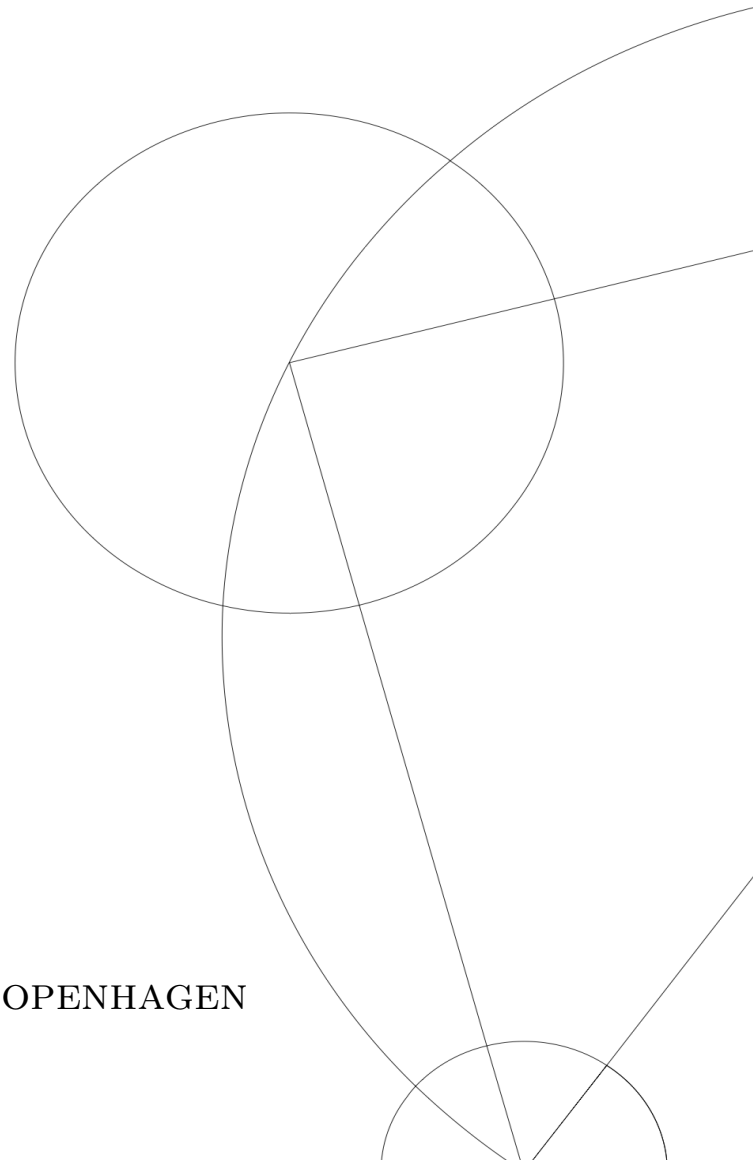
Development and implementation of a neural network based PBL turbulence parameterization scheme

Master Thesis

Written by *Kasper Tølløse*

02.03.2020

Supervised by
Eigil Kaas



University of Copenhagen

UNIVERSITY OF COPENHAGEN

Name of Institute:	University of Copenhagen
Name of Department:	Niels Bohr Institute
Author(s):	Kasper Tølløse
Email:	qrm173@alumni.ku.dk
Title and subtitle:	Development and implementation of a neural network based PBL turbulence parameterization scheme
Supervisor(s):	Eigil Kaas
Handed in:	02.03.2020

Abstract

In the planetary boundary layer (PBL), turbulent dynamics on subgrid-scale are responsible for the vertical transport of momentum, heat and moisture. Higher-order accuracy parameterization schemes for subgrid-scale dynamics are computationally heavy and account for a substantial part of the computation time in numerical weather prediction models. This study proposes and tests a new type of turbulence closure model using artificial neural networks (ANN) to establish the relation between the turbulent diffusivities and the known prognostic variables. The dataset for training the neural networks, was generated using data from forecasts with the Weather Research and Forecast model (WRF) with the relatively expensive Mellor-Yamanda-Nakanishi-Niino level 2.5 PBL scheme (MYNN). After several theoretical and technical considerations and tests, the final design of the ANN was established.

To evaluate the performance of the ANN based PBL scheme, the ANN was adapted to and implemented into the Fortran code of WRF. The MYNN scheme was then compared to both the ANN scheme and two other PBL scheme options in WRF, the YSU and MYJ schemes. This comparison showed that, out of these three PBL schemes, the ANN scheme is the one that best resembles the MYNN scheme: throughout the two 72-hour simulations selected for test cases, the ANN scheme's predictions of turbulent fluxes, temperature and wind near the surface have both the lowest root mean square error and highest pattern correlation with the MYNN scheme.

Further, the part of the ANN scheme that predicts diffusivities is a factor of 10 faster than the part of the MYNN scheme that was substituted. This thus suggests that a neural network based PBL scheme has the potential of being a very efficient alternative to computationally expensive second order PBL schemes such as MYNN.

The study indicates that neural networks are suitable for turbulence parameterization and demonstrates a method, which can easily be generalized and potentially be used to develop a new more accurate turbulence parameterization model by training on higher quality data from, e.g., large-eddy simulation.

Acknowledgements

First, I would like to thank my supervisor Prof. Eigil Kaas for providing valuable supervision throughout the process. Also thanks to rest of the Atmospheric Science group at the University of Copenhagen for beneficial discussions at our weekly meetings. Additionally, I would like to thank my fellow students with whom I shared the master students' office, whose company made this year a little easier, and my fellow student Frederik Faye for engaging in many enlightening discussions and for his useful comments on my writing. A final remark of appreciation goes to my family and friends, who have shown support throughout the entire process.

Contents

Introduction

One challenge in atmospheric modeling is dealing with turbulent micro-scale dynamics. The characteristic spatial scales of atmospheric dynamics range from global structures to eddies down to the order of 10^{-3}m. The governing equations for the atmosphere are the Navier-Stokes equation, the continuity equation, the thermodynamic equation and the equation of state [1, Ch. 8]. Since there are no known analytical solutions to the equation system, the only option is numerical solution. However, due to the enormous range in spatial scales, direct numerical simulation is not possible. In addition, due to the turbulent nature of the micro-scale dynamics, the atmosphere is essentially a chaotic system, meaning that it is only theoretically predictable for time scales characteristic for the dynamics considered. For the micro-scale meteorology, these time scales are typically much shorter than the time scales interesting for weather forecasting. Therefore, pursuing an exact prediction of the micro-scale dynamics in meteorology is not only difficult but also to some extend pointless, because the numerical solution will quickly diverge from the truth, regardless of the resolution. Instead, the micro-scale dynamics are filtered out by *Reynolds averaging* the equations. This way, only atmospheric dynamics on scales larger than a certain spatial scale is handled by numerical solution, while the effects of dynamics on smaller scales are parameterized.

