes: Harvard University, Washington University, NASA Earth Science Division etc.
2) CMAQ	US Environmental Protection Agency (EPA)
3) CAM-Chem	National Center for Atmospheric Research(NCAR), US
4) WRF-Chem	National Center for Atmospheric Research(NCAR), US
5) LMDz-GCM	Laboratoire de Météorologie Dynamique (LMD), France
6) TM3	Max Planck Institute for Biochemistry, Germany
7) FLEXPART	Norwegian Institute for Air Research (NILU)
8) NIES/FRCGC	NIES, FRCGC, Japan
9) CCAM	Centre for Australian Weather and Climate Research (CAWCR)
10) NICAM	University of Tokyo, the Japan Agency for Marine-Earth Science and Technology (JAMSTEC)

2.1 GEOS-Chem

Based on our survey, we chose the model '**GEOS-Chem**' for our project, since it is open-source with extensive online documentation, has a community on Github for any issues and discussion, and it is a very flexible and customizable model. It models various atmospheric processes, like advection, chemistry, aerosol microphysics, radiation, boundary layer turbulence, dry & wet convection and deposition.

It can be used on a global scale or over user-defined regions with dynamic boundary conditions. GEOS-Chem comes in 2 variants: GCClassic and GCHP (GC High performance). GCClassic is the standard variant that uses a rectilinear latitude-longitude grid, but can run on a single node (upto about 30 cores), while GCHP uses a cube-sphere grid (a cube projected on the surface of the sphere) and is designed to work on multiple nodes, making it possible to use upto 500-1000 cores. In our work we have specifically used GCClassic and we interchangeably call it GEOS-Chem in this work.

INPUTS

Below are the inputs that GEOS-Chem takes for a tracer transport simulation:

1. Start and end dates of simulation
2. Grid resolution (deg x deg). Accepted values are: $4^\circ \times 5^\circ$ (450 x 560 km), $2^\circ \times 2.5^\circ$ (225 x 280 km), $0.5^\circ \times 0.625^\circ$ (56 x 70 km) and $0.25^\circ \times 0.3125^\circ$ (28 x 35 km).
3. No of vertical levels (72 or 47). The topmost level in both cases is at 80 km height.
4. Longitude-latitude range (ie region of the simulation)
5. Transported species and its properties relevant for the simulation, eg its chemical formula and reactions etc.
6. Transport and chemistry timesteps (seconds)
7. Emission/sink fields of tracer (in $kg/m^2/s$ or $kg/m^3/s$)
8. Initial concentrations of the tracer
9. Tracer concentration output frequency and averaging duration (hourly, daily, monthly etc)
10. Major meteorological fields:
    - eastward wind (3D, unit: m/s)
    - northward wind (3D, unit: m/s)
    - surface pressure (2D, unit: Pascal)
    - specific humidity (3D, unit: kg water vapour/kg air)
    - air temperature (3D, unit: K)
    - information about the geography/land-types etc.

OUTPUT

GEOS-Chem can provide the instantaneous/averaged concentration of the transported species/gas in *moles per mole of dry air* over the 3D grid with a time-frequency as defined by the user. It can also optionally provide the diagnostics for the atmospheric processes modeled.

2.2 Numerical Transport Modeling

In this section, we first present a short overview of how transport modeling is done numerically in general [24] and then look at the kind of numerical scheme that GEOS-Chem uses.

A transport model essentially has to solve one equation, that is the **continuity equation**, that expresses mass-conservation of a tracer species within a parcel of air:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) = s \tag{2.1}$$

where ρ is the mass density of the tracer, v is the wind velocity, and s is the net local source of the species. The second term in the LHS is called the *transport term*, the RHS is called the *local term* and this particular equation is called the **Eulerian flux form** of the continuity equation. The transport term accounts for transport of the tracer due to winds (advection), turbulence and convection, while the local term accounts for the contributions from chemistry, emissions, and deposition. Equation 2.1 can be written in terms of the **tracer mixing ratio** $\mu = \rho/\rho_a$, where ρ_a is the denstiy of air:

$$\frac{\partial \mu}{\partial t} + v \cdot \nabla \mu = \frac{s}{\rho_a} \tag{2.2}$$

this is called the **Eulerian advective form** of the continuity equation. These are called Eulerian forms because they are written from the frame of reference of the ground. The **Lagrangian form** of the equation which is based on the frame of reference moving with the flow looks like:

$$\frac{d\mu}{dt} = \frac{s}{\rho_a} \tag{2.3}$$

where we now have the full derivative instead of partial derivative. These equations can also be written in terms of the **molar mixing ratio, C** of the tracer which is given by,

$$C = \mu \cdot \frac{M_a}{M} \tag{2.4}$$

where M and M_a are the molecular mass of the tracer and air respectively. The above equations are continuous, however in numerical simulations we have to use their discretised versions. Processes like turbulence, convection etc cant be resolved on the typical discrete scales used, so they are represented using approximate parametrizations. Advection on the other hand is resolved and the rate of change of mass density of tracer is given by,

$$\left[\frac{\partial \rho}{\partial t}\right]_{adv} = -\nabla \cdot (\rho v) \tag{2.5}$$

which is essentially Eulerian equation 2.1 without the local term, however this applies only to the model-resolved winds for which we have actual information on $\mathbf{v}$. Smaller-scale transport, such as by turbulence are modeled by a diffusion-based parameterization called *turbulent diffusion/mixing* that has 2nd order partial derivative of the tracer density. So actually solving equation 2.1 on a discrete model involves parameterising all the processes that we want to model:

$$\frac{\partial \rho}{\partial t} = \left[\frac{\partial \rho}{\partial t}\right]_{adv} + \left[\frac{\partial \rho}{\partial t}\right]_{turbulence} + \left[\frac{\partial \rho}{\partial t}\right]_{emission} + ... \tag{2.6}$$

and finally we get a partial differential equation (PDE) which can be in 1D, 2D or 3D. As we saw in equation 2.5, advection (which is the only process relevant for us here, except emission) involves the divergence of the product of (wind) velocity field and density field of the tracer. So for a discretised model, the partial derivatives need to be approximated on a discrete grid of points in space and for this 2 very common methods are: *finite difference method* and *finite volume method*.

In **Finite Difference Methods**, a partial derivative, say $\partial\rho/\partial x$ at a point x_i is approximated by finite-differences using the Taylor expansion of ρ around the point:

$$\rho(x_i + \Delta x) = \rho(x_i) + \Delta x \left(\frac{\partial \rho}{\partial x}\right)_{x_i} + \frac{\Delta x^2}{2!}\left(\frac{\partial^2 \rho}{\partial x^2}\right)_{x_i} + \frac{\Delta x^3}{3!}\left(\frac{\partial^3 \rho}{\partial x^3}\right)_{x_i} + ... \tag{2.7}$$

From this we get,

$$\left(\frac{\partial \rho}{\partial x}\right)_{x_i} = \frac{\rho(x_{i+1}) - \rho(x_i)}{\Delta x} + O(\Delta x) \approx \frac{\rho(x_{i+1}) - \rho(x_i)}{\Delta x} \tag{2.8}$$

where $O(\Delta x)$ is the error term, proportional to Δx, hence this is called *first-order approximation*. Similarly, one can obtain finite-difference approximations of higher

orders for better accuracy. Approximations of partial derivative with respect to time $\partial\rho/\partial t$ can also be obtained in a similar way and in this way, the time evolution of ρ can be computed numerically over the grid.

In **Finite Volume Methods**, the density ρ is not calculated for points of a grid, but expressed as average quantites over *grid cells* formed by the points. In equation 2.1 ρv essentially describes the flux F, so in a 1D case, it looks like (without local term):

$$\frac{\partial \rho}{\partial t} + \frac{\partial F}{\partial x} = 0 \tag{2.9}$$

The density at time $t_{n+1} = t_n + \Delta t$ can be written as -

$$\rho(x, t_{n+1}) = \rho(x, t_n) - \int_{\Delta t} \frac{\partial F}{\partial x} dt \tag{2.10}$$

Now for our discrete case, since we will only have average of ρ over the cell i, with its boundaries denoted by $i - 1/2$ and $i + 1/2$:

$$\langle \rho_i(t_{n+1}) \rangle = \langle \rho_i(t_n) \rangle - \frac{1}{\Delta x_i} \int_{\Delta t} (F_{i+1/2} - F_{i-1/2}) dt \tag{2.11}$$

Taking time derivative of this and substituting $F = \rho v$, we get:

$$\frac{\partial \rho_i}{\partial t} = \frac{\rho_{i-1/2}\, v_{i-1/2} - \rho_{i+1/2}\, v_{i+1/2}}{\Delta x_i} \tag{2.12}$$

The densities and velocities at a previous time-step for the cell-averages will be known. To compute those quantities for cell boundaries, an interpolation method is used. In this way, the densities at future time-steps can be calculated. Finite-volume methods ensure conservation of mass despite being discrete but finite-difference methods in general do not.

In the above schemes, the equations are solved for a fixed grid relative to the ground, ie the frame of reference is the ground. Such methods are called **Eulerian Methods** and one important consideration for them is numerical stability, ie whether the numerical errors remain stable in magnitude or not. For Eulerian schemes, the *Courant–Friedrichs–Lewy (CFL) criterion* says that if $\alpha = c\frac{\Delta t}{\Delta x}$ then $\alpha \leq 1$ should be satisfied for numerical stability. α is called the *Courant number*, c is the wind velocity, Δt and Δx are the time and space step/resolution respectively. Hence for a given grid

spacing, the CFL condition imposes an upper limit on the time-step that we can use, which has repercussions on the computational complexity of the overall numerical model.

Another important consideration is the effect called **numerical diffusion**. One of the ways that the numerical errors arising from the approximations (like discussed above) manifest is a smoothening effect of the features, especially of sharp gradients in the density. The effect is reminiscent of molecular diffusion, but has no physical origin/counterpart. Fig 2.1 [25] shows an example of this for a 4th-order Eulerian scheme, where the initial tracer distribution is a rectangular function but numerical diffusion smears out the shape after some time-steps. *Numerical diffusion is a major concern for Eulerian schemes and (along with advection) is much more significant than molecular diffusion, which is why the latter is ignored in large-scale transport models.* Hence GEOS-Chem also considers molecular diffusion negligible, which is why our deep learning model won't be learning its effects either (however it will learn the effects of numerical diffusion).

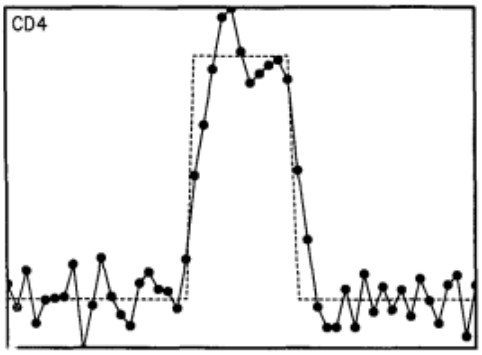


Figure 2.1

Another class of numerical models involve using the frame of reference of the fluid/air instead and we follow the trajectory of infinitesimally small air parcels. These are called **Lagrangian Methods** and the trajectory of an air parcel is calculated using-

$$\frac{dr(t)}{dt} = v(r(t)) \tag{2.13}$$

This equation is integrated using a numerical approximation like-

$$r(t+\Delta) = r(t) + \frac{\Delta t}{2}[v(r(t)) + v(r(t+\Delta t))] \tag{2.14}$$

which is a second-order solution, solved iteratively. For 3-dimensional simulations of global scale, trajectories of a large number of such parcels is calculated. These parcels maintain their integrity and particles in different parcels dont mix together, chemically or physically.

The advantages of these methods are that there is no numerical diffusion, spatial resolution can be improved by simply increasing the number of air parcels and the time resolution is not limited by the CFL condition for numerical stability. However, the absence of mixing between air parcels can be a significant problem for chemistry and aerosol processes. They cant provide uniform coverage since depending on the flow, some regions will be densely populated while others not. Also, in a large simulation, tracking a large number of particles can be computationally expensive.

In **semi-Lagrangian Methods (SLM)**, the elements of both Lagrangian and Eulerian frameworks are combined. The concentrations/densities are calculated on a fixed set of grid points, but the movement of fluid elements is tracked backward in time, from their current positions ('arrival points' that are taken to be the grid points) to their positions at a previous time step ('departure points'). The latter usually dont coincide with a grid vertex, so the concentration is interpolated using neighboring values (see Fig 2.2 [24])

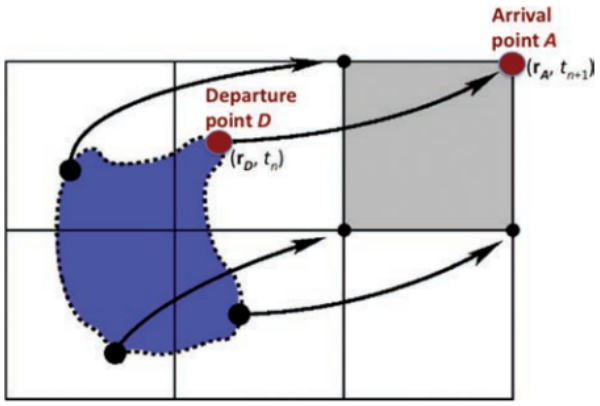


Figure 2.2

A relevant example of an interpolation method is the *Piecewise Parabolic Method (PPM)*, in which the distribution $\rho(x)$ inside a cell is represented by a quadratic function where the coefficients are determined by the boundary and cell-average values.

In the above SLT scheme based on the grid points, the crucial requirement of mass conservation is not satisfied, without explicit adjustment. In the finite-volume

formulation of this method, mass conservation is ensured by back-tracking a whole grid cell instead of the points. The mass before and after remains the same, the 'arrival' volume is the grid cell, the 'departure' volume is computed based on which the departure density is computed using interpolation, so we get the arrival density by-

$$\begin{aligned} \langle \rho_i^{n+1} \rangle V_i^{n+1} &= \langle \rho_i^n \rangle V_i^n \\ \Rightarrow \langle \rho_i^{n+1} \rangle &= \frac{\langle \rho_i^n \rangle V_i^n}{V_i^{n+1}} \end{aligned} \tag{2.15}$$

Like lagrangian methods, SLMs are also computationally stable for any time-step but in addition, mixing and chemistry between air parcels is allowed. The computational complexity of tracking individual particles and numerical diffusion are also mitigated. However, compared to Eulerian methods, SLMs can be slower. With this background, we can discuss the numerical scheme that GEOS-Chem uses [15][25].

The CFL condition for stability is highly prohibitive because for a global model with a latitude-longitude grid, the Δx separation in the east-west direction gets very small at the polar regions. This drives the maximum timestep we can keep to smaller values, increasing computational complexity. So to maximise efficiency, **GEOS-Chem uses a (modified) semi-Lagrangian finite-volume scheme for transport along the east-west direction only. For the north-south and vertical directions, it uses the faster Eulerian scheme**. Because of this, the time-step is limited only by the meridional Courant number which allows for a time-step big enough to make the model faster. For interpolation, the PPM scheme is used.

Chapter 3

Deep Learning Preliminaries

3.1 Relevant concepts

Deep Learning networks are deep neural networks that are trained and used to perform complex tasks. Neural Networks are composed of multiple *layers* of *neurons* that are connected to each other to form a *network* and these connections have *weights* associated with them. The network as a whole takes inputs in the form of arrays of numbers that are processed layer by layer until the last layer is reached. The values of neurons at the last layer are considered as the output. A very simple neural network *architecture* is shown below in Fig 3.1 [1].

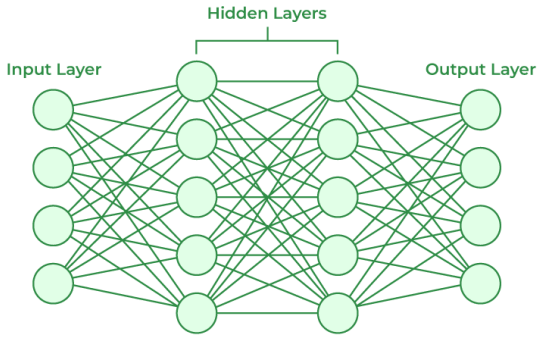


Figure 3.1: Example of a Neural Network (with dense layers)

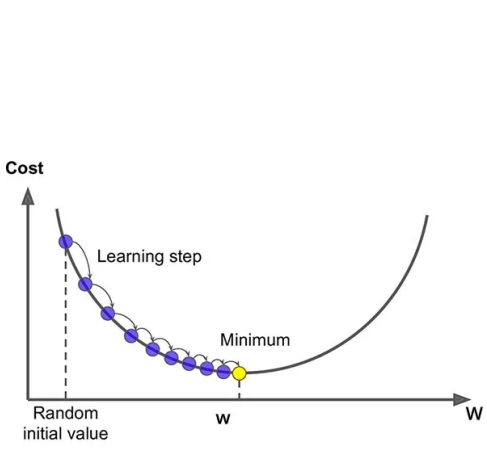
A very common way to train a neural network is as follows: initialise the weights w_i of

[1]Source: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/

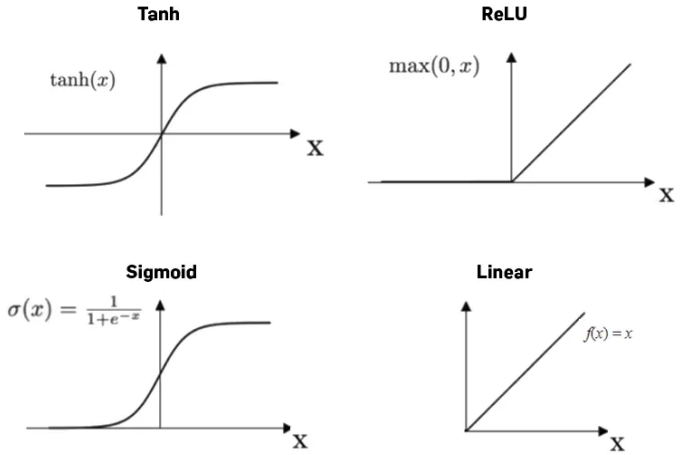
the network with random values, feed the input to the network and record the output, calculate the dissimilarity of this output with the *correct* output using some suitable function, often called the 'cost/loss function' $L(w)$. Next, calculate the gradient of this cost function with respect to *every* weight of the network, $\frac{\partial L(w)}{\partial w_i}$. Then update each of the weights using the following expression-

$$w_i = w_i - \alpha \cdot \frac{\partial L(w)}{\partial w_i} \tag{3.1}$$

where α is a scalar value called the *learning rate*. The loss function is a function of all the weights of the network and depending on their values, it will attain its minimum value somewhere. We want our weights to get as close to those set of values so that our network's predicted outputs are as close to the true outputs. Repeating this step iteratively takes us to a (local) minima of the cost function. In machine learning in general, this algorithm of 'training' a model using the gradients of a loss function is called the **gradient descent algorithm** (figure 3.2 below [2]) and when we explicitly give our model the 'correct' outputs to learn from it is called *supervised learning*.



(a) Stochastic Gradient Descent



(b) Examples of activation functions

In neural networks, the calculation of the gradient of the loss starts from the parameters of the last layer and ends with the ones in the first layer. This process of propagating the loss in the backward direction is called **backpropagation**.

The value of the learning rate α is often kept small (0.1 to 0.001). This is because we dont want to make huge jumps while going towards the minima of the cost function,

[2]Source: https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/

especially close to the minima, lest we will keep oscillating around the minima and not get closer.