In most of the atmosphere, the flow is essentially laminar and dynamics on scales smaller than the resolution can be ignored without consequences. However, a laminar flow in contact with a solid surface will have a region, in which the fluid *transitions* from being stationary at the surface to moving with the laminar flow some distance away from the surface. In this transition region, called the *boundary layer*, the flow is likely to become turbulent. Likewise, most of the micro-scale dynamics in the atmosphere occurs in the planetary boundary layer, *PBL*. Hence, parameterization models for subgrid-scale dynamics are traditionally called PBL parameterizations, even though the same schemes are often used throughout the atmosphere so that it is able to also handle the case of *free air turbulence*.

The Reynolds averaging introduces a set of new variables such that the equation system has more variables than equations. There are different approaches to close the equation system. Common for all approaches, however, is that data either from observations/experiments or from

high resolution simulations is used to estimate closure constants. One example is the Mellor-Yamanda-Nakanishi-Niino scheme, *MYNN* [10, 11, 12, 13], which has several closure constants, which are estimated by fitting to *large-eddy simulation* data. Large-eddy simulation, or simply *LES*, is a numerical method, which uses a resolution fine enough to resolve the *energy-containing eddies* but typically much coarser than the length scale of the *dissipative eddies* [1, Ch. 6]. The concepts of energy-containing eddies and dissipative eddies are explained in detail in Chapter 1. Hence, it seems like a natural next step to use a more complex machine learning based regression model, where some of the (maybe inaccurate) assumptions applied in traditional turbulence parameterizations can be avoided.

During recent years, machine learning algorithms have been tested and implemented in a wide range of problems. Especially artificial neural networks, *ANN*, or simply neural networks, have proven to be powerful in both classification and regression problems. To mention a few applications, neural networks have been successful in complex classification problems such as speech recognition and computer vision including object and face recognition in images [32]. Also in turbulence modeling, machine learning algorithms have been suggested as alternatives to traditional approaches. Ling et al. [36] successfully used deep neural networks to predict the stress tensor elements for different types of turbulent pipe flows. This is a good example of usage of neural networks for a regression problem, where no prior assumptions about the functional relation is needed. In weather and climate modeling, machine learning has also been suggested as an approach to parameterize subgrid-scale processes. Li et al. [37] did use neural networks to predict the boundary layer height for stable boundary layers, but to our knowledge, machine learning methods have not yet been used for turbulence parameterization models in weather and climate modeling. Several other physical parameterization models have been developed, such as models for parameterizing subgrid-scale clouds and deep convection [38, 39, 40].

This thesis proposes a neural network based PBL parameterization trained on synthetic data from simulations performed with the Weather Research and Forecast model, *WRF* using the MYNN PBL scheme. We focus on regional forecasts for Scandinavia, and therefore the training data is based on six simulations using a small domain covering most of Scandinavia and the British Islands. The main purpose is to show that neural networks are suitable for this type of problem. The idea is that if a neural network can emulate the behavior of the MYNN scheme, it would most likely be possible to train a neural network on data from high resolution simulations, which could then potentially improve the quality of turbulence modeling in the future. Further, the study aims to examine whether a neural network based PBL scheme can be an efficient alternative to some of the computationally heavy PBL parameterizations such as the MYNN scheme. To examine neural network based PBL scheme's behavior in a numerical weather

prediction model, the scheme has been implemented in the WRF model and tested against both the MYNN scheme and two other PBL schemes available in WRF, the YSU scheme and the MYJ scheme [15, 16].

The structure of the thesis is as follows: Chapter 1 describes the theoretical background for atmospheric turbulence modeling. Chapter 2 describes the WRF model and the configurations used for the different parts of the study. In Chapter 3, the theoretical background for neural networks is described, and Chapter 4 describes the process of developing the neural network based PBL scheme. Chapter 5 presents results of WRF simulations using the neural network based PBL scheme and compares these with WRF simulations using existing PBL parameterizations. Finally, a discussion of main key points as well as an overall conclusion is presented in Chapter 6.

Chapter 1

Atmospheric turbulence

This Chapter will give a thorough description of the planetary boundary layer dynamics. The derivations somewhat follow J. C. Wyngaard's textbook on atmospheric turbulence [1, Ch. 8]. In addition, before starting this work, the present author made a *project outside course scope*[1] on atmospheric turbulence. This study largely covered the theoretical background for the thesis, and therefore, the description presented in this Chapter is somewhat modified from that preliminary study [5]. This means that, in general, the text is similar to [5] with several identical paragraphs. However, the sections 1.1.2 and 1.3.5 are new.

In Section 1.1, the Reynolds averaged equations for the PBL are derived. In Section 1.2, the planetary boundary layer will be described, and in Section 1.3, we describe different approaches to parameterizing the effects of turbulence in the PBL.

1.1 Governing equations for the PBL

To be able to properly describe the PBL, we first need to understand the underlying dynamics. To obtain such understanding, Section 1.1.1 first briefly introduces the equations describing the dynamics of the atmosphere. Next, Reynolds averaging is used to make the equations numerically solvable. In Section 1.1.3, an interpretation of the *Reynolds terms* is provided. In Section 1.1.4, the equation for turbulent kinetic energy, or *TKE*, will be derived. In addition to play an important role in PBL parameterization, the TKE equation provides a way of understanding the physical processes responsible for the turbulence in the PBL.

Similiar to [1], different mathematical notation will be used, depending on the purpose. In most sections, index notation will be used because it simplifies the manipulations of vector equations. However, when appropriate, the equations will be written in component form. In index notation, if two vectors or matrices with a common index are multiplied together, it implies

[1] A 7.5 ECTS point project at the University of Copenhagen, supervised by Eigil Kaas.

a summation over that index. Thus, a dot product of two vectors $\vec{v}$ and $\vec{w}$ can be written as $\vec{v} \cdot \vec{w} = \sum_i v_i w_i = v_i w_i$. Similarly, the matrix product of a two matrices $\mathbf{A}$ and $\mathbf{B}$, $\mathbf{C} = \mathbf{AB}$ can be written as $C_{ij} = A_{ik} B_{kj}$ [2, App. B].