A neuron upon receiving the values of neurons from the previous layer (combined with the weights), it uses an **activation function** on the final value. This is a non-linear function and without it any neural network no matter how big is only a linear function in effect. Because of the activation functions, a neural network is able to learn complex non-linear patterns in the data. Common examples of activation functions are `ReLU`, `sigmoid`, `tanh` etc as shown in Fig 3.2 [3].

In principle, the loss function depends on the weights of the network *and* all the inputs and true outputs that we give it during the training. This implies that in order to do the step in Eq 3.1 even once, we need to calculate the loss for all the input-output pairs. This process becomes a huge limitation when our network is large and we have a large number of input-output pairs. Hence in practice, the dataset is divided into 'mini-batches' and the weights are updated by taking one mini-batch at a time, instead of the whole dataset. While this helps speed-up the training process, not using the whole data set means that we are using an *approximation* of the true gradient of the cost function. So our weight updates won't exactly be in the direction of the minima, but for a sufficiently large batch-size it will be a good approximation. This is called **mini-batch gradient descent**. Furthermore, in practice we have to go through the entire dataset (over multiple mini-batches) multiple times (called **epochs**) to achieve desired results.

As noted earlier, the gradient descent algorithm allows us to reach a local minima of the cost function. However, the local minima might not be good enough for our purposes and we would like to initialise our network such that its less likely to get caught up in shallow local minimas. Empirically, weights initialisation using a uniform/normal distribution around 0 has yielded good results. The width of these distributions is a small number dependent on the number of neurons from the previous layer that are connected to a neuron in the current layer. Some common methods are called `Xavier/Glorot` initialisation, `He` initialisation etc.

The gradient descent is one way of updating the weights of the network. Turns out, that even with good initialisation methods, it can often suffer from local minimas, slow convergence and sub-optimal performance. A modified version of gradient descent called

[3]Source: https://machine-learning.paperspace.com/wiki/activation-function

Adam optimiser has become the default choice for a wide range of problems. The Adam optimiser brings 3 major improvements: firstly for a given iteration it replaces the gradient by the moving average of the past gradients (denoted by m_i); secondly it divides the learning rate by the square root of the moving average of the *square* of the gradient (denoted by v_i); and lastly it calculates these averages separately for all the weights. The first modification is also called *momentum*, since it allows the training to go past shallow local minimas as if it had some 'momentum'. The second modification is useful when the network comes around the global minima but due to the momentum it keeps oscillating around it. The average squared gradient is essentially a measure of the variability of the gradient and during this oscillation, the large variability factor in the denominator will help more-than counteract the momentum, allowing the network to reach the global minima more smoothly. The third modification helps because each parameter has different needs of the momentum and variaiblity factors and so we get much better performance because of this. The weight update step for a weight w_i in `Adam` looks like-

$$w_i = w_i - \alpha \cdot \frac{m_i}{\sqrt{v_i} + \epsilon} \tag{3.2}$$

where ϵ is a very small scalar (eg 10^{-8}) added to prevent division by zero.

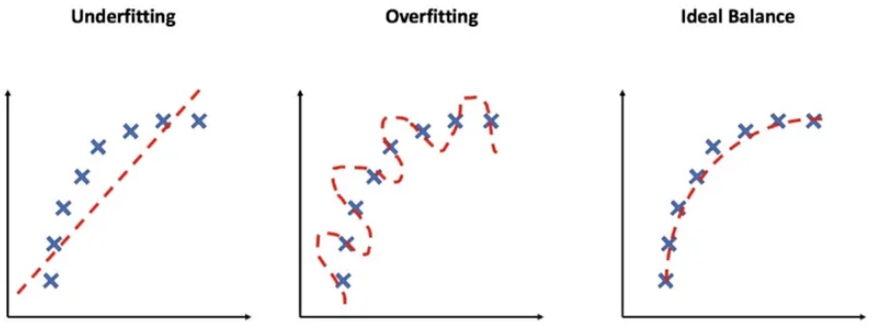


Figure 3.3: Example of underfitting and overfitting

In the process of training a machine learning model in general, a very common problem is **overfitting**, ie if the model being trained is more complex than inherent structure of the data, it will start fitting to each point in the data exactly in order to minimise the error/loss as much as possible (Fig 3.3 [4]). The model starts losing its generalisability and will perform much worse when tested on new data. Hence, to monitor the generalizability of the network during training, its common to keep a fraction of the

[4]Source: https://medium.com/geekculture/investigating-underfitting-and-overfitting-70382835e45c

whole dataset hidden from the training process; this subset of the data is called the **validation set/data**. During training we not only calculate the loss of the network for the *training data* (for backpropagation etc) but also for the validation data set as well. The loss on the latter tells us whether the generalisability of the model is getting better or worse. Usually, training is stopped when the validation loss starts increasing beyond a limit.

Another practical problem while working with real training data is that different features of the data can have different scales of values eg when they represent different physical quantities. In such scenarios the loss function can get much more sensitive to certain features while not learning from other features at all. To avoid this, all features are normalised to a similar small scale. Often the output can also be scaled down for better performance.

A relatively new technique to further improve training is called **Batch Normalisation**. In this, each feature in the input to a hidden layer (commonly before activation) is normalised across the mini-batch by its mean and standard deviation and then rescaled using 2 learnable parameters β and γ (see Fig 3.4[5]). Each Batch Normalisation layer keeps track of the moving average of the mean and standard deviation of each feature/channel during training and uses them and the learned parameters for normalising and rescaling the data when the network is applied on the validation data. Batch Normalisation can be done for all hidden layers and has been found to stabilise the training, improve generalisability (ie reduce overfitting), reduce the need for careful initialisation of the weights and allow for higher learning rates (faster convergence).

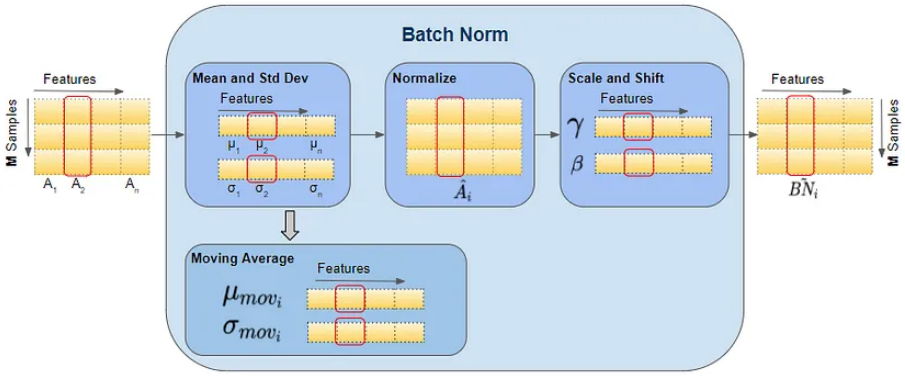


Figure 3.4: Illustration of Batch Normalisation

[5]Source: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

The layers in Fig 3.1 where a neuron is connected to every neuron in the previous layer are called **dense/fully-connected layers**. However there are many different kinds of layers, one of the most common being **Convolutional Layers**. These are very effective when the data has spatial structure and patterns like in images. In a convolutional layer, the 'weights' are essentially a **kernel**- a filter usually square of size (in the case of 2D convolution for example) typically ranging from 3×3 to 9×9 that slides along the images and the elemental multiplication of the kernel with the inputs is turned into the value of the neuron in next layer (followed by activation). An example of this 2D convolution is shown in Fig 3.5 [6]. A similar computation can be done if the data is 3D and has spatial structure in 3 dimensions; in that case a 3D kernel is used (eg $3 \times 3 \times 3$) that slides along the data in the 3 dimensions and the output is also 3 dimensional. Furthermore, a given convolutional layer consists of many **channels**; each channel has its own kernel and it has been observed that while training, different channels capture different spatial features from the previous layer (which might also be a convolutional layer with multiple channels). The different features in the channels help the network get a complete representation of the data.

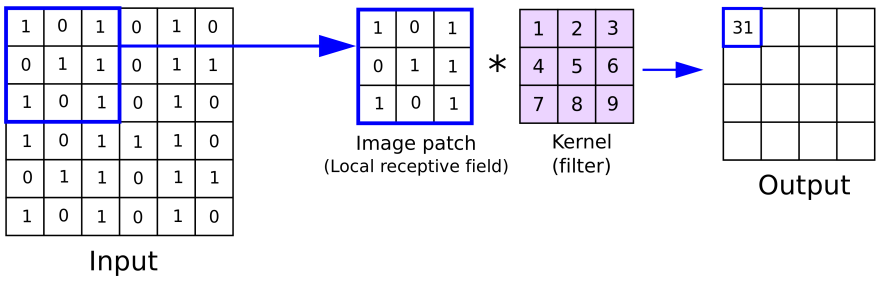


Figure 3.5: Example of a 2D Convolution Layer

Often many architectures reduce the dimensionality of the channels by *downsampling* and increase it in further layers by *upsampling*. This has been found to increase the generalisability of the network and help in learning (see section 4.1 for example). For downsampling a very common way is to use **MaxPooling Layers**, where a window (also called kernel) of size say $k \times k$, slides along the channel and the maximum value in the window is taken as the value for the cell in the next layer. The window slides with a *stride*, ie for a stride > 1, it skips $stride - 1$ cells, which allows a reduction of the dimensionality by factors of 2 or 3 easily. An example of max-pooling with a 2×2

[6]Source: https://iq.opengenus.org/content/images/2023/01/2023_01_20_0te_Kleki-min.png

kernel and stride 2 is shown in Fig 3.6[7].

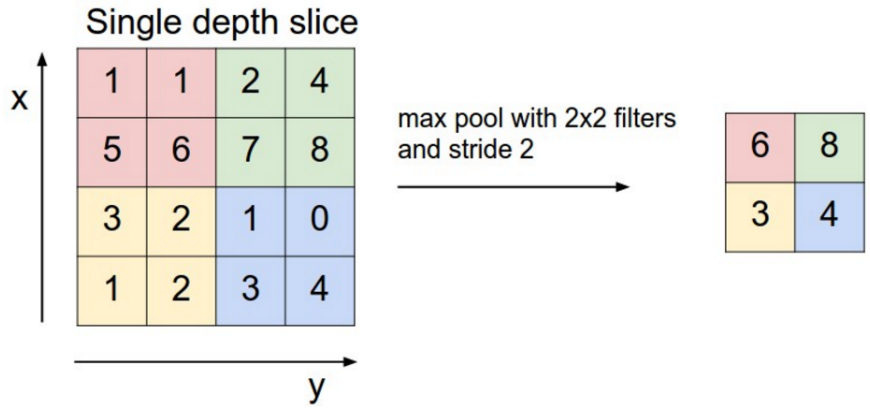


Figure 3.6: Example of a 2D MaxPooling Layer

For upsampling, a very common method is to use the **Transposed Convolution Layers**. For a given input, kernel size, stride and other parameters like *padding* and *dilation*, this operation's output is such that if one applies a convolution layer to it with the same parameters, then we get an output of the same dimensionality as the input to the transposed convolution layer. An example to see how this is done is shown in Fig 3.7[8], where the kernel is 2×2 and the stride is 1. Its important to note that while its sometimes loosely referred also as a *deconvolution* operation, it is not. If we applied convolution on the 3×3 output like in the example with the same kernel values, the result will be 2×2 but **not** with the same values that we began with.

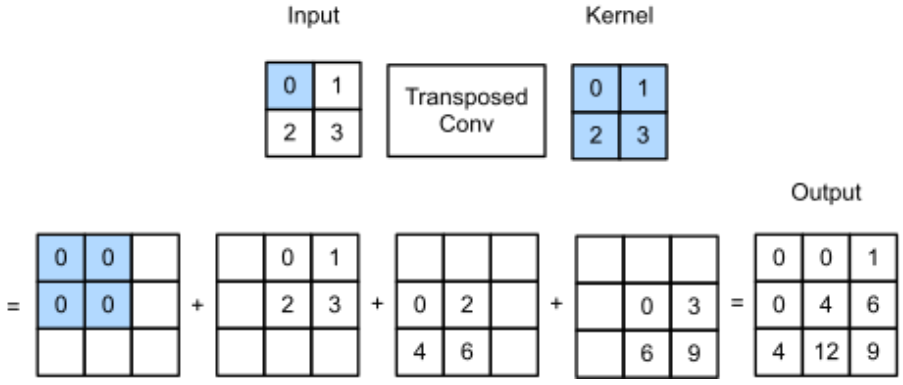


Figure 3.7: Example of a 2D Transposed Convolution Layer

[7]Source: https://www.geeksforgeeks.org/apply-a-2d-max-pooling-in-pytorch/
[8]Source: https://d2l.ai/chapter_computer-vision/transposed-conv.html

Except for computing activation functions, most of the training of a Neural Network is addition and multiplication calculations. However, for deep neural networks ('deep learning'), because of the large number of layers and size of the data, the learning algorithm has to perform billions of such calculations repeatedly. Performing this on a CPU becomes prohibitively slow, so instead the use of **Graphics Processing Units (GPUs)** has become extremely common for deep learning. GPUs are different processing circuits that are designed to perform a large number of calculations in parallel. For this reason, GPUs help in speeding-up NN training by huge factors.

Lastly, it should be noted that in most conventional (supervised learning) applications of deep learning, the 'correct' input (and output) is known and hence it is kept fixed while the network's weights are treated as tunable parameters to learn the input-output dependence. We will see in the next section how we have tried to modify this usual scheme for emission correction.

3.2 Proposed Deep Learning solution

As mentioned earlier, our algorithm for estimating emissions or to be more precise, correcting a prior distribution of emissions involves 2 parts. One is building a deep learning surrogate of an existing numerical transport model and the second part involves using the surrogate model to perform emission correction. The step-by-step plan for the 2 parts is explained below-

Part 1: Training a deep learning surrogate of the transport model

This would involve the following steps:

1. Run simulations with the numerical transport model with varying emission distributions, meteorological conditions and atmospheric concentrations of the gas (CO_2 in our case) at the start of the simulations.

2. The (untrained) DL surrogate takes as input (from the above simulations) (i) the emission distribution (ii) meteorological conditions and (iii) initial concentrations and gives as output the concentration of CO_2 in the future. This output is compared with the output of the transport model and the error is backpropagated to update the weights of the network.

3. Repeat step 2 using the many input-output pairs generated in step 1. Keep a fraction of the training data aside as the 'validation set' to gauge the network's performance on unseen data. Keep updating the parameters of the neural network until the validation error stops decreasing.

4. Finally we have a deep-learning surrogate of the transport model that can reproduce its outputs as closely as possible. In other words, we have a neural network that can predict concentrations of a gas into the future.

Fig 3.8 shows a flowchart for the above steps.

The core of deep learning is to compute gradients of a cost function with respect to the parameters of the model. We can use the same functionality to compute the gradients

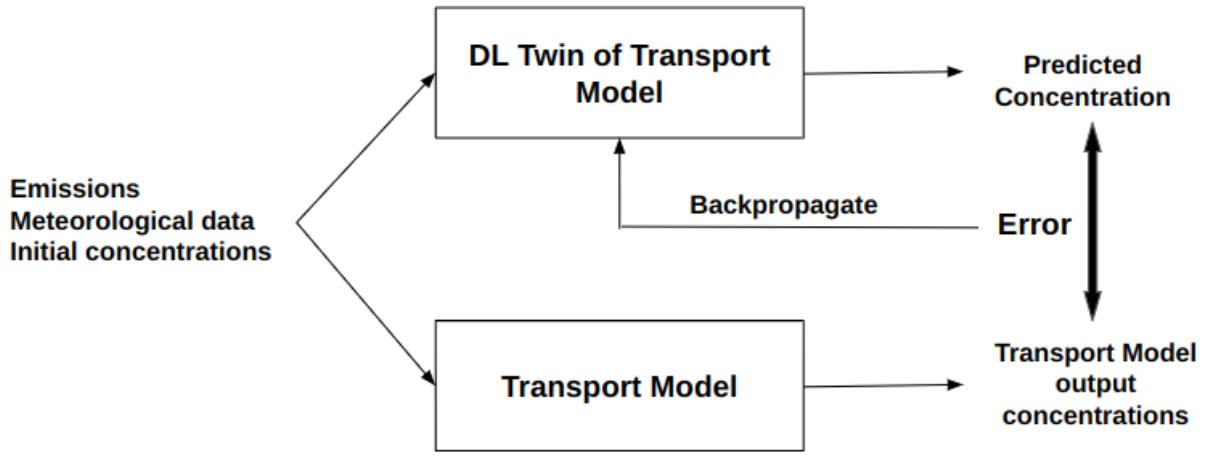


Figure 3.8: Training of the DL-surrogate of the transport model

with respect to the input predictors as well and the major advantage of deep learning networks for our problem is that we can do this readily. Part 2 of our solution is explained below:

Part 2: Use the transport surrogate to perform emission-correction

1. Take an initial (prior) estimate of flux/emission distributions. This estimate can come from bottom-up inventory-based methods or remote sensing observations and analyses.
2. Collect satellite/ground-stations-based data of the concentrations of the gas for the relevant dates. For the former, these will be total-column measurements produced using vertical averaging kernels, so the data will be 2-dimensional. For the latter, we will only have surface measurements.
3. Use the transport surrogate trained in part 1 to generate one-day prediction of the concentrations using the relevant meteorological data and initial concentrations.
4. These concentrations will be over a 3D grid. If using satellite data, convert them into total-column using the same averaging kernels as used for and provided by the satellite data. The result will be a 2D grid of concentration predicted by the transport surrogate.
5. Compare the transport surrogate's predictions with the observational data and compute the 'loss'. Here we make the very strong assumption that the

measurements and the transport surrogate have no inaccuracies and the source of the error is entirely due to the inaccuracies in the (prior) emission distribution.

6. Hold all the parameters of the transport surrogate fixed. Backpropagate the loss through the whole network to the emission values and update them.

7. With the updated emission values, repeat steps 3-6 while monitoring the magnitude of the loss between the surrogate's predictions and concentration measurements. Stop the iteration when the loss stops decreasing.

For any estimate of emission distribution that needs to be corrected, the methodology described in Part 2 can be followed. The result will be an updated version of the emissions that agree as much as possible with the future measurements of concentrations (assuming the transport model to be completely accurate). Fig 3.9 illustrates all the above steps.

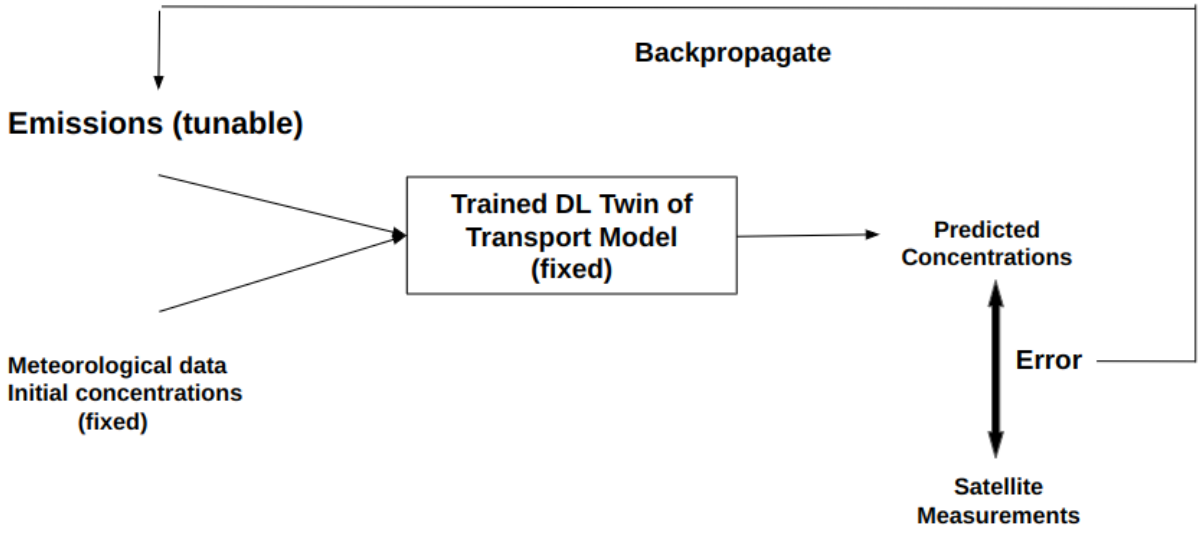


Figure 3.9: Emission correction using transport surrogate and satellite measurements

As noted earlier, our ability to do emission correction will strongly depend on whether the transport surrogate can learn to predict concentrations with good accuracy. It will need to learn the effect of the emissions on the output very well. Hence, in this thesis we focus only on Part 1 of our broader solution; implementation of Part 2 is left for future work.

Chapter 4

Training Deep Learning surrogate of the numerical transport model

4.1 Neural Network Architecture

Since the DL model is supposed to emulate the behavior of the numerical transport model, as noted earlier the network should take as input:

- the emission distribution of CO_2 over a 2D grid
- its initial atmospheric concentrations over a 3D grid and
- the required meteorological conditions eg winds

The output of the network should be the concentrations at a future time on a 3D grid. As explained later, the prediction time-scale of our network will be very short. For short time-scales the concentration value at a given grid cell is affected only by its neighboring cells, hence there is no need of using dense layers in our neural network, since in them the value of a neuron is affected by all neurons in the previous layer. In fact atmospheric transport can be looked at as a time evolution of the spatial structure in the 3D concentrations due to processes like advection etc. Hence, using 3D Convolutional layers in our network should be the most effective.

Suppose we take initial concentrations over a grid of size $h \times d \times d$, then the size of the

output will also be $h \times d \times d$. As mentioned in section 3.1, in order to ensure the convolutional network learns the inherent structure, trends and patterns (which in our case come from the physical processes) in the data, it is common to reduce the dimensionality of successive layers (by downsampling), then increase it again in following layers in such a way so that at the output layer the dimensionality matches that of the first layer. This helps in learning because it forces the network to *encode* the information in the data in a condensed form in those middle layers, then *decode* it in the second half of the network.

This overall structure, which is also the basis of *convolutional auto-encoders* is present in the **U-Net architecture** first proposed for the problem of biomedical image segmentation [26]. This architecture uses the typical encoder-decoder structure, where the dimensionality of the layers is reduced in the encoder using *max-pooling layers* and increased in the decoder using *transposed convolution layers* (explained in the previous chapter). However U-Net builds on this by introducing *skip connections*, where some of the layers in the neural network are skipped and the output of one layer is incorporated into the input to the next non-adjacent layers. One way this is done is to *concatenate* the output of a previous layer with the input to one of the further layers. More precisely in U-Net, the output of a layer of dimensionality d in the encoder is concatenated with the input to the corresponding layer in the decoder part of the network of the same dimensionality d (please see Fig 4.1 for reference). Skip connections are helpful because they allow information and gradients (that might otherwise be lost or diluted by passing through multiple layers) to flow more easily through the network. They can also help to combine features from different levels of abstraction and resolution, which can enhance the representation power of the network. This results in several benefits such as improved accuracy and generalization, solving the vanishing gradient problem, and enabling deeper networks. Because of these benefits and the architecture being very suitable for us, we have used U-Net for our problem.

As mentioned earlier the number of vertical layers we can use in our transport model GEOS-Chem is either 47 or 72 (both options have the same maximum height). Also, the lowest resolution available is $4^\circ \times 5^\circ$ which for a global grid results in the horizontal grid of size 46×72. Now the output of our network would be the future 3D concentrations of CO_2 on the same grid. The larger the grid (in terms of the number of cells) we use, the more number of individual concentration values the network will have to predict. Hence if we use a very fine grid, training the network to predict those many values will get

much harder. Hence to simplify this aspect of the problem a bit, we use 47 vertical levels and a horizontal region of size 16×16 at the $4° \times 5°$ resolution, which roughly covers most of Asia. We have trained our network only for the meteorological conditions over Asia, details are explained in section 4.2.

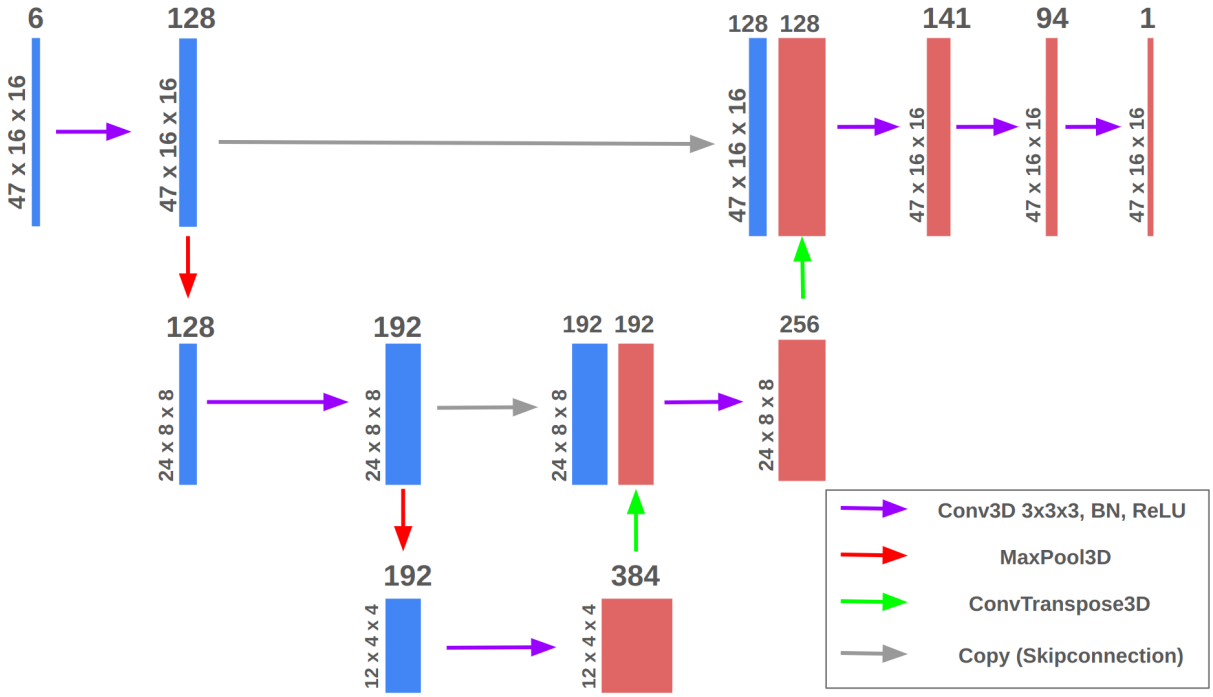


Figure 4.1: The U-Net architecture used in this work
For example, the input (size $6 \times 47 \times 16 \times 16$) is provided to the first 3D Convolution Layer which outputs 128 channels each of size $47 \times 16 \times 16$

The emission distribution will be 2D of size 16×16 since we are considering emission only from the surface. The initial concentrations input will be 3D of size $47 \times 16 \times 16$. The rest of the inputs are various meteorological quantities that we can obtain separately. The numerical model GEOS-Chem takes a large number of these quantities to model various physical processes. So our network would also require all the meteorological quantities if we try to train it for all the processes. Another related issue is the **prediction timescale** of our network. Ideally, we would want our network to be able to predict concentrations several weeks into the future if not more, since CO_2 has a very long *atmospheric lifetime* (ie time-scale in which a molecule of that gas remains in the Earth's atmosphere before it is removed or destroyed by various processes). However, training the network for many physical processes and predicting several weeks into the future quickly becomes unfeasible. For the numerical model at a given timestep,

it takes only the required data and it computes different physical processes separately. However for our neural network, the more processes we try to train the network for and the longer our prediction time-scale, the larger our required input gets, which greatly inflates the model complexity, making it very difficult to train. For example, if the prediction time-scale for our model was 10 days, then we would need to provide the daily meteorological data for all the 10 days as an input to the network. Whereas if we were predicting over 5 days then the required input size is reduced by half.

Hence to simplify our task further, we have trained our network to **predict future concentrations only 1 day into the future** based only on wind-driven **advection**. As explained in Section 2.2, molecular diffusion is considered negligible compared to the effects of numerical diffusion, which will be present in our training data. The most important physical quantities for this are the *horizontal winds* in the east-west and north-south direction. Additionally, we have to include *pressure* and *temperature* as well, since GEOS-Chem uses them to compute the number density of air which affects the concentrations of our transported gas. Hence we have 4 more 3D grids of size $47 \times 16 \times 16$ and in total we have 6 different *features* that go as input to the network. We put these as 6 *channels*, so the input that the network takes is a 4D array of size $6 \times 47 \times 16 \times 16$ (we reshape the 2D emission into 3D keeping the values at all levels as zeros except the first vertical level). This goes as input to a *3D Convolutional layer* with 128 channels, and its output is saved (say, 'skipconn1' to be used for skip-connection later). This is followed by 3D max-pooling which reduces the dimensions to $24 \times 8 \times 8$, then another 3D Convolution with 192 channels (the output is again saved as say, 'skipconn2'). Next, another 3D max-pooling reduces the dimensions to $12 \times 4 \times 4$ which is then followed by a 3D convolution with 384 channels. This goes as input to a transposed-convolution layer that upsamples the features to $24 \times 8 \times 8$ with 192 channels. This output is concatenated with *skipconn2* along the channel-axis so that as a result we get features of size $392 \times 24 \times 8 \times 8$. This is followed by a 3D convolution with 256 channels which is again upsampled by a transposed-convolution layer, and after concatenation of *skipconn1*, our feature size becomes $256 \times 47 \times 16 \times 16$. This is followed by 3 convolutional layers that reduce the number of channels to 141, 94 and finally to just 1 channel so the output of the network is a 3 dimensional array, which will predict the *change in concentration* and not the concentrations themselves (reason explained later) at each grid cell. The full architecture that we used is shown in Fig 4.1.

4.2 Generation of training data

To train any neural network via supervised learning, we need many input-output pairs; that is we need to tell the network what the 'correct' output is for a given input and we need many such pairs. This is where we use the numerical transport model GEOS-Chem for our problem, since the goal is to emulate its output in the first place. We ran several 1-year long simulations using GEOS-Chem using different emission/sink distributions and the meteorological conditions of the year 2019. The simulations gave daily average concentrations as output and since our network predicts concentrations one day into the future, we chopped a simulation into 365 input-output pairs for training. The concentrations predicted by the simulation at the i^{th} day became the 'true' output for one training sample and the initial concentration input for the next sample. The emission and meteorological data were divided into the 365 samples accordingly. The details of the emissions, initial concentrations and meteorological data that we used in the simulations are explained below.

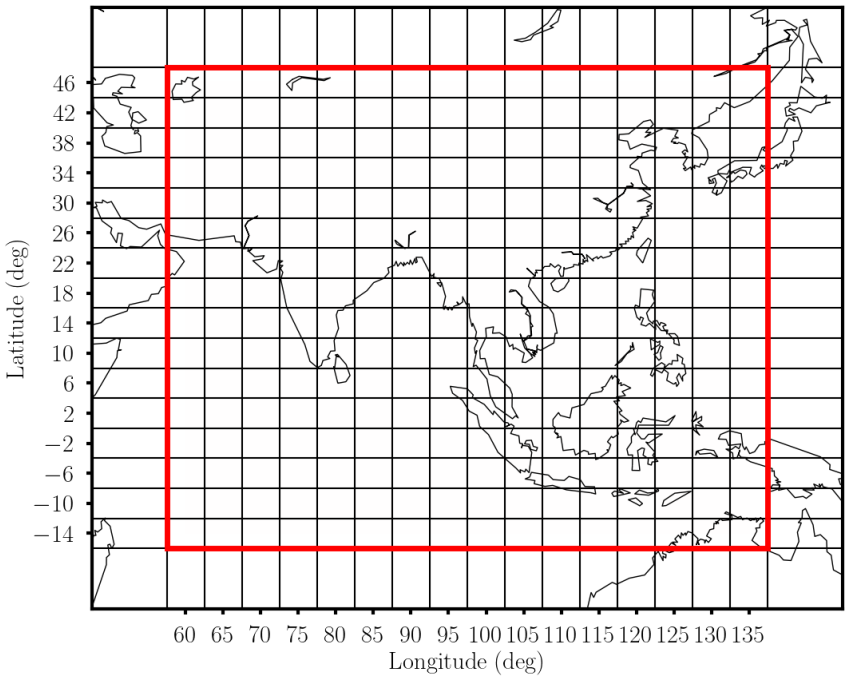


Figure 4.2: Region of Asia used for all simulations (red square)

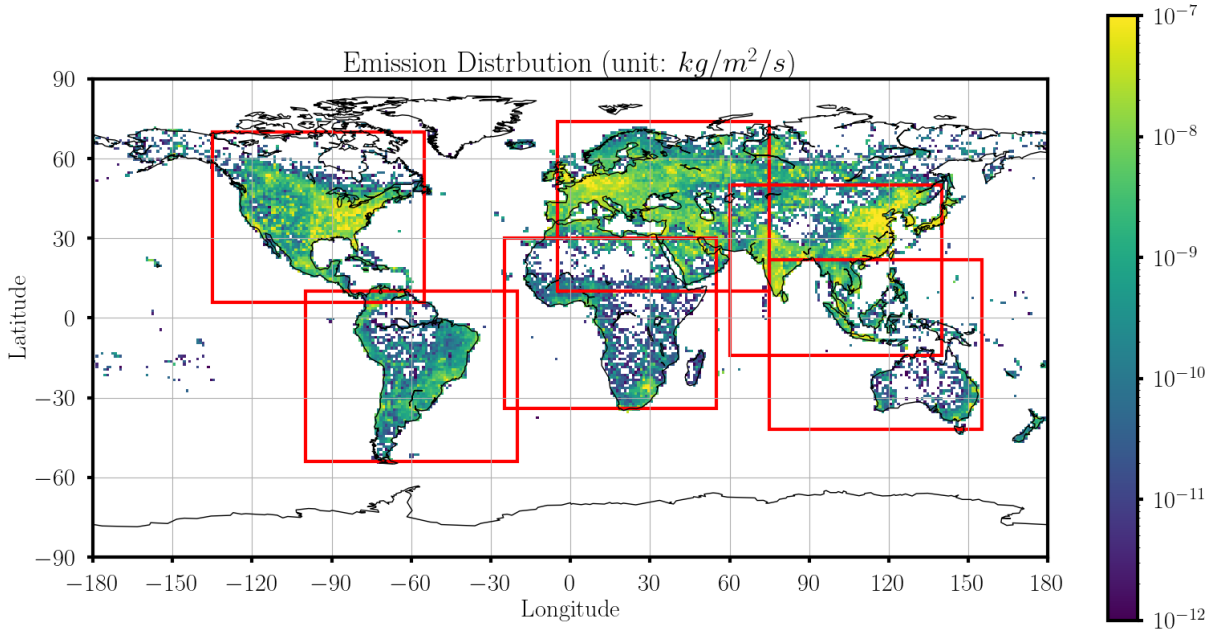
We took different sources/sinks distributions from various processes like biosphere exchange, fossil fuels, biofuel, ocean exchange etc. taken from different sources [27][28]

[29][30][31][32]. For emission/sinks of all processes except ocean exchange, only the land masses (the continents) had non-zero values. So we took regions of size 16×16 over each continent that had non-negligible sources/sinks and created new global distributions where the sources/sinks are 0 everywhere except the 16×16 region over Asia where the above source/sink distributions were fed in. The precise region that we chose for Asia was $[-14°, 46°]$ in latitude and $[60°, 135°]$ in longitude (shown in Fig 4.2). For ocean exchange, the non-zero values came only from the oceans, so we took the source/sink distributions from over all the ocean regions (Fig 4.3). We also generated separate global distributions where the source/sink regions were rotated by 90° and then fed into the 16×16 region over Asia. Please refer to Fig 4.4 for reference. The above steps not only helped us to (i) increase the number of unique emission distributions that the network will see during training and validation, but also (ii) prevent the network from biasing itself towards repeated similar emission distributions (since eg, the fossil fuel distribution coming from say Asia will always have similar hotspot regions in China and India). The emissions/sinks are in the units $kg/m^2/s$.

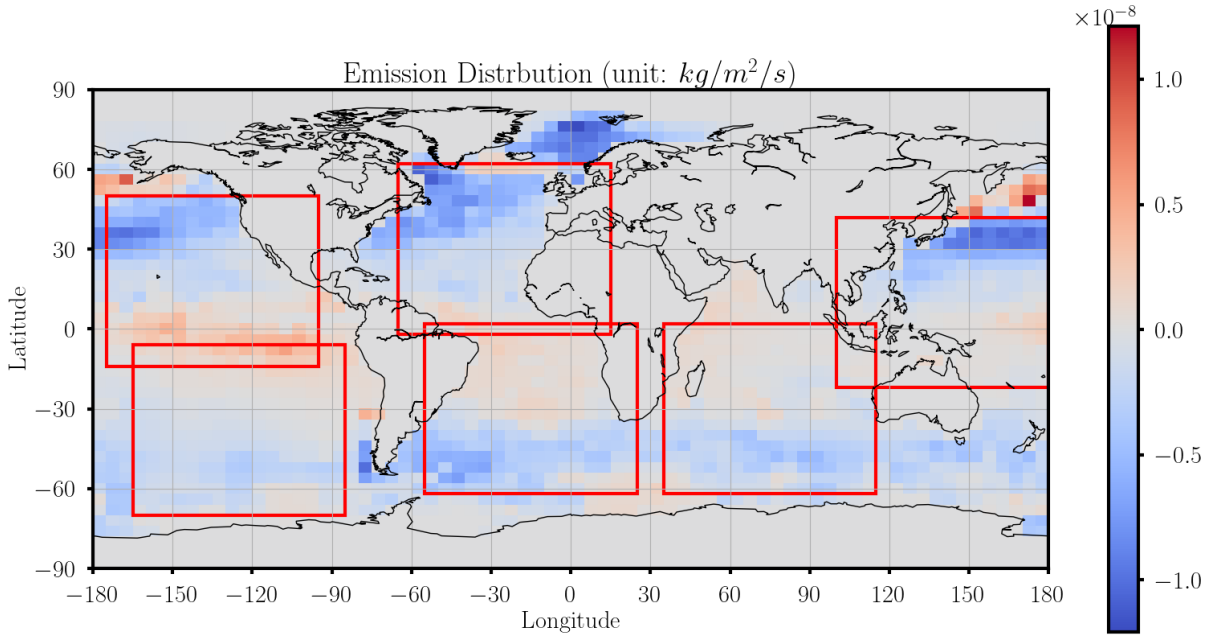
For the initial CO_2 concentrations for our simulations, we took the sample data that GEOS-Chem provides for CO_2 simulations. It does not correspond to 'real' measurements of the concentrations but we visually inspected it to be okay for our purposes. The concentrations for some of the 47 levels over Asia that we used for initial concentrations of our simulations are shown in Fig 4.5.