1.1.1 The fluid dynamical equations

Following Wyngaard [1, Ch. 8], we start by deriving the equations for the dry atmospheric boundary layer. At the end of the section, it is briefly described how the equations can be extended to include effects of water vapor and liquid water. We will use a *Lagrangian* description, where the changes of the physical variables are "seen" from the perspective of a fluid parcel following the flow. For this purpose, we introduce the total derivative

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \tilde{u}_i \frac{\partial}{\partial x_i},$$

where $\tilde{u}_i$ is the wind vector describing the velocity of the fluid parcel. Each atmospheric variable will be separated into a *base state* and the flow around it, where the base state is isentropic, static, and in hydrostatic balance. The hydrostatic assumption simply states that the pressure gradient force balances gravity

$$-\frac{1}{\rho_0} \frac{\partial p_0}{\partial x_i} + g_i = 0, \tag{1.1}$$

where the nought denotes the base state variables. For any atmospheric variable, say ρ, we will denote the *full* variable with a tilde and write $\tilde{\rho} = \rho_0 + \tilde{\rho}'$, where the prime then denotes the variation from the base state. $(x_1, x_2) = (x, y)$ are the horizontal coordinates, and $x_3 = z$ the vertical, which implies $g_i = -g\delta_{3i}$[2]. Further, in the derivations we need the equation of state and the definition of potential temperature

$$\tilde{p} = \tilde{\rho} R_d \tilde{T}, \tag{1.2}$$

$$\tilde{\theta} = \tilde{T} \left(\frac{\tilde{p}(0)}{\tilde{p}(z)} \right)^{R_d/c_p}, \tag{1.3}$$

where R_d is the gas constant for dry air, and c_p is the heat capacity for air at constant pressure. We first consider mass conservation expressed by the continuity equation

$$\frac{D\tilde{\rho}}{Dt} = -\tilde{\rho} \frac{\partial \tilde{u}_i}{\partial x_i},$$

As discussed by Wyngaard, in most applications to the PBL, we can use the *Boussinesq approx-*

[2]Note that g_i is the sum of the gravitational force and the fictive centrifugal force experienced by an air parcel seen from the perspective of the rotating coordinate system.

imation, where the continuity equation simplifies to requiring the wind field to be divergence free

$$\frac{\partial \tilde{u}_i}{\partial x_i} = 0, \tag{1.4}$$

Next, we consider momentum conservation expressed by the Navier-Stokes equation

$$\frac{D\tilde{u}_i}{Dt} = -\frac{1}{\tilde{\rho}}\frac{\partial \tilde{p}}{\partial x_i} - g\delta_{3i} - 2\epsilon_{ijk}\Omega_j \tilde{u}_k + \nu\nabla^2\tilde{u}_i,$$

where the terms on the right-hand side are, in order, pressure gradient force, gravity, Coriolis force, and viscous forces. The tensor ϵ_{ijk}, called the *Levi-Civita symbol*, is used to express the cross product in index notation and has a value of either $+1$, -1 or 0 depending on the permutation of i, j and k [2, App. B]. Assuming that density variations are small compared to the base state value, $\tilde{\rho}'/\rho_0 \ll 1$, we can write the pressure gradient force as

$$-\frac{1}{\tilde{\rho}}\frac{\partial \tilde{p}}{\partial x_i} = -\left(\frac{1}{1+\tilde{\rho}'/\rho_0}\right)\frac{1}{\rho_0}\frac{\partial \tilde{p}}{\partial x_i} \approx g\delta_{3i}\left(1 - \frac{\tilde{\rho}'}{\rho_0}\right) - \frac{1}{\rho_0}\frac{\partial \tilde{p}'}{\partial x_i}, \tag{1.5}$$

where we Taylor expanded the factor $(1+\tilde{\rho}'/\rho_0)^{-1}$ to first order, used Equation (1.1) and ignored second order terms. Expanding $\tilde{\rho}'$ to first order around the base state using (1.2), we get

$$\tilde{\rho}' \approx \left.\frac{\partial \tilde{\rho}}{\partial \tilde{T}}\right|_0 \tilde{T}' + \left.\frac{\partial \tilde{\rho}}{\partial \tilde{p}}\right|_0 \tilde{p}' = -\frac{\rho_0}{T_0}\tilde{T}' + \frac{1}{R_d T_0}\tilde{p}' \approx -\rho_0\frac{\tilde{T}'}{T_0} = -\rho_0\frac{\tilde{\theta}'}{\theta_0}, \tag{1.6}$$

where, as discussed by Wyngaard [1, Ch. 8], the second term involving $\tilde{p}'$ can be neglected[3]. The last rewriting simply uses (1.3). Finally, we can rewrite Navier-Stokes equation using Equation (1.5) and (1.6)[4]

$$\frac{D\tilde{u}_i'}{Dt} = -\frac{1}{\rho_0}\frac{\partial \tilde{p}'}{\partial x_i} + g\delta_{3i}\frac{\tilde{\theta}'}{\theta_0} - 2\epsilon_{ijk}\Omega_j\tilde{u}_k' + \nu\nabla^2\tilde{u}_i', \tag{1.7}$$

where the second term on the right-hand side is called the buoyancy term. The effect of buoyancy is that a fluid parcel with a temperature deviating from its surroundings, $\tilde{\theta}' \neq 0$, will feel an upwards or downwards force.

To obtain an equation for potential temperature, we look at the first law of thermodynamics,

[3] Wyngaard shows that pressure anomalies are approximately of the order $Mach^2$, so these should be negligible for velocities $u \ll c$ (speed of sound).

[4] Notice that primes have been added to the velocities here. Since the base state velocity is zero, this changes nothing and is merely done for consistency.

expressed in terms of specific entropy [5]

$$\tilde{T}\frac{D\tilde{s}}{Dt} = c_p\frac{D\tilde{T}}{Dt} - \frac{1}{\tilde{\rho}}\frac{D\tilde{p}}{Dt} \quad \Leftrightarrow \quad \frac{D\tilde{s}}{Dt} = \frac{c_p}{\tilde{\theta}}\frac{D\tilde{\theta}}{Dt},$$

where the second form is obtained by using Equation (1.2) and (1.3). The change in specific entropy can also be written as

$$\frac{D\tilde{s}}{Dt} = \frac{\tilde{Q}}{\tilde{T}},$$

where $\tilde{Q}$ is the rate of total heat transfer to the air parcel. Neglecting viscous dissipation, this can be written as minus the divergence of the heat fluxes due to conduction and radiation

$$\tilde{Q} = -\frac{1}{\tilde{\rho}}\frac{\partial}{\partial x_i}\left(-k\frac{\partial\tilde{T}}{\partial x_i} + \tilde{R}_i\right),$$

where k is the thermal conductivity and $\tilde{R}_i$ is the radiation. Together with the first law of thermodynamics, this finally leads to an equation for $\tilde{\theta}$

$$\frac{D\tilde{\theta}}{Dt} = \frac{\tilde{\theta}}{\tilde{T}}\alpha\nabla^2\tilde{T} - \frac{\tilde{\theta}}{\tilde{\rho}c_p\tilde{T}}\frac{\partial\tilde{R}_i}{\partial x_i} \approx \alpha\nabla^2\tilde{\theta} - \frac{\tilde{\theta}}{\tilde{\rho}c_p\tilde{T}}\frac{\partial\tilde{R}_i}{\partial x_i},$$

where $\alpha = k/(\tilde{\rho}c_p)$ is the thermal diffusivity. To get the second equality, we assume that variations in $\tilde{\theta}/\tilde{T}$ are negligible on the spatial scales, where thermal conductivity are important. Due to this equation's linearity in the variable $\tilde{\theta}$, we can similarly write the governing equation for the deviation from the base state as

$$\frac{D\tilde{\theta}'}{Dt} = \alpha\nabla^2\tilde{\theta}' - \frac{\tilde{\theta}'}{\tilde{\rho}c_p\bar{\tilde{T}}}\frac{\partial\tilde{R}_i}{\partial x_i}, \tag{1.8}$$

The set of equations (1.2), (1.3), (1.4), (1.7) and (1.8) describes the atmosphere under the Boussinesq approximation and are in principal numerically solvable. However, to numerically solve these equations, we need an extremely fine grid to resolve eddies on all scales. In Section 1.3.2, we find that the smallest eddies are of the order of Kolmogorov micro-scale, which for the atmosphere is typically $\eta \sim 10^{-3}$m [1, Ch. 1]. This makes direct numerical simulation impossible in practice. Therefore, we average the equations to get rid of all the small-scale turbulent motion. We rewrite all relevant variables as the sum of the slowly varying mean flow

[5] Starting from $dI = Q + W$, where I is the internal energy, Q is the diabatic heating, and $W = -pdV$ is the work done on the fluid parcel by its surroundings. We then use that $dS = Q/T$ and the definition of enthalpy $H = I + pV \Rightarrow dI = dH - d(pV)$ to write $TdS = dI + pdV \Rightarrow TdS = dH - Vdp$. Dividing through by the mass of the fluid parcel and using that the specific enthalpy is $h = c_pT$, we get $Tds = c_pdT - dp/\rho$.

and the turbulent fluctuations

$$\tilde{p}' = P + p, \qquad \tilde{u_i}' = U_i + u_i, \qquad \tilde{\theta}' = \Theta + \theta, \qquad \frac{\tilde{\theta}'}{\tilde{\rho} c_p \tilde{T}} \frac{\partial \tilde{R}_i}{\partial x_i} = R + r.$$

When averaging, we assume that the average of a *mean flow variable* is the variable itself, and the average of a fluctuating variable is zero. In addition, the average of the product of a mean variable and a fluctuating variable is also zero. These assumptions lead to following averaging rules for the mean of a variable and the mean of a product of two variables, say $A + a$ and $B + b$

$$\begin{aligned} \overline{A + a} &= A, \\ \overline{(A + a)(B + b)} &= AB + \overline{ab}. \end{aligned} \tag{1.9}$$

Using these averaging rules, we obtain the average of equations (1.4), (1.7) and (1.8)

$$\frac{\partial U_i}{\partial x_i} = 0, \tag{1.10}$$

$$\frac{D_{_U} U_i}{Dt} = -\frac{\partial}{\partial x_j} \overline{u_j u_i} - \frac{1}{\rho_0} \frac{\partial P}{\partial x_i} + g \delta_{3i} \frac{\Theta}{\theta_0} - 2 \epsilon_{ijk} \Omega_j U_k + \nu \nabla^2 U_i, \tag{1.11}$$

$$\frac{D_{_U} \Theta}{Dt} = -\frac{\partial}{\partial x_j} \overline{u_j \theta} - R + \alpha \nabla^2 \Theta, \tag{1.12}$$

where we defined total derivative of a fluid parcel following the mean flow

$$\frac{D_{_U}}{Dt} = \frac{\partial}{\partial t} + U_i \frac{\partial}{\partial x_i}.$$

We will use this from now on instead of the *real* total derivative, since only the mean wind is resolved in the model.

For completion, and for later use, we find the equations for the fluctuating parts simply by subtracting Equation (1.10)-(1.12) from Equation (1.4), (1.7) and (1.8). We obtain

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{1.13}$$

$$\frac{D_{_U} u_i}{Dt} = -\frac{\partial}{\partial x_j} (U_j u_i + u_j u_i - \overline{u_j u_i}) - \frac{1}{\rho_0} \frac{\partial p}{\partial x_i} + g \delta_{3i} \frac{\theta}{\theta_0} - 2 \epsilon_{ijk} \Omega_j u_k + \nu \nabla^2 u_i, \tag{1.14}$$

$$\frac{D_{_U} \theta}{Dt} = -\frac{\partial}{\partial x_j} (u_i \Theta + U_i \theta - \overline{u_j \theta}) + \alpha \nabla^2 \theta, \tag{1.15}$$

where, as discussed by Wyngaard [1, Ch. 8], the radiation fluctuations r can be neglected and thus do not occur in (1.15).

1.1.2 Including effects of water vapor and liquid water

When the air contains water vapor, it changes the gas constant R in the equation of state (1.2). The correct gas constant for moist air can be expressed as a combination of the gas constants for dry air, R_d and for water vapor R_v

$$R = \frac{M_d R_d + M_v R_v}{M_d + M_v} = R_d \left(1 - \tilde{q}_v + \frac{R_v}{R_d}\tilde{q}_v\right),$$

where M_d and M_v are the masses of the dry air and the water vapor, respectively. Further, we have introduced the *specific humidity* $\tilde{q}_v = \frac{M_v}{M_d + M_v}$. However, instead of changing gas constant, one can introduce a fictitious temperature, called the virtual temperature T_v, and redefine the equation of state

$$\tilde{p} = \tilde{\rho} R_d \tilde{T}_v, \tag{1.16}$$

$$\tilde{T}_v = \left(1 + \left(\frac{R_v}{R_d} - 1\right)\tilde{q}_v\right)\tilde{T} \approx (1 + 0.61\tilde{q}_v)\,\tilde{T}. \tag{1.17}$$

The virtual temperature can be interpreted as the temperature a dry air parcel would need to have the same density at the same pressure as the moist air parcel. By similarly defining the virtual potential temperature

$$\tilde{\theta}_v = (1 + 0.61\tilde{q}_v)\,\tilde{\theta}. \tag{1.18}$$

By repeating the derivations in this Section, it can be shown that the effects of water vapor consist of substituting $\tilde{\theta}$ with $\tilde{\theta}_v$ in the buoyancy term in Equation (1.11). In addition, a new prognostic equation for tracing q_v is needed.