However as explained above, this doesnt mean that the initial concentration of all our training samples are the same. As different year-long simulations have different emissions, the time evolution of the concentrations after each day is different. For concentrations, GEOS-Chem uses the units of *mole (CO_2) per mole of dry air* which we converted to *parts per million (ppm)* for our training data by simply multiplying by 10^6.

As mentioned previously, we used the meteorological conditions of the year 2019 in our training data and the relevant fields for us are the east-west and north-south winds, pressure and temperature. This data is readily available at http://geoschemdata.wustl.edu/ExtData/ . We used the 'MERRA-2' data product at horizontal resolution of $4° \times 5°$. MERRA-2 is a reanalysis data product from NASA's Global Modeling and Assimilation Office (GMAO). It is produced with GMAO's GEOS-5 Data Assimilation System (GEOS-5 ADAS), which as the name suggests uses sophisticated data assimilation algorithms to obtain reliable data on various

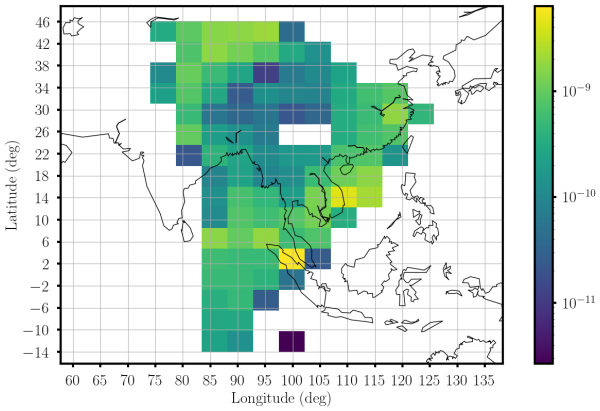


(a) Land regions of size 16×16 used for generating different emission distributions

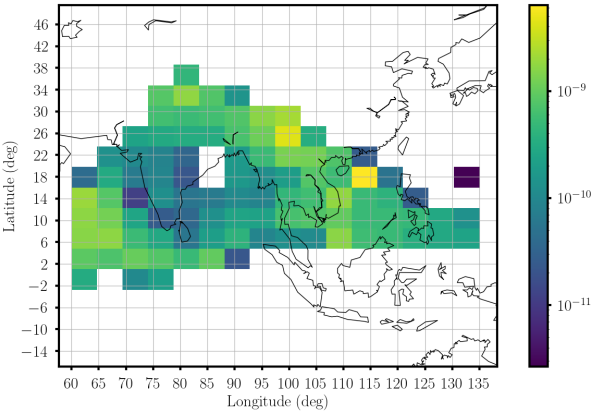


(b) Ocean regions of size 16×16 used for generating different emission distributions

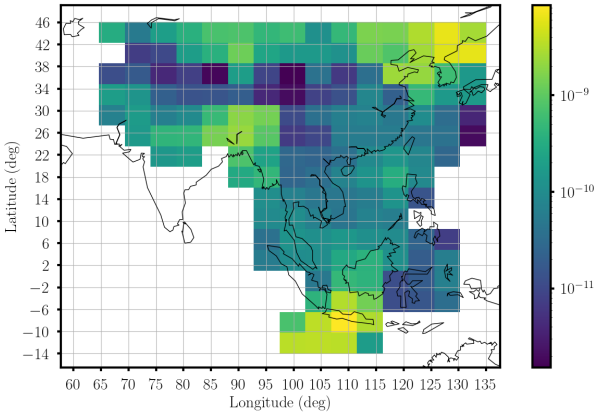
Figure 4.3



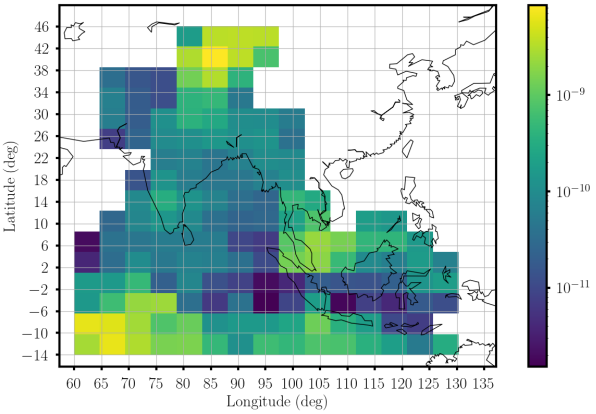
(a) Fossil Emissions from South America



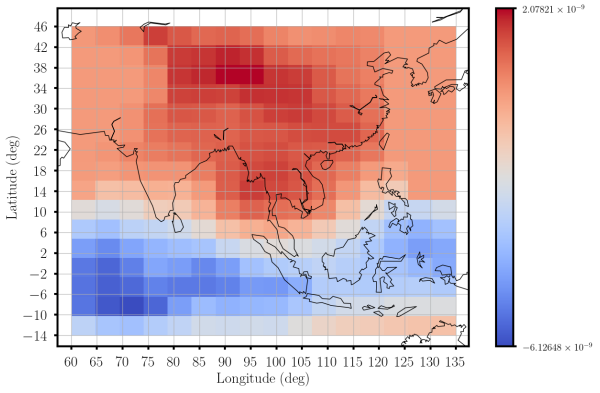
(b) Fossil Emissions from South America Rotated by 90° anti-clockwise



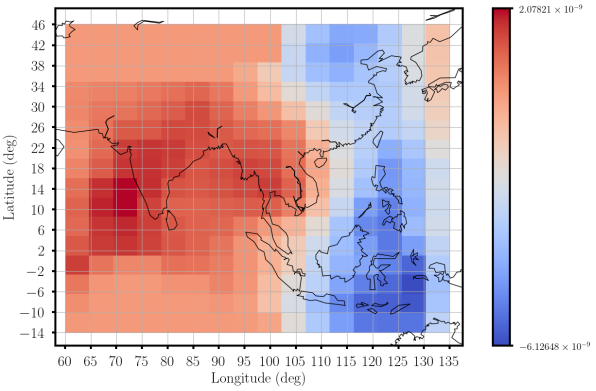
(c) Fossil Emissions from Africa



(d) Fossil Emissions from Africa rotated by 180°

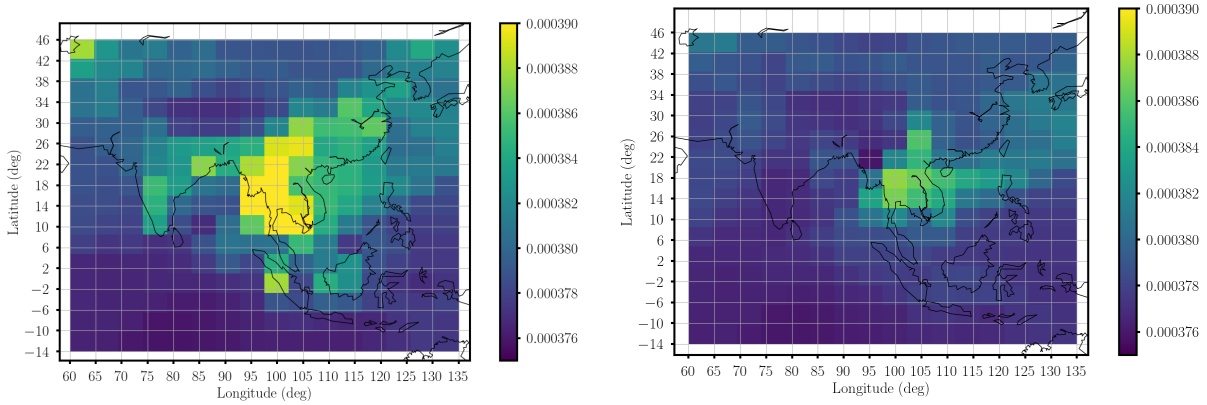


(e) Ocean Exchange Emissions/Sinks from South Atlantic ocean



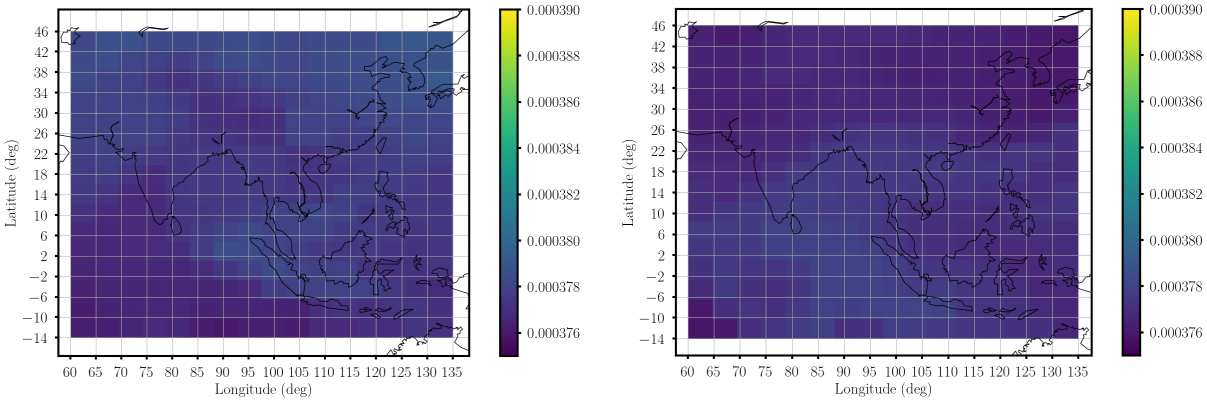
(f) Ocean Exchange Emissions/Sinks from South Atlantic ocean rotated by 90° anti-clockwise

Figure 4.4: Examples of cut-outs of emissions/sinks taken from different parts of the globe and fed into the emissions over Asia for the generation of different emission distributions. These new distributions had 0 emissions everywhere except this Asian region and were used for running simulations on GEOS-Chem for the generation of training data.



(a) Level 1

(b) Level 11



(c) Level 21

(d) Level 31

Figure 4.5: Concentrations (in mol (CO_2) per mol of dry air) over Asia at different vertical levels, used as initial concentrations for the simulations run for generating training data

meteorological quantities. The data has 72 vertical layers, so for our training data, we lumped the data (using the mean) in the vertical direction according to the heights of the 47 layers. The first 36 layers in both the grids are at the same height. From the 37^{th} to 40^{th} layer, every 2 subsequent layers were lumped together and from the 41^{st} to 47^{th} layer, every 4 layers were lumped together.

With all the above inputs ready, we ran 1 year-long **global** simulations on GEOS-Chem v14.2.0, saving daily averaged concentrations (in units of *moles of* CO_2 *per mole of dry air*) with only advection turned on. The timestep of transport calculation was 10 min and the simulations took about 1 hr each to run on an i5 8th generation Intel processor on one core with 8 GB RAM. The simulations were global but in the training data, we used only the 16×16 sized subset of the data over Asia for everything; emissions, concentrations, winds, pressure and temperature. The possible boundary condition problems arising for our neural network due to this are addressed in the next section.

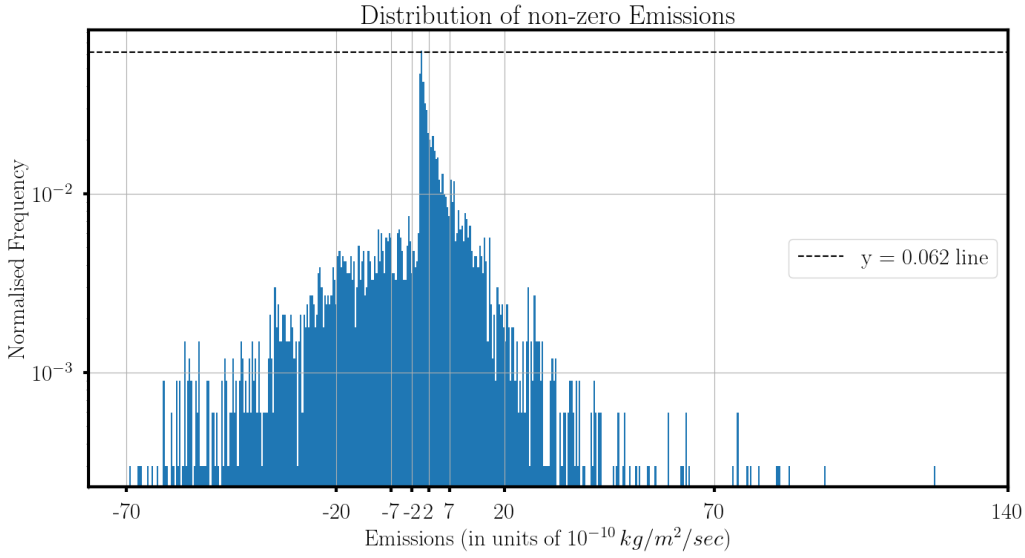
4.3 Specifications of NN training

The exact details of the training were heavily influenced by the challenges in our problem that make it more complicated than an image processing problem:

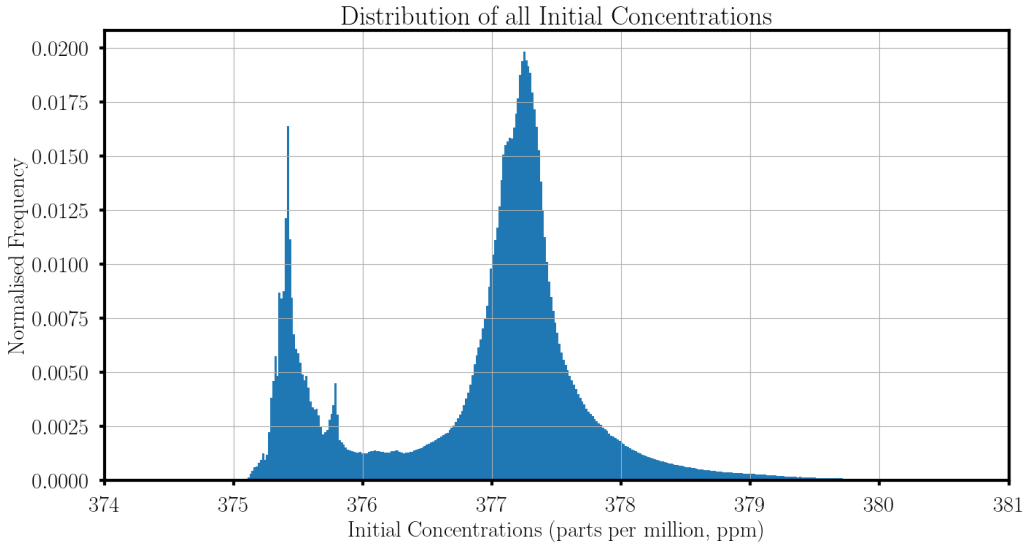
1. We have 5 unique physical quantities in our input that have different units. Not only do they differ wildly in scales from each other, the values of individual features are spread across many scales. While emission/sinks magnitudes vary between 10^{-11} and 10^{-8} $kg/m^2/s$, concentrations in our simulations mostly go from 300 to 410 ppm. In the meteorological fields, the magnitude of east-west winds go mostly from 0.01 to 60 m/s and the north-south winds go mostly from 0.01 to 20 m/s, the pressure is mostly between 0.1 (at the highest layers) and 10^5 *Pascals* (close to the surface), and the temperatures go from about 300 to 410 K. The distributions of these quantities for the year 2019 are shown in Fig 4.6.

   As explained in the previous chapter, its well known that neural networks train better and faster when all the inputs and outputs are scaled/normalised to a similar scale of small values, typically between 0 and 1 or -1 and 1. If the scales are vastly different, gradients computed during training may have varying scales across for different parameters. This can lead to a highly non-uniform landscape of the loss function. As a result, the optimization process may prioritize certain parameters over others, potentially leading to elongated or distorted contours in the loss function. Renormalising the features to the same scale ensures that the loss landscape becomes more uniform and the network learns the importance of various features from the inherent structure in the data, rather from the scales/magnitudes of their values.

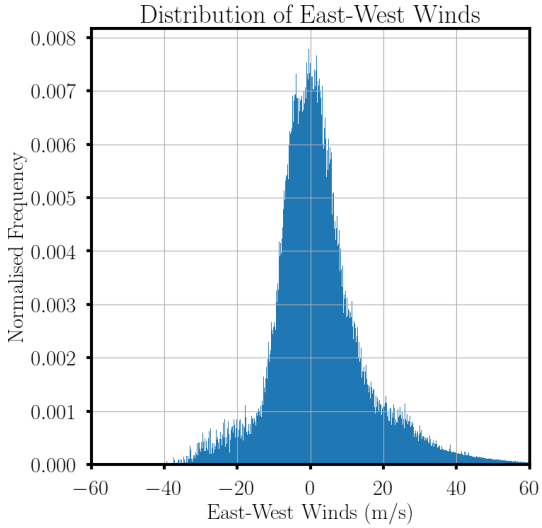
2. Since we are providing the initial concentrations as input to the network, it is the *change* in the concentrations that the network is learning. Moreover, instead of computing the loss/cost function in terms of the inaccuracies in the predicted concentrations, using the change in concentrations as the target to predict will allow for more sensitivity in learning. However when we look at the distribution of the concentration change in our data set, we see that the values range from about 10^{-4} to about 5 ppm. As before, this large spread of the values will make the training difficult.