The effects of liquid water can be included in a similar way, however, one needs to take into account the diabatic heating and as consequence of the condensation and evaporation. Also, the prognostic equations for water vapor and liquid water will have source/sink terms as consequence of condensation and evaporation. One approach, as suggested by Mellor and Yamanda [9], is to introduce variables, which are conserved even when phase changes occur. Thus, Mellor and Yamanda uses the total water content $\tilde{q}_w = \tilde{q}_v + \tilde{q}_l$, where the $\tilde{q}_v$ and $\tilde{q}_l$ are the specific humidity and specific liquid-water content, respectively. Similarly, instead of $\tilde{\theta}$, they use the *liquid water potential temperature* defined as

$$\tilde{\theta}_l = \tilde{\theta} - \frac{\tilde{\theta}}{\tilde{T}}\frac{L_v}{c_p}\tilde{Q}_l, \tag{1.19}$$

where L_v is the latent heat of vaporization. The liquid water potential temperature is conserved under condensation and evaporation and can be interpreted as the potential temperature an air

parcel would have, if all liquid water in the parcel evaporated. Thus, in addition to the equations for mass and momentum conservation, Mellor and Yamanda introduces the following two mean field equations

$$\frac{D_U \Theta_l}{Dt} = -\frac{\partial}{\partial x_j}\overline{u_j \theta_l} - \frac{\Theta}{T}\left(\frac{1}{\rho c_p}\frac{\partial F}{\partial z} + \frac{L_v}{c_p}\frac{\partial G}{\partial z}\right), \tag{1.20}$$

$$\frac{D_U Q_w}{Dt} = -\frac{\partial}{\partial x_j}\overline{u_j q_w} + \frac{\partial G}{\partial z}, \tag{1.21}$$

where F is the net radiative flux, and G is the gravitational settling flux (positive downward), i.e. the flux of the liquid water due to settling water droplets. Note that the molecular diffusion terms have been neglected, which can be justified for all the mean field equations, since the effects are generally very small compared to the remaining terms [1, Ch. 8].

1.1.3 Interpretation of Reynolds terms

The averaging procedure used above is called *Reynolds averaging*, and as we see in Equation (1.11) and (1.12), it introduces new so-called *Reynolds terms*. Mathematically, the Reynolds terms are simply the variances and covariances of all the turbulent fluctuating variables. To be able to interpret these physically, we will start by considering the kinematic flux[6] of some quantity λ that may or may not be a vector quantity. The kinematic flux of this quantity is simply, the product with the velocity vector, $u_i\lambda$. Taking, for example, λ as the potential temperature $\lambda = \theta$, we see that the kinematic flux is $u_i\theta$. Similarly, for $\lambda = u_j$, we get the kinematic flux of momentum per unit mass, $u_i u_j$.

From the above, we can see that terms of the form $\frac{\partial}{\partial x_i}\overline{u_i u_j}$ can be interpreted as the divergence of the average turbulent flux of momentum u_j. Similarly, $\frac{\partial}{\partial x_i}\overline{u_i\theta}$ is the divergence of the average turbulent flux of potential temperature.

The turbulent fluctuations are expected to be random, however, the variances and covariances are not random. This means that if we can determine the statistics of the average turbulent fluxes, we can estimate the effects of the turbulence without directly resolving it. The idea is that the turbulent statistics will depend on the mean field variables and can be parameterized based on these. Different approaches for this is described in Section 1.3.

1.1.4 Turbulent kinetic energy

The process of obtaining the governing equation for TKE includes several steps and is a bit cumbersome, and therefore some details are left out. The idea behind the derivation is, however, not conceptually complicated and will be outlined below.

[6] The dynamic flux divided by ρ.

First, we specify that by TKE we denote the average turbulent kinetic energy $1/2\overline{u_i u_i}$. To obtain a prognostic equation, we start by writing an expression for the unaveraged quantity, and then we can simply use the averaging rules of Equation (1.9). We use that

$$\frac{D_U}{Dt}\left(\frac{1}{2}u_i u_i\right) = u_i \frac{D_U}{Dt}u_i,$$

Inserting the expression for $\frac{D_U}{Dt}u_i$ from Equation (1.14) in the equation above, and using the zero-divergence requirement (1.13), then averaging and rewriting, we obtain the governing equation for TKE

$$\frac{D_U}{Dt}TKE = -\overline{u_i u_k}\frac{\partial U_i}{\partial x_k} + \frac{1}{\theta_0}g\delta_{3i}\overline{\theta u_i} - \frac{\partial \overline{u_i u_i u_k}}{\partial x_k} - \frac{1}{\rho_0}\frac{\partial \overline{p u_i}}{\partial x_i}, -\varepsilon,$$

where the viscous terms are represented by ε, which is the viscous dissipation rate. Note that the Foriolis term, before averaging, is $-u_i 2\epsilon_{ikm}\Omega_k u_m$, or in vector notation $-2\mathbf{u}\cdot\mathbf{\Omega}\times\mathbf{u}$. Thus, the term vanishes, since the vector $\mathbf{\Omega}\times\mathbf{u}$ is perpendicular to $\mathbf{u}$. It is common to assume for the PBL that the fields are horizontally homogeneous and mean vertical velocity is zero. Hence, the TKE equation becomes

$$\frac{D_U}{Dt}TKE = -\overline{uw}\frac{\partial U}{\partial z} - \overline{vw}\frac{\partial V}{\partial z} + \frac{g}{\theta_0}\overline{w\theta} - \frac{\partial \overline{u_i u_i w}}{\partial z} - \frac{1}{\rho_0}\frac{\partial \overline{pw}}{\partial z} - \varepsilon, \tag{1.22}$$

The first and second terms are shear production related to the vertical gradients of the wind components. The third term is buoyant production of TKE. The fourth term is the rate of change in TKE due to turbulent transport, i.e. transport of TKE by the turbulent fluctuating components of the wind. The fifth term is the rate of change in TKE due to pressure transport, and the sixth term is the viscous dissipation rate.

1.2 Turbulence in the planetary boundary layer

In fluid dynamics, a non-dimentionalized version of the Navier-Stokes equation is often used to characterize the flow of a fluid. In this non-dimentionalized version, we can characterize the flow solely by its Reynolds number, which combines information about the spatial and velocity scales with the viscosity into one single number. The Reynolds number is defined as

$$Re = \frac{UL}{\nu},$$

where U and L are the characteristic velocity and length scales of the flow, and ν is the kinematic viscosity. Mathematically, the Reynolds number is approximately the ratio of the inertia forces and viscous forces of the flow [2, Ch. 15]. While the inertia forces try to keep the flow moving,

the viscous forces are internal friction forces, which try to slow down the flow. Thus, for low Reynolds numbers, viscous forces are strong enough to make the flow sluggish and laminar, whereas for high Reynolds numbers, typically larger than $\sim$ 2000, the inertia forces will be strong enough to keep the flow lively [1, Ch. 1]. The viscosity will in this case result in shear forces that will cause the flow to become turbulent in certain regions. When a fluid with a low viscosity, such as atmospheric air moves in contact with a stationary, solid surface, there will always be a region, the *boundary layer*, where turbulence can occur.

One important characteristic of turbulence is that it is chaotic. This means that even infinitely small differences in initial conditions of two physical systems will, after some time, grow exponentially large and make the solutions diverge. This makes chaotic systems, and hence turbulent flows, impossible to predict for longer than the time scale, τ characteristic for the turbulence. Turbulence can be thought of as the fluid's tendency to create vortices, and turbulent flows are therefore dominated by rotation on all spatial scales. The larger eddies are also called the *energy-containing eddies*, since they contain most of the turbulent kinetic energy. However, the smaller eddies are important, because they drain kinetic energy from the larger eddies. This drainage of energy by smaller eddies continues at all spatial scales until turbulence cannot exist any longer, and the kinetic energy dissipates to heat due to viscous friction. This key property of turbulence is called *energy cascade*. It implies that, from eddies of any spatial scale, energy gets transferred to smaller scales at a rate that must be equal to the viscous dissipation rate ε. If this is not at least statistically true, kinetic energy would accumulate at certain spatial scales [1, Ch. 3]. In the PBL, the size of the largest eddies is typically of the order $l \sim 10^3 m$ with corresponding wind speeds $u \sim 1 m s^{-1}$. This gives a characteristic time scale, also called the *eddy turnover time*, of approximately $\tau \sim l/u \sim 10^3 s \sim 20$ minutes.

To gain better understanding of turbulence in the PBL, it is useful to further divide it into sublayers, which can then be analyzed individually. In the following sections, these layers will be described one by one.

1.2.1 Surface layer

Immediately above the surface is a *viscous sublayer*, where no turbulence can exist, because all velocities must vanish at the boundary. This sublayer is very thin, typically of the order of Kolmogorov microscale $\eta \sim 10^{-3}$m, since this is the length scale of the smallest possible eddies, see Section 1.3.2. There will in general be large gradients in this viscous sublayer, which causes molecular diffusion of momentum, temperature and moisture to be important here. However, above the viscous sublayer, where turbulent fluctuations are nonzero, *eddy diffusion* starts to dominate, and molecular diffusion can generally be ignored, see Section 1.3.1.

Near the surface, the turbulent shear stress points in the direction of the mean surface wind.

It is conventional in PBL applications to use a coordinate system, where the x-axis is aligned with the surface wind, such that $V = 0$, and the shear stress is $\tau_0 = -\rho_0 \overline{wu}$.

Further, we define the *friction velocity*, u_* as a measure of the strength of the kinematic surface stress (equivalent to the turbulent fluxes from 1.1.3) τ_0/ρ_0

$$u_* = \sqrt{\frac{\tau_0}{\rho_0}}. \tag{1.23}$$

It can be shown that the turbulent fluxes are, to a good approximation, independent of height in the lowest part of the PBL, [1, Ch. 10]. This region, where the turbulent fluxes are constant, is called the surface layer, or sometimes the *constant-flux layer*. The surface layer is typically the lowest $\sim 10\%$ of the PBL, [1, Ch. 11].

1.2.2 Intermediate layer

Above the surface layer is an intermediate layer, in which the turbulent fluxes are no longer independent of height. The structure of this intermediate layer strongly depends on the static stability of the PBL, [1, Ch. 11-12]. The stability of the PBL is determined, most importantly, by the surface flux of virtual potential temperature, but it also depends on entrainment of warmer air from the top of the PBL. We denote the surface flux of virtual potential temperature $\left(\overline{w\theta_v}\right)_0$. The sign of $\left(\overline{w\theta_v}\right)_0$ determines whether the atmosphere is heated or cooled by the surface, and therefore determines the slope of the Θ-profile.

From the TKE equation (1.22), we see that turbulent kinetic energy is balanced by production due to mean velocity shear, production/destruction due to buoyancy, transport associated with turbulent fluctuations, transport associated with the pressure gradient, and viscous dissipation. The sign of buoyancy term depends on the sign of θ_v, which in general depends on the temperature profile. Assume a stable atmosphere $\frac{\partial \Theta_v}{\partial z} > 0$, then if a fluid parcel is displaced upwards, it will be colder than its surroundings, $\theta_v < 0$, and the parcel will be subject to a downward force. Likewise, if it is displaced downwards, $\theta_v > 0$, it will be subject to an upwards force. We say the PBL is stable, because it dampens vertical motion. As a natural consequence of this damping, in the stable boundary layer, *SBL*, the buoyancy term in Equation (1.22) acts as a TKE drain. In an unstable atmosphere, where $\frac{\partial \Theta_v}{\partial z} < 0$, the opposite arguments are applicable. Hence, a fluid parcel that is vertically displaced, will be subject to a force in the same direction as the initial displacement. It follows that in an unstable/convective boundary layer, *CBL*, the buoyancy term in Equation (1.22) acts as a TKE production term.

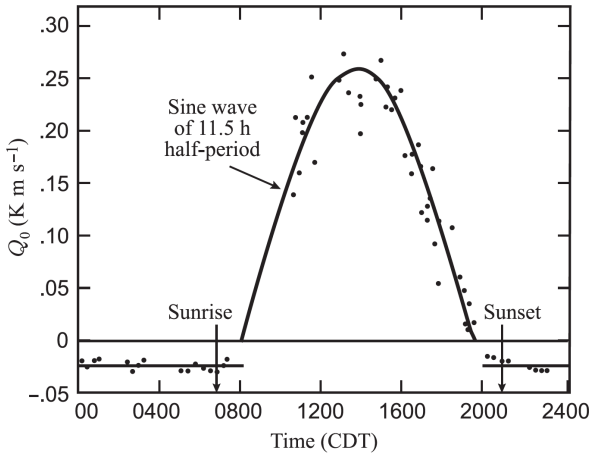


Figure 1.1: Shows observed surface temperature flux from the Kansas experiment. The values are 1-hour averages obtained from data collected over 3 weeks. Figure source: [1].