Distribution of non-zero Emissions
Normalised Frequency
10^{-2}
10^{-3}
y = 0.062 line
-70
-20
-7
-2
2
7
20
70
140
Emissions (in units of $10^{-10}\,kg/m^2/sec$)

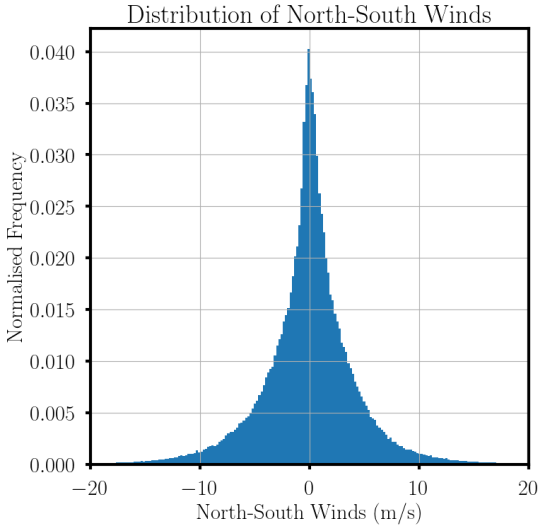
(a)


Distribution of all Initial Concentrations
Normalised Frequency
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
374
375
376
377
378
379
380
381
Initial Concentrations (parts per million, ppm)

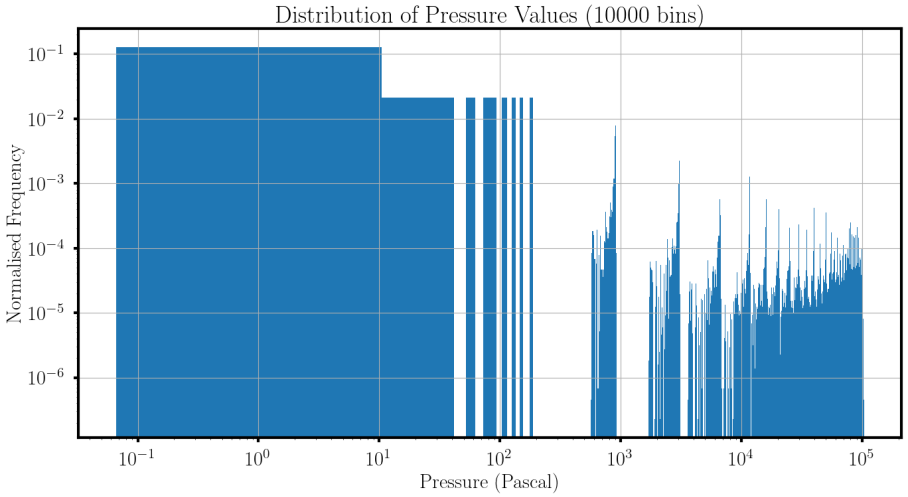
(b)


Distribution of East-West Winds
Normalised Frequency
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
−60
−40
−20
0
20
40
60
East-West Winds (m/s)

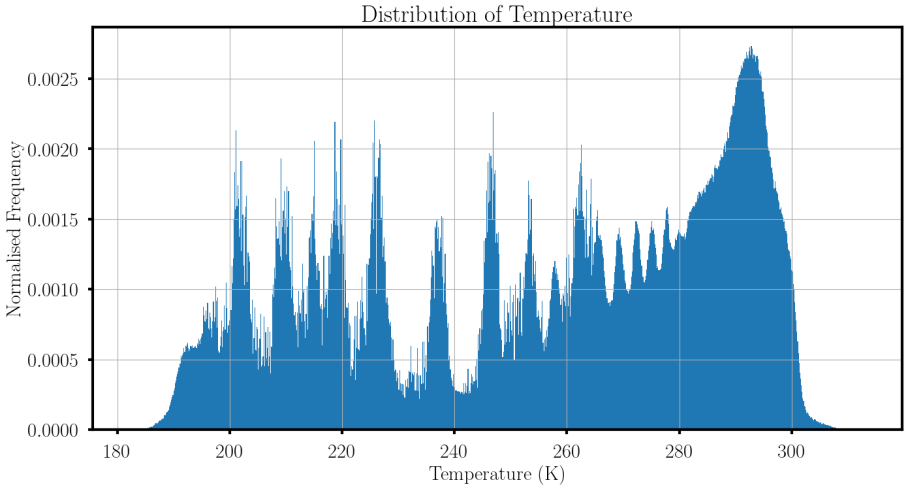
(c)


Distribution of North-South Winds
Normalised Frequency
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−20
−10
0
10
20
North-South Winds (m/s)

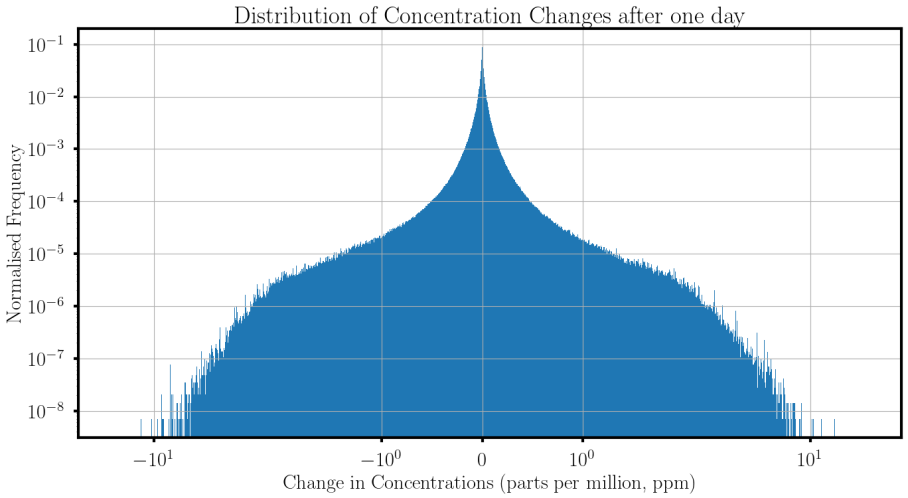
(d)



(e)



(f)



(g)

Figure 4.6: Distributions of the original values of the input and output quantities used in the training of the model

3. U-Net was designed for image segmentation; moreover a large number of deep learning applications involve images where the maximum number of input features is 3 (Red, Green, Blue). In our case however, even after simplifying the problem substantially (as explained in previous sections), we have 6 features; emissions, initial concentrations, east-west winds, north-south winds, pressure and temperature. In more realistic versions of this problem, the number of input features will increase substantially. A large input demands the whole network to become more complex which makes the training more difficult.

4. As mentioned earlier, for every input our network is supposed to predict ~ 10000 values over a 3D grid. Hence this is a regression problem over about 10000 variables, which will naturally require a model with high complexity. From our initial experiments with neural networks with $\sim 100K$ parameters, it was clear from the high training errors that the data was being under-fitted. It was only when the network size was increased to the order of 10M parameters that we started to see good performance on the training set. From this trend, we expect the model to perform better with even more tunable parameters. With increasing complexity of the network, the maximum size of the minibatch we can use decreases, since the available GPU memory is limited and the optimizer can only process so much training data along with a very complex model. A small batch-size takes more time in training, so with limited GPU resources, we have to balance the model complexity with the batchsize and the number of epochs and it may not be possible to get the best performance from our model.

5. A related constraint arises due to the fact that we are using 3D convolution layers unlike in many computer vision problems where 2D convolution layers are used. Training with 3D convolution layers is much slower than 2D; for an input of size $w \times h \times d$ and kernel $k \times k \times k$, 3D convolution layers have to perform $k \cdot d$ times more calculations than the corresponding 2D case, assuming the number of channels are the same. Typical value for k is 3 and d is 47 for our case, so the network has to make ~ 140 times more computations. In fact we can expect that the 3D case would require more number of channels in order to learn the features in 3D data, making the model more complex. These factors again limit the model complexity and number of epochs we can run the training for.

6. Our *regional* training data comes from *global* simulations, so while the latter maintains the conservation of mass for the gas, our trained neural network will not

be able to do so. This is not very concerning, since in principle if we use this regional model on 16×16 patches over the whole globe, as a whole the gas mass will be conserved. However the actual problem is that in our 16×16 horizontal grid, the concentrations in the outer cells (close to the edge) will be affected by the conditions (ie concentrations, winds etc) outside of this grid also. But since those conditions are not included in this particular training sample, the network simply wont be able to predict the concentrations for these outer cells with as much accuracy as the inner cells.

In the light of the above constraints, below are all the details about the training of our network:

1. Input and Output rescaling:

   (a) Emissions: Since most emission values lie between $1e^{-11}$ and $\sim 1e^{-8}$, we use **log scaling** followed by normalisation by a factor of 3. This is done while preserving the signs (so that there is a clear segregation of sources and sinks). The values were then re-centered to 0.5 to bring most of the values in the range [0,1].

   (b) Winds: Most east-west winds lie between -15 and 15 and north-south winds between -10 and 10, So we simply scaled the values down by 15 and 10 respectively and then recentered them to 0.5.

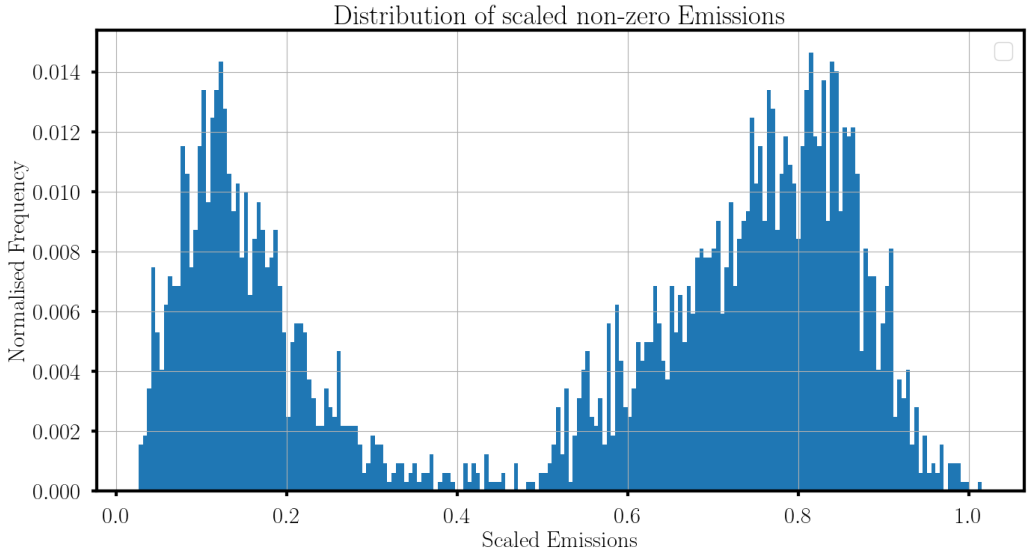
   (c) Pressure: After some experimentation, we arrived on setting the cells with pressure less than 100 to 0, and then using log scaling followed by normalisation by 3.

   (d) Temperature: we used the simple *min-max scaling* which goes as:

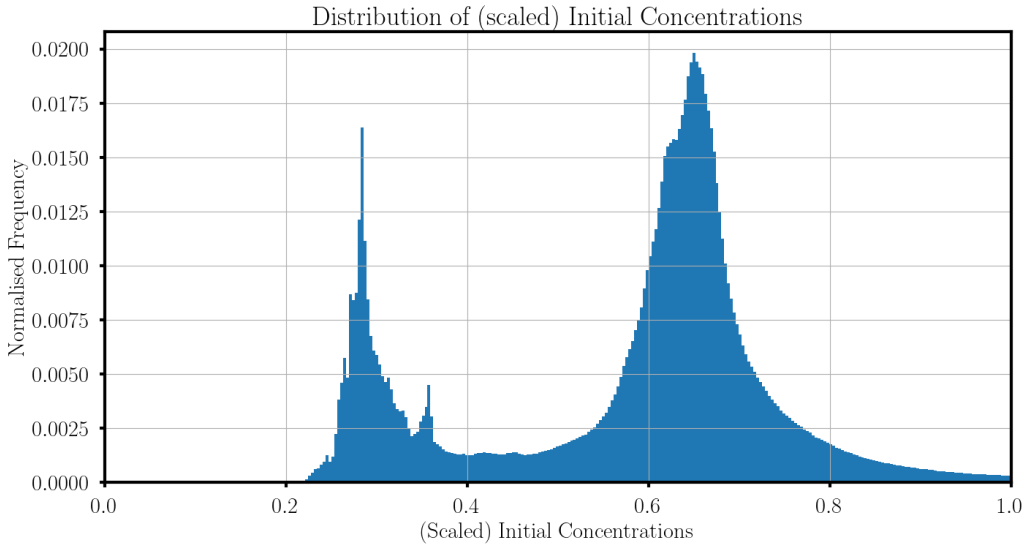
   $$T_{scaled} = \frac{T - T_{min}}{T_{max} - T_{min}} \tag{4.1}$$

   This scales the whole range to [0,1]. For T_{max} and T_{min} we used 310 and 180 K respectively.

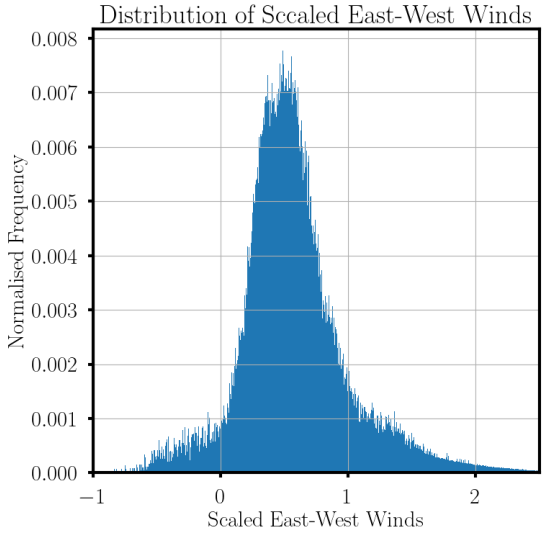
   (e) Target Output (change in concentration): After some experimentation with linear scaling, we found that by simply multiplying by 10.0 and recentering to 0.5 gives relatively better results.


Distribution of scaled non-zero Emissions
Normalised Frequency
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0.0
0.2
0.4
0.6
0.8
1.0
Scaled Emissions

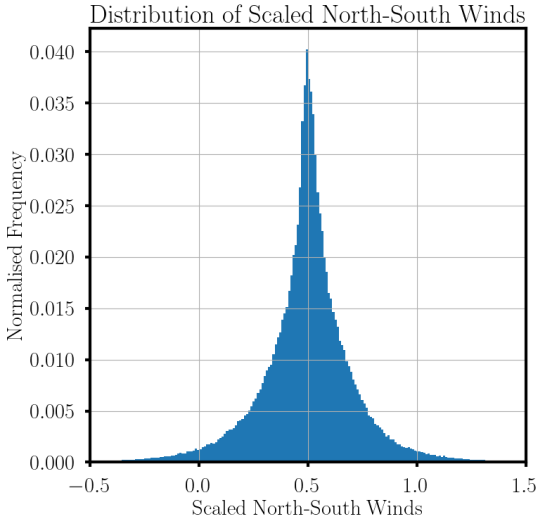
(a)


Distribution of (scaled) Initial Concentrations
Normalised Frequency
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0.0
0.2
0.4
0.6
0.8
1.0
(Scaled) Initial Concentrations

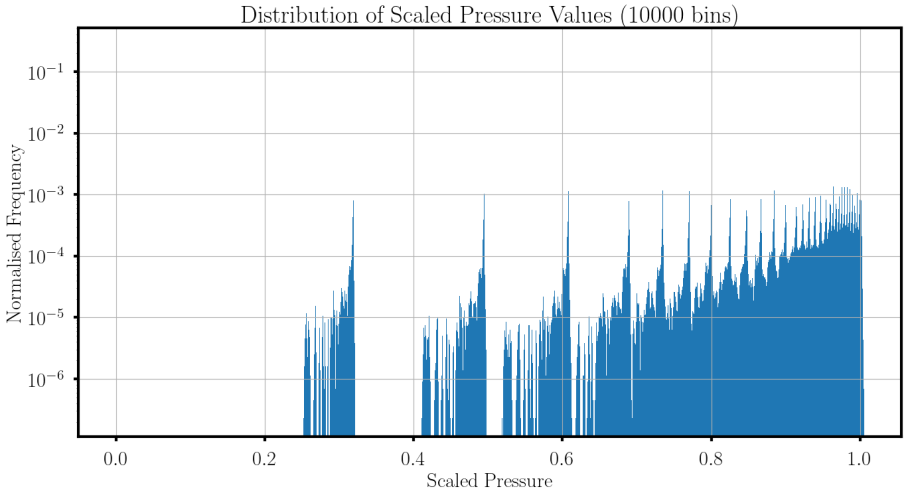
(b)


Distribution of Sccaled East-West Winds
Normalised Frequency
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
−1
0
1
2
Scaled East-West Winds

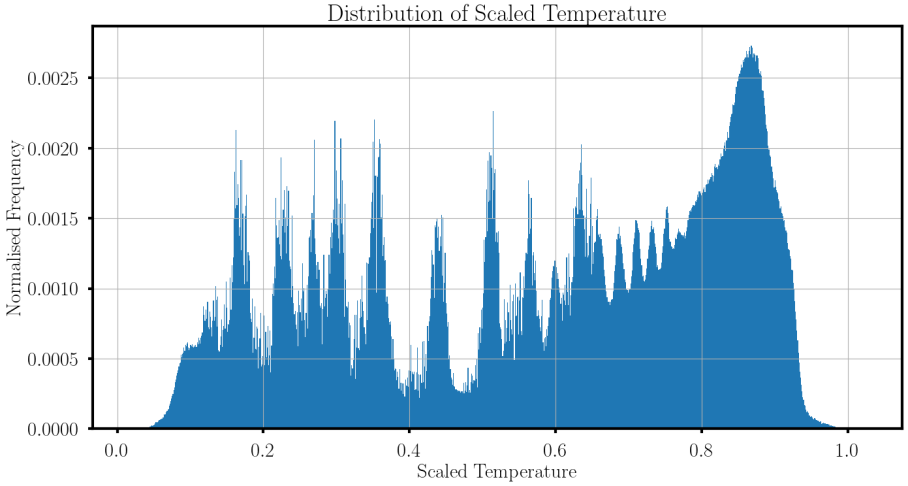
(c)


Distribution of Scaled North-South Winds
Normalised Frequency
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−0.5
0.0
0.5
1.0
1.5
Scaled North-South Winds

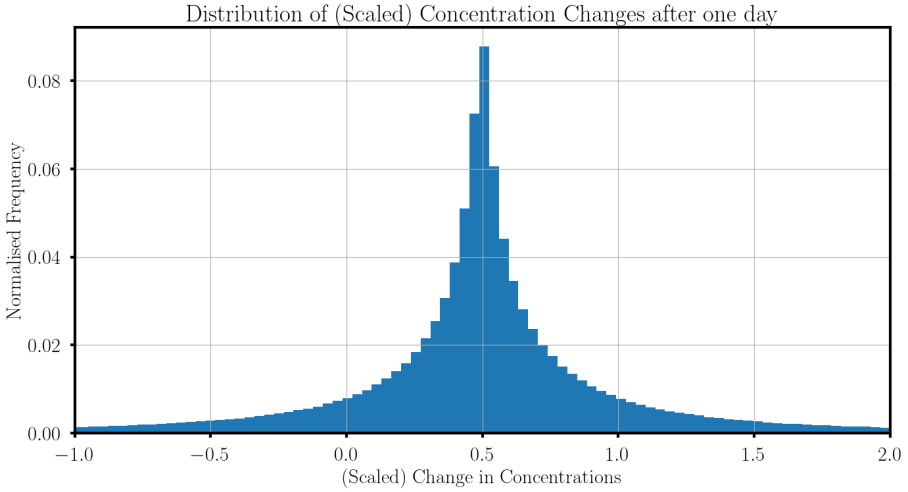
(d)



(e)



(f)



(g)

Figure 4.7: Distributions of the **scaled** values of the input and output quantities used in the training of the model

See Fig 4.7 for the distributions of the features after the above re-scaling.

2. Convolution Kernel Size: Naturally, the concentration at a cell is affected the most by its adjacent cells. In the first hidden layer, we would like the network to capture the most immediate/primitive features in the data, just like in a computer vision problem. So we used the usual kernel size of $3 \times 3 \times 3$. The effect of cells further away is not lost however, since the field of influence keeps increasing because of the convolutions in the subsequent hidden layers.

3. Boundary Conditions: Most horizontal winds have magnitudes below around 15 m/s (Fig 4.6 (c) and (d)), so in 24 hrs, a cell will be affected mostly by concentrations about 1300 km away from itself in any direction which corresponds to the distance covered by 3 cells for our grid resolution. So while the output of the network is of the shape $47 \times 16 \times 16$, in order to account for the lack of boundary conditions, we used the values only in the central $47 \times 10 \times 10$ cells for the computation of the loss function and hence for the training.