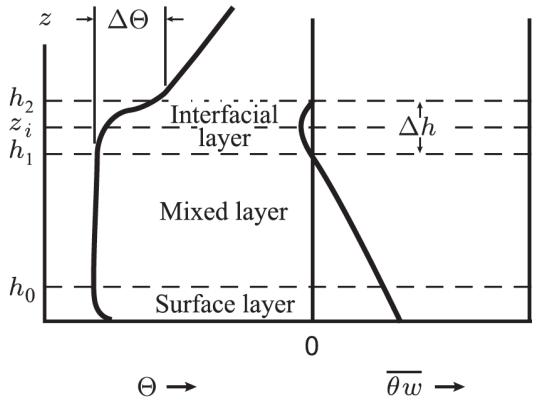


Figure 1.2: Shows typical vertical profiles for Θ and $\overline{w\theta}$. h_0 is the height of the surface layer, h_1 the top of the mixed layer, and h_2 the top of the interfacial layer. z_i is the *average* physical height of the mixed layer, and Δh is the thickness of the interfacial layer when averaging. Figure source: [1].

Due to the diurnal cycle in incoming short wave radiation from the sun, there is a corresponding diurnal cycle in the surface temperature flux. During the day, the atmosphere is typically heated from the surface, whereas it is cooled during night, see Figure 1.1. This means that during the day, the PBL is typically unstable and dominated by vertical movements, where buoyancy is the main TKE production term. Due to the vertical motion, there will be efficient mixing of the air, which leads to a nearly neutral temperature profile in this intermediate layer, see Figure 1.2. Therefore, it is generally referred to as the *mixed layer* in the CBL case. From the figure, we also see a typical profile for $\overline{\theta w}$, where we see that due to the warmer air above the PBL,

the sign of the flux changes in the *interfacial layer* described in the following section. The measurements shown in Figure 1.1 are from the 1968 Kansas experiment, where the surface temperature flux has been measured over a dry prairie. But as described in the beginning of this section, when moisture is present, the surface flux of virtual potential temperature should be considered instead [1, Ch. 9].

During night, when vertical motion is damped, there is less vertical mixing, and the slope of the temperature profile is positive throughout SBL. Normally, no individual name is provided for the intermediate layer in the stable case, however, it is still important to distinguish it from the surface layer, since the turbulent fluxes cannot necessarily be assumed constant. In general, the TKE balance is much more fragile in the SBL, since the shear production term may not be sufficient to sustain turbulence in very stable conditions [1, Ch. 11].

1.2.3 Interfacial layer

At last, there is a thin *interfacial layer*, which forms the boundary between the turbulent boundary layer and the free laminar atmosphere above. The instantaneous thickness of this interfacial layer is down to the scale of Kolmogorov microscale, η, but when averaged over time, space or ensembles, the thickness can be a substantial fraction of the height of the PBL [1, Ch. 9]. This is also evident from the sketch of a CBL temperature profile in Figure 1.2.

1.3 Turbulence parameterization in NWP

In Section 1.1, we saw that averaging the variables introduced *Reynolds terms*, which are essentially average turbulent fluxes of the mean field variables. While this averaging makes it possible to solve the equations on a much courser grid[7], it introduces several new variables. To close the equation system, we therefore need to introduce a new set of equations, so that the number of equations matches the number of variables. A simple way of doing this, described and motivated in Section 1.3.1, is by using the concept of eddy diffusion and determine an eddy diffusion coefficient, K for each turbulent flux at each point in space. Sections 1.3.2 and 1.3.3 describe the concept of similarity theory and how this was used by Monin and Obukhov to make a parameterization of the surface fluxes. Both of these methods are examples of so-called *first order* models. Another type of models, described in Section 1.3.4, use the prognostic equations for all or some of the turbulent fluxes. This approach is called second order closure, because second order terms are determined from the prognostic equations, while third order terms are parameterized. As an example of a second order closure model, the *Mellor-Yamanda* model [7, 8, 9] is described in Section 1.3.4. Finally, Section 1.3.5 describes the MYNN scheme,

[7] The characteristic spatial scales of the mean fields are much larger than of the turbulent fluctuations.

which is an improved version of the Mllor-Yamanda model, developed by Nakanishi and Niino [10, 11, 12, 13].

1.3.1 K-closure

K-closure is based on the concept of eddy diffusion, which is the idea that turbulent fluxes works like a macroscopic version of molecular diffusion. To motivate this idea, we will start by presenting the *mixing-length hypothesis* presented by Ludwig Prandtl [1, Ch. 4.5]. Prandtl hypothesized that the turbulent fluctuations could be linked to the gradient of the mean field variables and a *mixing length*, d. This mixing length is a measure of the characteristic distance a fluid parcel is displaced by the turbulent velocity field, before it interacts with its surroundings. This assumption seems quite reasonable, since the fluid parcel will carry its properties of heat, momentum and moisture until it mixes with the "new" surrounding air. Considering the vertical displacement Δz of a fluid parcel, the deviation of potential temperature of this fluid parcel from the surrounding air is

$$\theta = \Theta(z) - \Theta(z + \Delta z) \approx -\Delta z \frac{\partial \Theta}{\partial z}.$$

Since vertical variations in Θ are generally much larger than horizontal variations, we ignore the horizontal components of the gradient, and we let d denote the length scale related to the turbulent fluid displacements in the vertical. It then follows that

$$\theta \sim -d\frac{\partial \Theta}{\partial z} \quad \Rightarrow \quad \theta w \sim -dw\frac{\partial \Theta}{\partial z} \quad \Rightarrow \quad \overline{\theta w} \sim -K\frac{\partial \Theta}{\partial z},$$

where $K = \overline{dw}$, the eddy diffusivity, is similar to the molecular diffusion coefficient in the diffusion equation. K generally has a vertical profile depending on the local properties, since d potentially depends on these. We can write similar equations for fluxes of momentum and moisture, each with potentially different eddy diffusivities. In principal, if we can empirically determine the functions for these eddy diffusivities, we can then solve the averaged equations (1.10)-(1.12).

1.3.2 Similarity theory

Similarity theory is a method to determine a universal relationship between a number of non-dimensionalized variables, which are characteristic for a physical system. These dimensionless variables can be determined using the *Buckingham Pi theorem*, which states:

- If a physical quantity q_m, which is functionally related to another $m-1$ independent physical quantities q_i, for $i = 1, 2, ..., m-1$, and the system has n physical dimensions,

then we can form $k = m - n$ independent dimensionless variables π_i for $i = 1, 2, ..., k$.