4. Loss function: If the correct concentration changes (over the 3D grid and inside the 10×10 horizontal region) are T and the corresponding predicted concentration changes are P, then the loss function we used is:

$$Loss = \frac{\Sigma(T_i - P_i)^2}{\Sigma T_i^2} \tag{4.2}$$

   so essentially its a ratio of the *mean squared error (MSE)* in the *change* in concentrations divided by the mean squared value of the target concentration change. For monitoring training performance, we used $\sqrt{Loss}$ averaged over all the data in the epoch. This quantity gives us an idea in ratio about how the magnitude of the RMS error in the *change in concentrations* compares with the average value of correct change in concentration. So for a loss of 0.21 we can say 'in terms of the change in concentrations, the model's accuracy is 21%'.

5. Model Validation: In the generation of our training data, each simulation gave rise to 365 samples; one sample belonging to each day. We picked 61 days in the year at random and used all samples belonging to those days for the *validation* of our model. Samples belonging to the rest of the days were used for training. In each epoch, apart from the model updates from the *training loss*, we computed the *validation loss* over these samples. Since 4 out of 6 input features are totally

different for different days, this would be a strong test for the generalisability of the model.

In addition to the validation loss, we also computed another metric to gauge the performance of the model. For a validation sample, if IC is the initial concentrations, P is the predicted change in concentrations and TC is the target concentrations, then the *Root Mean Square* of $TC - (IC + P)$ is the average error in the *predicted concentration* and this is what we also monitored during training.

6. Batch Size: We experimented with batchsizes ranging from ~ 150 to 2. Intuitively, a very small batch is not considered good for learning and even the benefits of Batch Normalisation rely on an appreciable batch-size. However we found batches of size 4 to be giving relatively better results than all other alternatives. We think that part of the reason could be that here the features are 3-dimensional; even if the batch size is small, the number of neurons/nodes affected by a parameter/weight is much larger owing to the higher dimensionality, so the stochasticity in the gradient is reduced. Similarly, since the mean and variance in batch normalisation are calculated over the whole 3D feature, the extra dimension might be helping in making the calculation statistically effective without a large batch size.

7. Software and Hardware: We used the `PyTorch` framework, version 2.1.2 to build the deep learning model and for the analyses. The Python version used was 3.10.13. All training was performed on a `Nvidia T4` GPU with 15 GB available memory.

8. Miscellaneous:

   (a) Model Size: ~ 7.6 million parameters

   (b) Model weights initialisation using **Kaiming-normal/He Initialisation** [33] where the initial weights are sampled from the normal distribution $\mathcal{N}(0, \sigma^2)$ where

   $$\sigma = \sqrt{\frac{2}{n}} \tag{4.3}$$

   where n is the number of inputs to the node in question. This initialisation method has been shown to avoid reducing or magnifying the magnitudes of input signals exponentially and leads to better convergence.

(c) Activation function: ReLU followed by Batch Normalisation after all Convolution Layers and concatenation in both skipconnection steps.

(d) Optimiser: `Adam`

(e) Learning Rate: 0.001

Chapter 5

Performance Analysis

5.1 Measures of Accuracy

We trained the model for 50 epochs which took about 5 hours. The training loss, validation loss, and the validation metric (average error in predicted concentrations in ppm) as defined previously are shown in Fig 5.1 below.

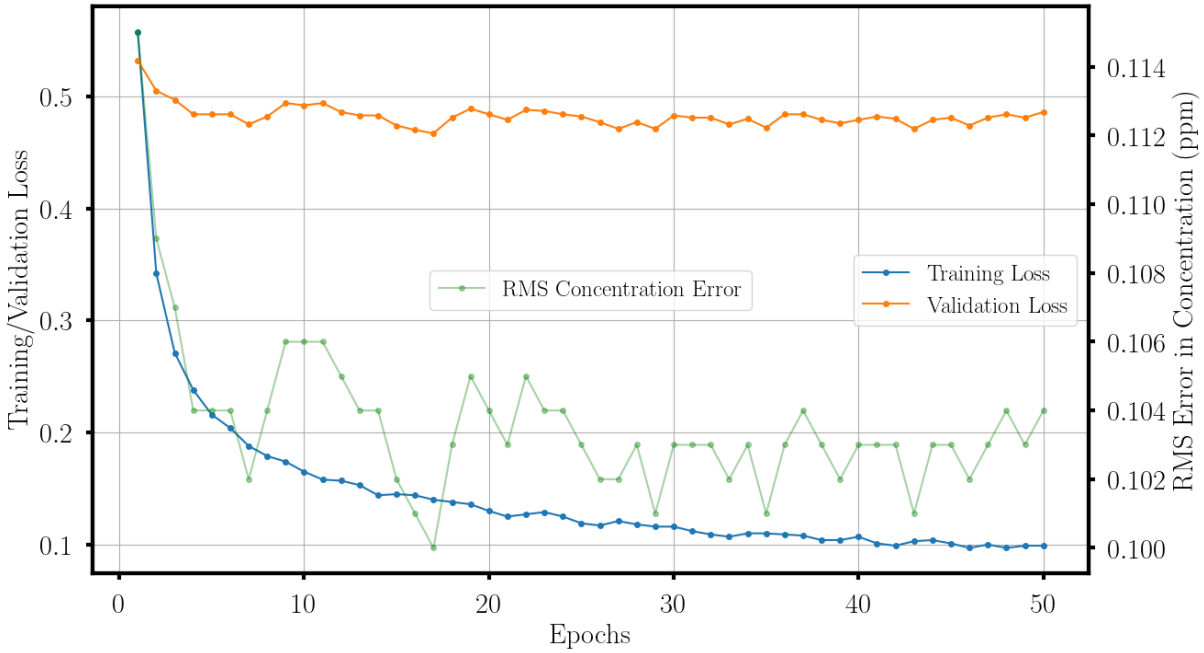


Figure 5.1

The values of the 3 quantities after 50 epochs were 0.107, 0.479 and 0.104 respectively.

Undoubtedly, the model's performance is much worse than expected. While the training loss keeps decreasing, the validation loss achieves its minimum value in the 7th epoch itself and stays around that. At its best, on validation data the model predicts with an error of 46.7%. Please note that this is for the *change* in concentrations and the loss is computed relative to the actual values of concentration change. If we look at the error in predicting the concentrations themselves, it is about 0.04%. This is simply because the average magnitude of the change in concentrations is smaller than the average concentrations by an order of 3 or more. Still, low validation accuracy implies the model has difficulty generalizing to the data well, that is it is over-fitting. We attempted ways to deal with the over-fitting (described in section 5.4) but out of all the many alternatives we tested, this is the best performance we could get from a model. However, for this kind of a difficult regression problem this is nevertheless a step forward, as we will see by looking at the performance from different perspectives.

For example, we have plotted the individual predicted concentrations against the 'correct' concentrations in Fig 5.2.

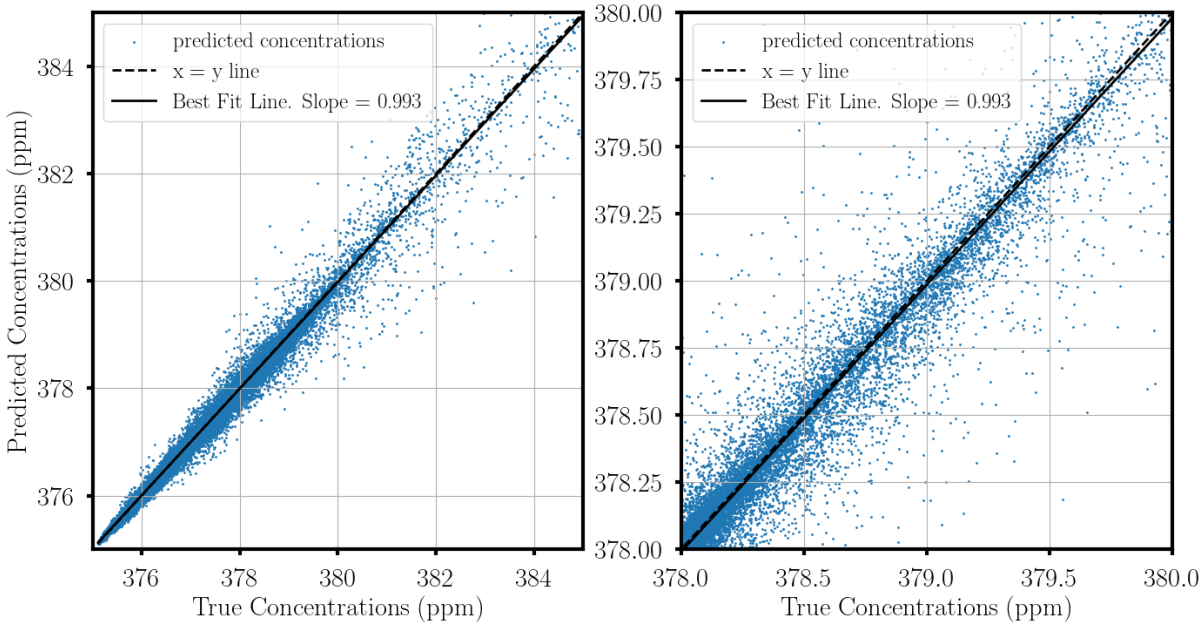


Figure 5.2: Scatter plot of the predicted concentrations against the true concentrations

The best-fit line has slope 0.993; the perfect model would have had a slope of 1. For comparison the $x = y$ line is also shown. In the interval [375, 380], the model does better than concentrations > 380 ppm, which is because they are under-represented. We can get a better look by plotting the predicted *change* in concentrations against the

'correct' values (Fig 5.3).

Here the best-fit line has slope 0.624. In the left plot in Fig 5.3(a), we can see that a small fraction of points have a large target concentration change and a correspondingly high deviation in prediction. These points bring the average measures of accuracy further down. When we consider only the values between -1 and 1 ppm, the slope improves to 0.655 (plot to the right in Fig 5.3(a)). In Fig 5.3(b) the same data is shown but the colors correspond to the density, showing the sheer domination of concentration change between -0.05 and 0.05.

We also computed the *Pearson correlation coefficient (r)*, defined as:

$$r = \frac{\Sigma(f_i - \bar{f})(y_i - \bar{y})}{\sqrt{\Sigma(f_i - \bar{f})^2 \cdot \Sigma(y_i - \bar{y})^2}} \tag{5.1}$$

where for our case, f_i's are the model predictions, y_i's are the actual values and $\bar{f}$ and $\bar{y}$ are their corresponding means. Between the predicted and true concentrations, r was 0.996 and for the change in concentrations it was 0.885.

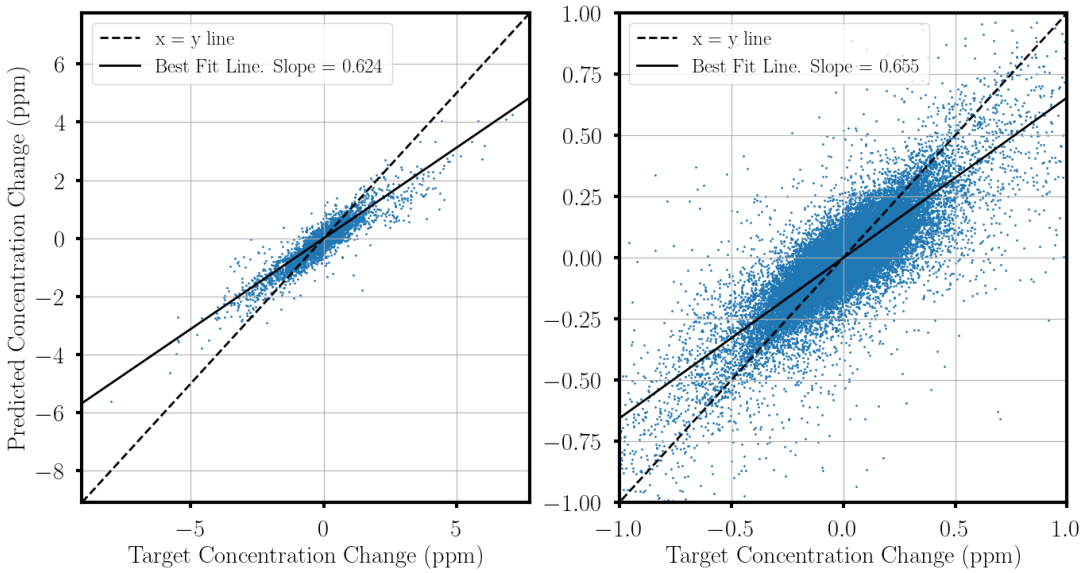
The *coefficient of determination (R)* can be a better indicator of how well the model explains the target data, by calculating the proportion of the latter's total variation explained by the model's predictions:

$$R^2 = 1 - \frac{\Sigma(y_i - f_i)^2}{\Sigma(y_i - \bar{y})^2} \tag{5.2}$$

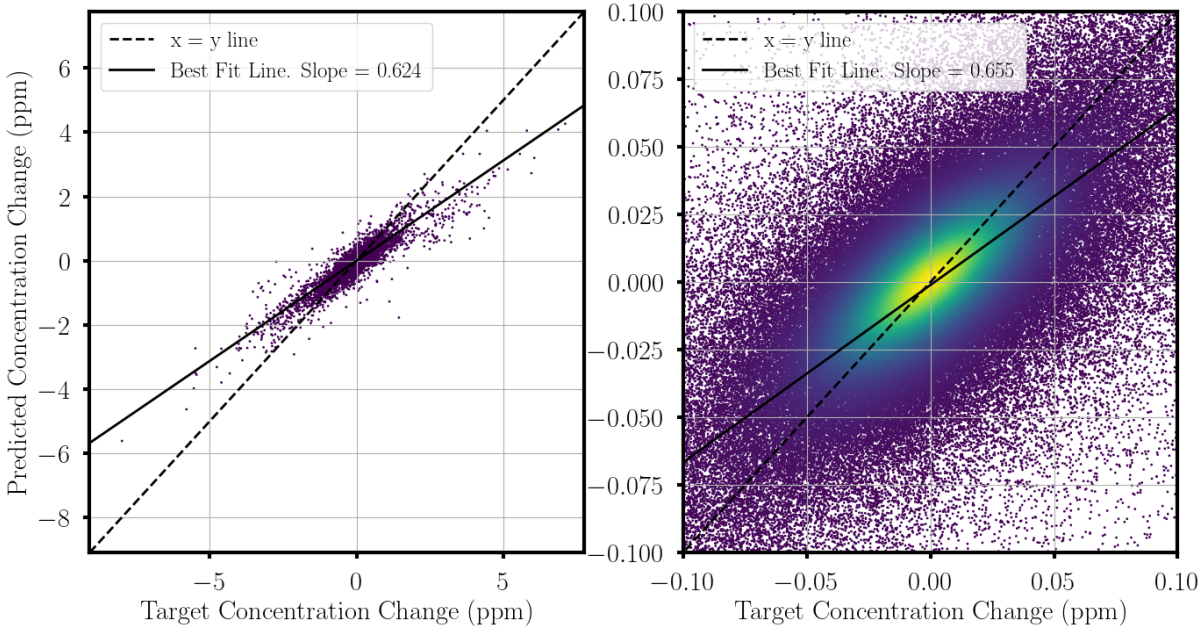
Adjusted for the number of predictors in the model it is given by:

$$\bar{R}^2 = 1 - (1 - R^2) \cdot \frac{n - 1}{n - p} \tag{5.3}$$

where n is the number of samples and p is the number of predictors. The adjusted coefficient of determination was 0.992 for concentrations and 0.751 for change in concentrations. The various measures of performance of our model are summarised in the table below. Our model is able to explain 75.1% of the variance in the data (which comprises of a huge number of values, of the order 10^6). Overall all the different measures of accuracy we have discussed illustrate the fair bit of learning the model has done, but at the same time there is still a significant gap in accuracy that it hasnt



(a) Scatter plot of the predicted concentration *change* against the true concentration change



(b) Density Scatter plot of the predicted concentration *change* against the true concentration change

Figure 5.3

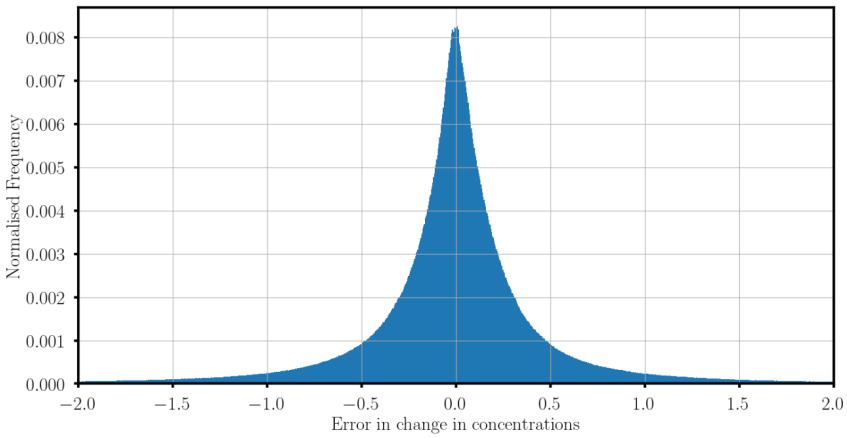
bridged.

Quantity	**Concentrations**	**Change in Concentrations**
Training Loss (after 50 epochs)	0.000089	0.107
Validation Loss (after 50 epochs)	0.0004	0.4794
Average Error in Concentration Change (ppm)	0.104	NA
Slope of Best Fit Line	0.993	0.624
Slope of Best Fit Line (zoomed)	NA	0.655
Pearson Correlation Coefficient	0.996	0.885
(Adjusted) Coefficient of Determination	0.992	0.751

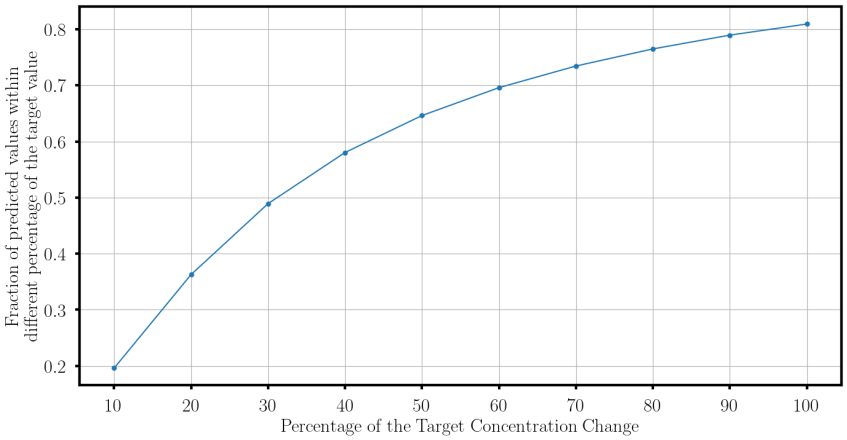
5.2 Distribution of Loss

While the validation loss is averaged over all the individual loss values, it can be helpful to look at the distribution of all the individual errors in the model's predictions of the concentration changes, shown in Fig 5.4(a). The primary takeaway from this is that the model neither leans towards overestimating the change in concentrations, nor underestimating it. It should also be noted that while there are errors as large as 2 ppm, about 99.6% of errors lie between -0.5 and 0.5. We can take a closer look by plotting the proportion of predictions that lie within different error percentages of the true values. This is shown in Fig 5.4(b). We see that while about 65% of predicted values (change in concentration) have a 50% error or less, there are still more than 15% of predictions with more than 100% error. These correspond to the cases where the actual change in concentration is of the order 0.001 or less and we know there is a good proportion of these.

We can also look at the variation of the (validation) loss along different axes, namely the horizontal direction, the vertical direction and also with different days of the year. Our model was trained on meteorological data corresponding to 304 randomly chosen days of 2019 and for our analyses we have been using the rest of the 61 days. In order to compute the variation of the validation loss for all days of the year, we generated some



(a) Distribution of the error in the values of individual predictions of change in concentrations (ppm)



(b) Proportion of predicted values with a given percentage error and below

Figure 5.4

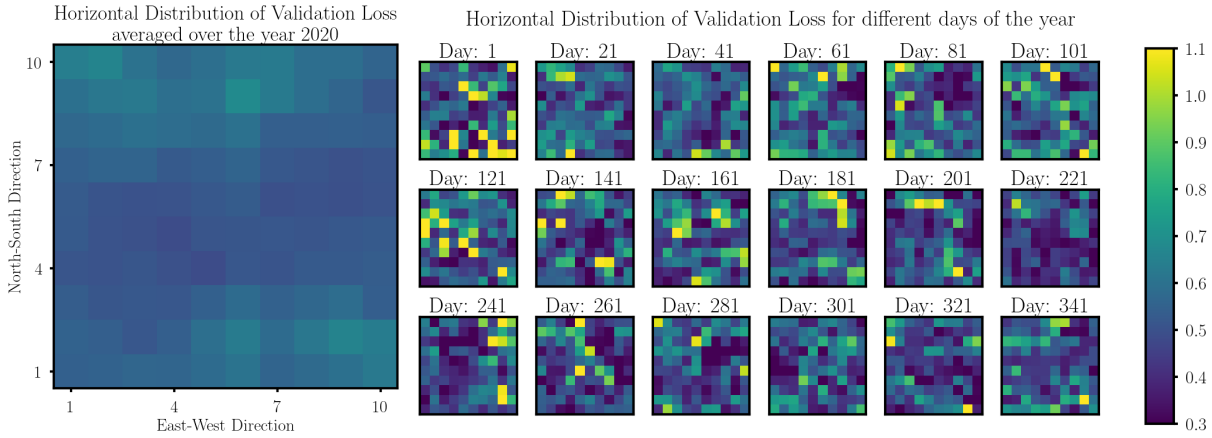
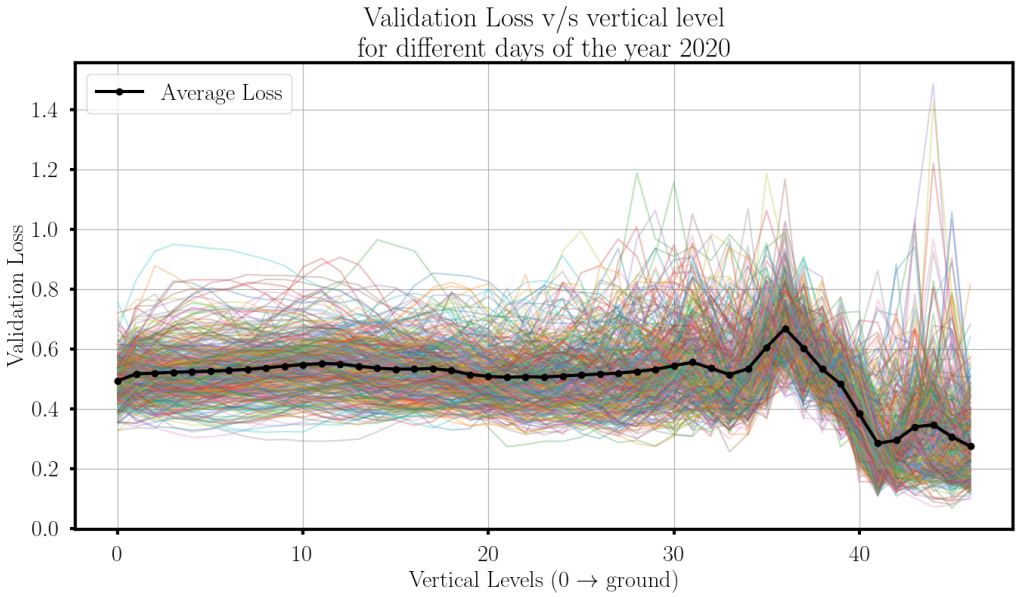
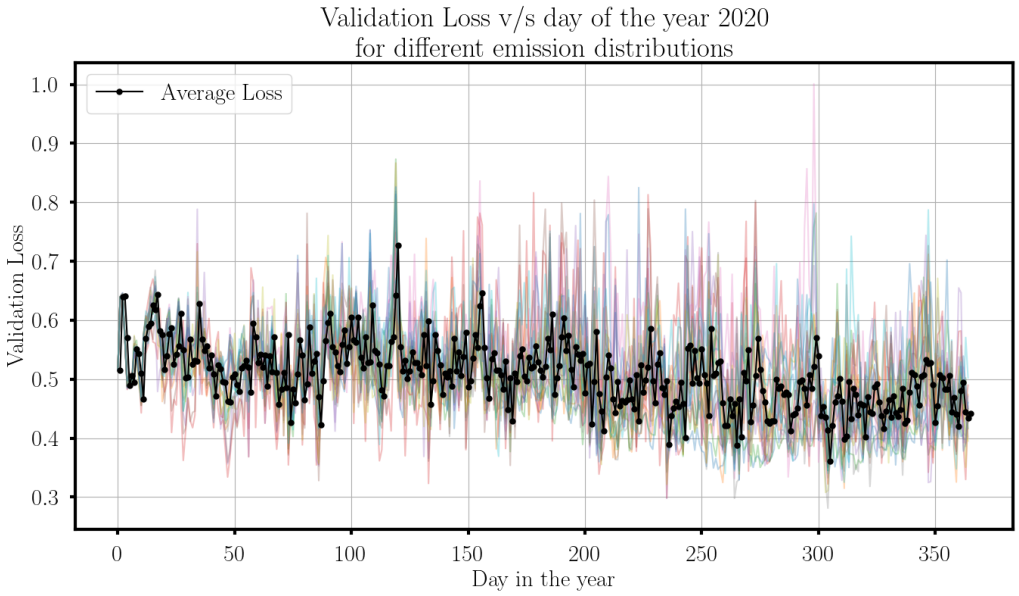


Figure 5.5


Validation Loss v/s vertical level
for different days of the year 2020
Average Loss
Validation Loss
Vertical Levels (0 → ground)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
10
20
30
40

(a)


Validation Loss v/s day of the year 2020
for different emission distributions
Average Loss
Validation Loss
Day in the year
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0
50
100
150
200
250
300
350

(b)

Figure 5.6

additional data for the year 2020. All the meteorological data of 2020 is new for the model, so we can compute the model loss for all 366 days. In each case below, the loss is averaged over all other axes of variation except the axis being considered.

- **Horizontal Direction**. The distribution of the loss along the horizontal direction averaged over all 47 vertical levels, 366 days, and all emission distributions is shown to the left in Fig 5.5. To the right, the distributions for some of the individual days are shown. While the individual days have their large maxima and minima ranging from 0.3 and 1.1, their average reduces to the range between about 0.45 and 0.6. Apart from some minor structure in the average distribution (present because the model can't predict with the same accuracy for all horizontal positions), we can confirm that it is more or less uniform as we would expect since the lack of boundary conditions has been addressed.

- **Vertical Direction**. The variation of the loss/error along the vertical direction averaged over the horizontal 10×10 plane, 366 days, and all emission distributions is shown in Fig 5.6(a). The loss averaged over all the days of the year remains stable around 0.5 but only till level 33 after which we see a rise in the loss, followed by a large reduction after level 36 till the 42nd level. A possible explanation is given below.

  Upon looking at the distribution of the winds along different vertical levels we found that the widest distributions lie between the levels around 33 and 37. These levels correspond to 13.6 km and 19.8 km altitude respectively. Starting from level 28 as we go higher, the effect of the **sub-tropical Jet Streams**[1] increases, resulting in stronger winds for increasingly more days of the year as we reach level 33. After the 40th level however, we are high in the Stratosphere[2], where we found that, the distribution of the winds is the narrowest among all the vertical levels, for *most* days, while for few other days (corresponding to the cold months), the distribution was the *widest* among all vertical levels.

  Before giving the east-west winds as input to our network, we rescale them by a factor of 15. This means that altitudes where the winds are particularly strong (> 45) (or equivalently where the distribution of winds is wider), the scaled winds will be much outside the ideal range of [-1,1]. This might be causing the model to

[1] these are fast flowing air currents, found at sub-tropical latitudes at high altitudes
[2] Tropopause in the equatorial regions is at about 17 km altitude

have a large error for the vertical levels between 33 and 37 and for the cold days for vertical levels above 41, especially since the data doesn't have a good representation of such strong winds. Similarly, the particularly weak winds above the 41st level (for the warm/hot months) allow almost all the wind values to be between -1 and 1 after rescaling, resulting in errors even lower than vertical levels between 0 and 30.

- **Day of the Year**. The variation of the loss with the day of the year (2020) averaged over the horizontal 10×10 plane, 47 vertical levels, and all emission distributions is shown in Fig 5.6 (b). Apart from a handful exceptions, most of the days correspond to loss between 0.4 and 0.6. The most notable feature is the large errors for some emission distributions between the days 120 (May) and 300 (end of October) which correspond to the summer and the monsoon (south Asia) seasons. These months are generally characterized by stronger winds compared to other months. During these months, emission distributions having major emissions in regions of particularly strong winds would result in a high error since such large change in concentrations are not represented well in the data as a whole. However, they are averaged out by other emission distributions that are not affected by the strong winds. In fact on first look, there seems to be a downward trend in the loss, however it is hard to confirm due to the large variance and large upward and downward movements.

5.3 Permutation Feature Importance

Permutation Feature Importance is a technique that can be used to evaluate the importance of input features in a wide range of models including deep learning. It's particularly useful for understanding the relative importance of different input features in making predictions. For a given feature it is calculated by first permuting the feature values between the validation samples, keeping all other features and output in their correct place. Then the predictions of the network are obtained on these modified samples. Permuting the values of a feature effectively disrupts the relationship between it and the target variable, resulting in a higher loss. The larger the increase in the loss compared to the 'original' loss, the higher is the *importance* of the feature and vice versa. We calculated the permutation feature importance of the 6 input features:

emissions, initial concentrations, east-west winds, north-south winds, pressure and temperature. The results are shown in the table below and Fig 5.7.

Feature Name	**Permutation Feature Importance (increase in validation loss)**
Emissions	0.11286
Initial Concentrations	0.86446
East-West Winds	0.59416
North-South Winds	0.44486
Pressure	0.06229
Temperature	0.06226

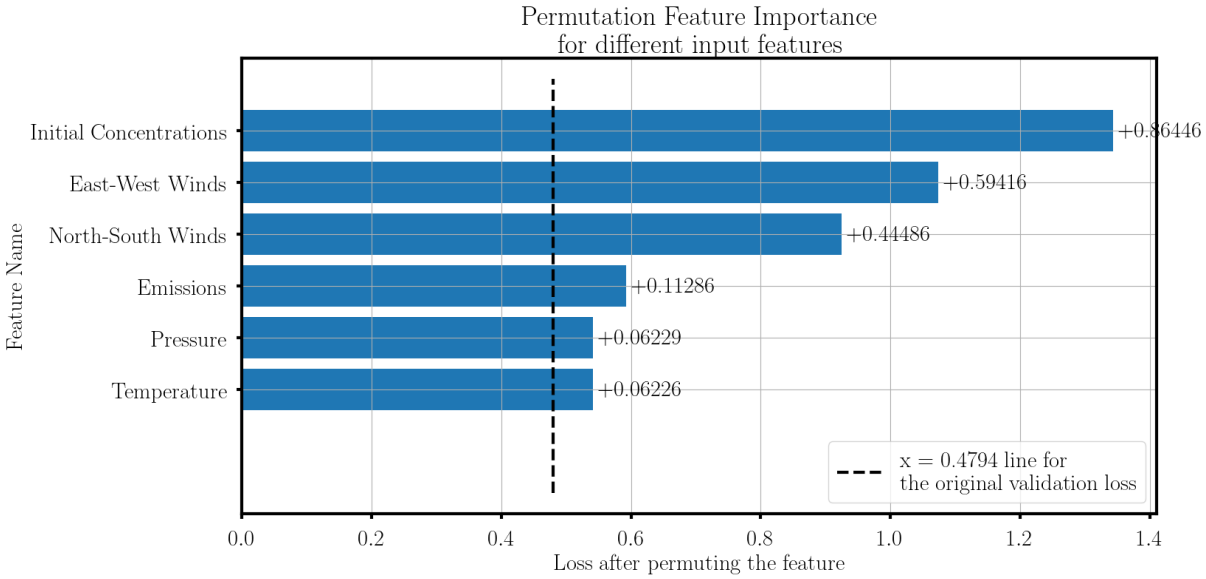


Figure 5.7: Permutation Feature Importance. The bars height shows the validation loss after permutation of that particular feature. The value on top of the bars is difference between the new loss and the original loss before the permutation ie 0.4794

As one would expect, the initial concentrations given to the model is the most important feature by far in predicting the change in concentrations one day into the future; giving the wrong values by permutation results in 86% more error. The horizontal winds are the second most important which is also intuitive since advection is the only process we have in our data and so (apart from emissions) it solely determines the time evolution of the gas. The east-west winds have higher importance than the

north-south winds simply because the former has stronger winds and hence higher impact on the output. Pressure and Temperature indeed have some bearing on the accuracy of the output; permuting them does lead to about 6% more error. The increase is small because their role in affecting the concentrations is indirect and relatively less.

However, emissions seem to have much less importance than what one would expect, knowing that it is responsible for adding more CO_2 in the air and incorrect emissions should result in a much higher loss. There are 2 possible explanations of this: 1) We are counting it as a separate feature, but emissions are still only 2 dimensional, while all other features are 3D, the third dimension being vertical height. Hence, emissions have 47 times less values and simply because of that, the impact of permuting emissions on the accuracy will be reduced significantly. 2) Secondly, the emission distributions in our data range mostly from 10^{-11} to about $10^{-8}\, kg/m^2/sec$, the distribution leaning more towards the lower values. These emissions are relatively low in magnitude to cause a very large effect for a one day prediction horizon, so their 'importance' would also be reduced.

5.4 Experiments for Model Improvement

In order to improve the performance of our model, we attempted to address the most pertinent issues with our network/data/training-strategy. The details for each experiment are provided below:

1) Regularisation

Since our model shows signs of overfitting (Fig 5.1), our first modification was using **L2 Regularisation**. In this regularisation method, the loss function involves an additional additive term that is proportional to the sum of square-norm of all the parameters of the model. So if the original loss function was $L(w)$, then the new loss function $L'(w)$ is:

$$L'(w) = L(w) + \lambda \Sigma_i w_i^2 \tag{5.4}$$

This is a popular regularisation technique to reduce over-fitting. With this modification and keeping everything else the same, we trained our model again. The training and validation loss v/s epochs are shown in Fig 5.8. While the training error is kept from reducing rapidly, we don't see an improvement in performance on the validation data; the loss stagnates at 0.50-0.51. Thus, a straightforward regularisation did not help in improving the performance.

2) Number of Training Samples

Seeing the model's difficulty in generalising, one would suspect that the number of data points is not enough. Indeed for the model presented above, the number of training samples is of the order $\sim 100K$ while the model has 7.6 Million parameters, a gap of almost 2 orders of magnitude (though it should be noted that each training sample involves regression over 4700 values, so each sample carries a lot of information). To bridge this gap, we did the following:

- The network was modified so that it takes all input features of horizontal extent 7×7 (instead of 16×16) and 47 in the vertical direction as before. The network

output was of dimensions $47 \times 7 \times 7$, but for the calculation of the loss, only the change in the concentration in the central cell, ie the (3,3) cell, was considered (to account for the lack of boundary conditions as before) for all 47 vertical levels. The other parameters of the network were unchanged so the number of model parameters remained the same.

- We increased the number of samples by taking portions of size 7×7 from our original data of size 16×16. So every sample from the previous dataset gave rise to 100 unique samples. This gave rise to a dataset of size ~ 3 million samples.

With the dataset size now matching with the model size, we trained the network with the same settings and hyperparameters. Because of the large dataset, training for just 11 epochs took about 18 hours 40 minutes. However, as shown in Fig 5.8, here too the validation loss stagnates at 0.50-0.51. While the data set size should always be as large as possible, it doesn't seem to be the limiting factor in our current attempts for better accuracy.

3) Scale of output values

As we saw in section 4.3, the output values (change in concentrations) range from the order 10^{-4} to 5 (Fig 4.4(g)) and in the model presented above, we have simply multiplied them by 10. This doesn't solve the problem of the values spanning over 4 orders of magnitudes. Its unreasonable to expect the model to cover this wide range of values. Moreover, most of the values lie between 0.001 and 0.1 so the loss will be high regardless of whether the model tries to focus on the small values or the large values (between 0.1 and 10). So we tried log-scaling the values and further squeezing them in the range $[0, 1]$ like the input features. With the modified target outputs, we trained the network and the performance is shown in Fig 5.8. It is significantly worse than before. We get a sense of whats happening by looking at the validation 'metric' used before: average error in the predicted *concentrations in ppm*. The metric fluctuates wildly from 14 initially to about 126552 and goes beyond the numerical limit (referred as `'inf'`). This is because squeezing 4 orders of magnitudes between 0 and 1 makes the validation loss and metric extremely sensitive to the output values, where a small change in a parameter (and hence in the output) brings a large change when converted to the actual ppm scale. Hence, the issue of the output being spread across many scales requires a

more clever solution.

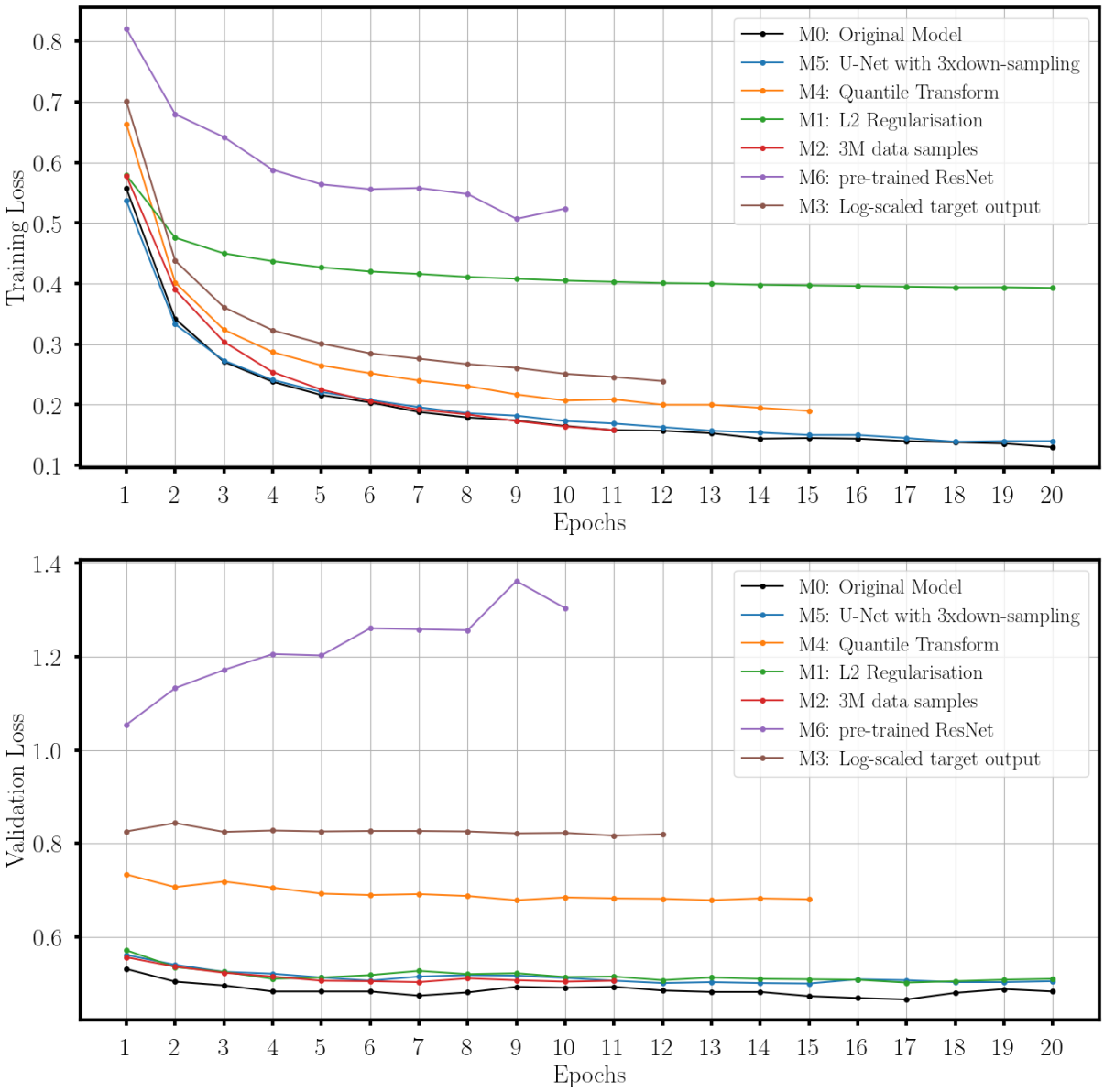


Figure 5.8: Training and Validation loss v/s epochs for the different alternative models tested. The losses for the original model are shown in black till 20 epochs. For some models the training was stopped pre-maturely since the validation loss had practically stopped decreasing by then. The model presented in Section 5.1-5.3 performs relatively better than all alternatives

4) Distribution of Input Features

Apart from the issue of the vast differences in the scales of the inputs and the outputs, another potential problem with our data is the considerable differences in and the non-trivial shapes of the distributions of various input features and the output. Features with different distributions can lead to gradients of different magnitudes for the same change in the parameters. Even if the variances/scales are the same, the scale of the gradients can vary due to the different shapes of the distributions. This can result in imbalanced updates during training, where parameters associated with features from certain distributions are adjusted more rapidly than others, potentially leading to slower convergence or oscillations during optimization. To address this, we attempted to modify the distributions of our input features using the **(i) Power Transform** and the **(ii) Quantile Transform**.

(i) A power transform is a statistical technique used to stabilize the variance and make the data more *like* normal distribution. The data transformation is done using a generalised power function of the form-

$$x'^{(\lambda)} = \begin{cases} \frac{((x_i+1)^{\lambda}-1)}{\lambda} & \text{if } \lambda \neq 0, x \geq 0 \\ ln(x_i+1) & \text{if } \lambda = 0, x \geq 0 \\ \frac{-((-x_i+1)^{(2-\lambda)}-1)}{(2-\lambda)} & \text{if } \lambda \neq 2, x < 0 \\ -ln(-x_i+1) & \text{if } \lambda = 2, x < 0 \end{cases} \tag{5.5}$$

and the λ that maximises the 'normality' of the transformed data is estimated. This particular transform is called the **Yeo–Johnson transformation**. We tried to apply this on the original values of the 6 input features in our data. In short, this transformation did not help with the most non-trivial features in our data like the pressure and emissions, while for the 2 wind features it increased the normality somewhat (however these were the 2 least problematic features to begin with).

(ii) We also tried applying the Quantile Transform, which unlike the Power Transform is not a parametric transformation. In this, first the cumulative distribution function (CDF) of the data is estimated then the values are transformed to a different distribution by one-to-one mapping using the latter's inverse CDF. We transformed all of our 6 input features to the standard normal distribution, $\mathcal{N}(0,1)$ and divided the values by 2 to bring most points between -1 and 1; we didn't transform the output

because of the previous experiment. Unlike the Power Transform that struggled to improve the normality of some features like pressure, the Quantile Transform was able to transform all of the features into visually-obvious normal distributions.

With this transformation we trained the model, but once again the performance was much worse (Fig 5.8) compared to the data-rescaling methods we described in Section 4.3. It seems that this transformation was too strong; some features of the original distribution of the data need to be preserved in our rescaling methods.

5) Alternative Architectures

(I) More Down-sampling

Down-sampling (and the subsequent up-sampling) is meant to help the model learn to generalise to unseen data. The more the down-sampling, the more difficult it is for the model to over-fit on the data (to a limit of course). So we tried training a modified version of the network architecture used until now, where the input is downsampled 3 times instead of 2. After the 3rd down-sampling each channel has dimensions $6 \times 2 \times 2$. However, it didn't perform better than the 2-times-down-sampling model, having a slightly higher loss (Fig 5.8).

(II) Using Pre-Trained Network

Transfer learning is a deep learning technique where a model trained on one task is reused or adapted for a model on a second related, or even unrelated task. Eg a model trained for classification of everyday non-medical images can be applied for medical image segmentation [34]. Moreover, a model trained for classification can also be used for regression tasks.

For applying transfer learning to our problem, we used a modified version of the `ResNet` architecture. ResNet is an architecture that was designed for image classification [35]. For classification of videos, using 3D convolution in ResNet is an intuitive extension. However, the `'R(2+1)D'` variation has been found to perform better and be more efficient for classification of Action Recognition [36]. Instead of 3D convolution, in this

variant it is decomposed into first a 2D convolution step along the *width* and the *height* dimensions, followed by a 1D convolution along the third remaining dimension. While our problem is significantly different from action recognition classification, the network's ability to break down local features in the 3D data and bring it together might be helpful for us as well, if ample room for learning is given to it.

Since the model was designed for videos, it's first layer takes only 3 input channels (corresponding to RGB), so we modified the layer so that it takes 6 input channels. Furthermore, as is very common in transfer learning, the final fully-connected layer was replaced by a new layer whose number of outputs matched with our problem. While training, the whole R(2+1)D network was kept fixed (ie the parameters were not modified), except the first layer, the last fully-connected layer and the layer preceding it (to allow the network to adapt to the new regression problem). However the model was not able to learn anything, having the worst accuracy out of all the alternatives (Fig 5.8).

(III) Others

Apart from these 2 architecture alternatives we also tried:

- Network with only down-sampling with/without a fully-connected layer at the end.
- U-Net architecture with only one downsampling step.
- U-Net architecture with 3D convolution only in the first 2 layers before the first down-sampling, then 2D convolution for the rest of the network.

However with these modifications as well, the network was not able to perform better than one presented in Sections 5.1-5.3.

Chapter 6

Conclusion and Further Directions

In this work we have made progress towards developing a deep learning surrogate of a numerical atmospheric transport model for CO_2. This is a part of a bigger proposed algorithm for improving our estimates of emission distributions, described in Section 3.2. Modeling transport using deep learning promises to have some advantages over their numerical counterparts, however in Section 4 we came across a large number of constraints and obstacles that need to be overcome in this as well.

After a number of simplifications, our network is able to learn to model wind-driven advection to some degree of accuracy. It explains 75.1% of the variance in the output and the correlation between its predictions and the true output is 0.885. Furthermore, it is able to identify the most important input features and performs equally well on different meteorological conditions in the year. However there is still a long way to go before it can achieve an acceptable level of performance.

While several attempts were made and alternatives were tested in order to improve the performance of the model, none were able to do so. In future work, it is going to be imperative to identify and test more carefully designed alternatives. Some of the ideas in that direction are as follows:

- Feature Engineering: Create new input features based on the existing ones to make their distributions more similar without the need of explicit transformations. The right combination of features might be able to capture complex relationships between them and also allow us to reduce the number of features going as input to

the neural network. This will further allow us to include more and more physical processes in the transport modeling.

- We used U-Net that was designed for visual (medical) images. Perhaps, the features extracted by such networks is not very appropriate for a problem like transport modeling where instead of detecting edges and other visual features, the network needs to learn to combine different physical quantities in the right mathematical way. For this, a more tailor-made architecture might be needed.

- Despite our many simplifications, our model was still supposed to predict 4700 values at a time over a grid of $47 \times 10 \times 10$. Identifying the right architecture might become easier if we further break down the transport problem into smaller parts, eg using less number of vertical layers and predicting concentrations over fewer cells (or even a single cell) at a time. Once the right model is identified for the simplest task, it can be scaled up relatively easily.

Bibliography

[1] "Kyoto protocol to the united nations framework convention on climate change," 1997.

[2] "Conference of the parties, adoption of the paris agreement," 2015.

[3] D. McMarrow, "Methods for remote determination of co2 emissions," p. 206, 01 2011.

[4] W. Peters, J. Miller, J. Whitaker, S. Denning, A. Hirsch, M. Krol, D. Zupanski, L. Bruhwiler, and P. Tans, "An ensemble data assimilation system to estimate co2 surface fluxes from atmospheric trace gas observations," *Journal of Geophysical Research. D, Atmospheres 110 (2005) D24*, vol. 110, 12 2005.

[5] I. T. van der Laan-Luijkx, I. R. van der Velde, E. van der Veen, A. Tsuruta, K. Stanislawska, A. Babenhauserheide, H. F. Zhang, Y. Liu, W. He, H. Chen, K. A. Masarie, M. C. Krol, and W. Peters, "The carbontracker data assimilation shell (ctdas) v1.0: implementation and global carbon balance 2001–2015," *Geoscientific Model Development*, vol. 10, no. 7, pp. 2785–2800, 2017. [Online]. Available: https://gmd.copernicus.org/articles/10/2785/2017/

[6] L. Feng, P. I. Palmer, H. Bösch, and S. L. Dance, "Estimating surface co 2 fluxes from space-borne co 2 dry air mole fraction observations using an ensemble kalman filter," *Atmospheric Chemistry and Physics*, vol. 9, pp. 2619–2633, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:54776303

[7] F. Chevallier, M. Fisher, P. Peylin, S. Serrar, P. Bousquet, F.-M. Breon, A. Chedin, and P. Ciais, "Inferring co2 sources and sinks from satellite observations: Method and application to tovs data," *Journal of Geophysical Research-Atmospheres*, vol. 110, 12 2005.

[8] P. Bergamaschi, S. Houweling, A. Segers, M. Krol, C. Frankenberg, R. Scheepmaker, E. Dlugokencky, S. Wofsy, E. Kort, C. Sweeney, T. Schuck, C. Brenninkmeijer, H. Chen, V. Beck, and C. Gerbig, "Atmospheric ch4 in the first

decade of the 21st century: Inverse modeling analysis using sciamachy satellite retrievals and noaa surface measurements," *Journal of Geophysical Research (Atmospheres)*, vol. 118, pp. 7350–7369, 07 2013.

[9] Y. Niwa, H. Tomita, M. Satoh, R. Imasu, Y. Sawa, K. Tsuboi, H. Matsueda, T. Machida, M. Sasakawa, B. Belan, and N. Saigusa, "A 4d-var inversion system based on the icosahedral grid model (nicam-tm 4d-var v1.0) – part 1: Offline forward and adjoint transport models," *Geoscientific Model Development*, vol. 10, pp. 1157–1174, 03 2017.

[10] P. Houtekamer and H. Mitchell, "Data assimilation using an ensemble kalman filter technique," *Monthly Weather Review - MON WEATHER REV*, vol. 126, 03 1998.

[11] F. Rabier and Z. Liu, "Variational data assimilation : Theory and overview," 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:160032700

[12] P. Lauret, F. Heymes, L. Aprin, and A. Johannet, "Atmospheric dispersion modeling using artificial neural network based cellular automata," *Environmental Modelling Software*, vol. 85, pp. 56–69, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364815216304583

[13] Q. Tao, F. Liu, Y. Li, and D. Sidorov, "Air pollution forecasting using a deep learning model based on 1d convnets and bidirectional gru," *IEEE Access*, vol. 7, pp. 76 690–76 698, 01 2019.

[14] B. Wang and F. Qian, "Three dimensional gas dispersion modeling using cellular automata and artificial neural network in urban environment," *Process Safety and Environmental Protection*, vol. 120, pp. 286–301, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957582018308395

[15] I. Bey, D. Jacob, R. Yantosca, J. Logan, B. Field, A. Fiore, Q.-B. Li, H. Liu, L. Mickley, and M. Schultz, "Global modeling of tropospheric chemistry with assimilated meteorology: Model description and evaluation," *Journal of Geophysical Research: Atmospheres*, vol. 106, 11 2001.

[16] J.-F. Lamarque, L. Emmons, P. Hess, D. Kinnison, S. Tilmes, F. Vitt, C. Heald, E. Holland, P. Lauritzen, J. Neu, J. Orlando, P. Rasch, and G. Tyndall, "Cam-chem: Description and evaluation of interactive atmospheric chemistry in the community earth system model," *Geoscientific Model Development*, vol. 5, pp. 369–411, 03 2012.

[17] G. Grell, S. Peckham, R. Schmitz, S. McKeen, G. Frost, W. Skamarock, and B. Eder, "Fully coupled "online" chemistry in the wrf model," *Atmospheric Environment*, vol. 39, pp. 6957–6975, 12 2005.

[18] F. Hourdin, I. Musat, S. Bony, P. Braconnot, F. Codron, J.-L. Dufresne, L. Fairhead, M.-A. Filiberti, P. Friedlingstein, J.-Y. Grandpeix, G. Krinner,

P. LeVan, Z.-X. Li, and F. Lott, "The lmdz4 general circulation model: Climate performance and sensitivity to parametrized physics with emphasis on tropical convection," *Climate Dynamics*, vol. 27, pp. 787–813, 12 2006.

[19] D. Byun and K. Schere, "Review of the governing equations, computational algorithms, and other components of the models-3 community multiscale air quality (cmaq) modeling system," *Applied Mechanics Reviews*, vol. 59, pp. 51–77, 03 2006.

[20] I. Pisso, E. Sollum, H. Grythe, N. I. Kristiansen, M. Cassiani, S. Eckhardt, D. Arnold, D. Morton, R. L. Thompson, C. D. Groot Zwaaftink, N. Evangeliou, H. Sodemann, L. Haimberger, S. Henne, D. Brunner, J. F. Burkhart, A. Fouilloux, J. Brioude, A. Philipp, P. Seibert, and A. Stohl, "The lagrangian particle dispersion model flexpart version 10.4," *Geoscientific Model Development*, vol. 12, no. 12, pp. 4955–4997, 2019. [Online]. Available: https://gmd.copernicus.org/articles/12/4955/2019/

[21] S. Maksyutov, P. Patra, R. Onishi, T. Saeki, and T. Nakazawa, "Nies/frcgc global atmospheric tracer transport model: Description, validation, and surface sources and sinks inversion," *Journal of the Earth Simulator*, vol. 9, pp. 3–18, 03 2008.

[22] J. L. McGregor and M. R. Dix, *An Updated Description of the Conformal-Cubic Atmospheric Model*. New York, NY: Springer New York, 2008, pp. 51–75. [Online]. Available: https://doi.org/10.1007/978-0-387-49791-4_4

[23] Y. Niwa, H. Tomita, M. Satoh, R. Imasu, Y. Sawa, K. Tsuboi, H. Matsueda, T. Machida, M. Sasakawa, B. Belan, and N. Saigusa, "A 4d-var inversion system based on the icosahedral grid model (nicam-tm 4d-var v1.0) – part 1: Offline forward and adjoint transport models," *Geoscientific Model Development*, vol. 10, no. 3, pp. 1157–1174, 2017. [Online]. Available: https://gmd.copernicus.org/articles/10/1157/2017/

[24] G. P. Brasseur and D. J. Jacob, *Modeling of Atmospheric Chemistry*. Cambridge University Press, 2017.

[25] S.-J. Lin and R. Rood, "Multidimensional flux-form semi-lagrangian transport schemes," *Mon. Wea. Rev.*, vol. 124, pp. 2046–2070, 09 1996.

[26] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[27] T. Oda and S. Maksyutov, "A very high-resolution (1 km×1 km) global fossil fuel co_2 emission inventory derived using a point source database and satellite observations of nighttime lights," *Atmospheric Chemistry and Physics*, vol. 11,

no. 2, pp. 543–556, 2011. [Online]. Available: https://acp.copernicus.org/articles/11/543/2011/

[28] R. J. Andres, J. S. Gregg, L. Losey, G. Marland, and T. A. Boden, "Monthly, global emissions of carbon dioxide from fossil fuel consumption," *Tellus B: Chemical and Physical Meteorology*, Jan 2011.

[29] T. Takahashi, S. C. Sutherland, R. Wanninkhof, C. Sweeney, R. A. Feely, D. W. Chipman, B. Hales, G. Friederich, F. Chavez, C. Sabine, A. Watson, D. C. Bakker, U. Schuster, N. Metzl, H. Yoshikawa-Inoue, M. Ishii, T. Midorikawa, Y. Nojiri, A. Körtzinger, T. Steinhoff, M. Hoppema, J. Olafsson, T. S. Arnarson, B. Tilbrook, T. Johannessen, A. Olsen, R. Bellerby, C. Wong, B. Delille, N. Bates, and H. J. de Baar, "Climatological mean and decadal change in surface ocean pco2, and net sea–air co2 flux over the global oceans," *Deep Sea Research Part II: Topical Studies in Oceanography*, vol. 56, no. 8, pp. 554–577, 2009, surface Ocean CO2 Variability and Vulnerabilities. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967064508004311

[30] S. C. Olsen and J. T. Randerson, "Differences between surface and column atmospheric co2 and implications for carbon cycle research," *Journal of Geophysical Research: Atmospheres*, vol. 109, no. D2, 2004. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2003JD003968

[31] B. B. B. T. e. a. Granier, C., "Evolution of anthropogenic and biomass burning emissions of air pollutants at global and regional scales during the 1980–2010 period," *Climatic Change*, 2011.

[32] J. Messerschmidt, N. Parazoo, D. Wunch, N. Deutscher, C. Roehl, T. Warneke, and P. Wennberg, "Evaluation of atmosphere-biosphere exchange estimations with tccon measurements," *Atmospheric Chemistry and Physics*, vol. 13, pp. 5103–5115, 05 2013.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[34] D. Karimi, S. K. Warfield, and A. Gholipour, "Transfer learning in medical image segmentation: New insights from analysis of the dynamics of model parameters and learned representations," *Artificial Intelligence in Medicine*, vol. 116, p. 102078, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0933365721000713

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 06 2016, pp. 770–778.

[36] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *2018 IEEE/CVF Conference*

on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society, jun 2018, pp. 6450–6459. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00675