- These k variables can be chosen arbitrarily and are functionally related. In other words, any physically meaningful function $f(q_1, q_2, ..., q_m) = 0$ has an equivalent dimensionless function $F(\pi_1, \pi_2, ..., \pi_k) = 0$, where $k = m - n$, and n is the dimensionality of the system.

Note that the theorem gives no information about the functional form of $F(\pi_i)$ so, generally, it must be determined empirically. Since the dimensionless variables can be chosen arbitrarily, there will be more than one way to do this. However, there will most likely be one choice that leads to a more obvious physical interpretation.

As we shall see in the following examples, this dimension analysis simplifies the problem by lowering the number of variables in the system. First, we will look at two simple examples that demonstrate how the theorem can be used, and in Section 1.3.3, we will look at the applications of this theorem by Monin and Obukhov.

Kolmogorov hypothesized that the length and velocity scales of the dissipative eddies in a turbulent fluid, denoted η and υ, depend only on the energy dissipation rate ε and the kinematic viscosity ν. Our physical equations are thus

$$\eta = \eta(\varepsilon, \nu) \quad \text{and} \quad \upsilon = \upsilon(\varepsilon, \nu).$$

In both cases, we have $m = 3$. The units of the variables are $[\varepsilon] = m^2 s^{-3}$, $[\nu] = m^2 s^{-1}$, $[\eta] = m$ and $[\upsilon] = m s^{-1}$, which implies that system is described by $n = 2$ dimensions, length and time. Hence, for both η and υ we have $k = m - n = 1$ independent variable. Since there are no other variables it can depend on, it must be a constant. For η, we'll choose the dimensionless variable to be the variable itself non-dimensionalized with ε and ν. The only way to combine these variables to give the dimension of length is $\eta_* = \left(\nu^3/\varepsilon\right)^{1/4}$, and therefore we have $\eta/\eta_* = constant$. Following a similar approach for υ, we get $\upsilon/\upsilon_* = constant$, with $\upsilon_* = (\nu\varepsilon)^{1/4}$. Taking these constants to be 1, we get Kolmogorov's famous scaling laws

$$\eta = \left(\frac{\nu^3}{\varepsilon}\right)^{1/4} \quad \text{and} \quad \upsilon = (\nu\varepsilon)^{1/4}.$$

Although, this is a simple case of dimensional analysis, it allows us to say something very general about the dissipative scale of turbulent flows, based solely on a physical intuition of which parameters should be of importance.

1.3.3 Monin-Obukhov similarity

Alexander Monin and Andrei S. Obukhov assumed that the *surface layer structures* for momentum and temperature depend only on the variables $u, l, \left(\overline{w\theta_v}\right)_0$ and g/θ_0. The velocity scale of

the turbulence is taken to be the friction velocity, $u = u_*$. The length scale of the turbulence, is taken to be the height above the surface, $l = z$. $\left(\overline{w\theta_v}\right)_0$ is again the surface flux of virtual potential temperature, and g/θ_0 is the buoyancy parameter. The *surface layer structure* denotes the mean vertical gradients as well as the turbulent fluxes. According to the assumptions by Monin and Obukhov, we can write $\frac{\partial U}{\partial z} = \frac{\partial U}{\partial z}(u_*, z, \left(\overline{w\theta_v}\right)_0, g/\theta_0)$ and $\frac{\partial \Theta}{\partial z} = \frac{\partial \Theta}{\partial z}(u_*, z, \left(\overline{w\theta_v}\right)_0, g/\theta_0)$. In each case we have $m = 5$ and $n = 3$, namely length, time and temperature. Using the Buckingham Pi theorem, we can form $k = m - n = 2$ independent dimensionless quantities. In each case, we will take the dependent variable and make it dimensionless with u_*, z and $\left(\overline{w\theta_v}\right)_0$. The second dimensionless quantity is taken to be z/L_M, where L_M is the *Monin-Obukhov length* defined as

$$L_M = -\frac{u_*^3 \theta_0}{\kappa g \left(\overline{w\theta_v}\right)_0}, \tag{1.24}$$

with the *von Kármán* constant, determined experimentally, $\kappa \approx 0.4$. As discussed in Section 1.2.2, the sign of $\left(\overline{w\theta_v}\right)_0$ is important for determining the stability of the PBL. If $\left(\overline{w\theta_v}\right)_0 > 0$, the air at the surface is heated, and the PBL is unstable, whereas if $\left(\overline{w\theta_v}\right)_0 < 0$, the PBL is stable because the air is cooled at the surface. Thus, it is apparent that the sign of the Monin-Obukhov length, L_M depends on the stability as well. If $L > 0$, the PBL is stable, and if $L_M < 0$, the PBL is unstable.

We know that the dimensionless quantities are functionally related, so we can write

$$\frac{\kappa z}{u_*}\frac{\partial U}{\partial z} = \phi_m\left(\frac{z}{L_M}\right) \quad \text{and} \quad -\frac{\kappa z u_*}{\left(\overline{w\theta_v}\right)_0}\frac{\partial \Theta}{\partial z} = \phi_h\left(\frac{z}{L_M}\right), \tag{1.25}$$

where ϕ_m and ϕ_h are unknown functions. The negative sign on the left-hand side of the definition of ϕ_h is chosen to make the function positive. Using data from the Kansas experiment, Hogstrom proposed following functional forms of ϕ_m and ϕ_h

$$\begin{aligned} \text{stable}: \quad & \phi_m = 1.0 + 4.8\frac{z}{L_M}, && \phi_h = 1.0 + 7.8\frac{z}{L_M}, \\ \text{unstable}: \quad & \phi_m = \left(1.0 - 19.3\frac{z}{L_M}\right)^{-1/4}, && \phi_h = \left(1.0 - 12.0\frac{z}{L_M}\right)^{-1/2}. \end{aligned} \tag{1.26}$$

In Figure 1.3 the functions, Equation (1.26) are plotted together with the experimental data. Since the fluxes are approximately constant in the surface layer, we can integrate (1.25) from the surface to the height z by inserting (1.26). In the stable case, we get

$$U(z) = \frac{u_*}{\kappa}\left[\ln\frac{z}{z_0} + 4.8\frac{z}{L_M}\right], \qquad \Theta(z) = \Theta(z_r) + \frac{\left(\overline{w\theta_v}\right)_0}{\kappa u_*}\left[\ln\frac{z}{z_r} + 7.8\frac{z}{L_M}\right], \tag{1.27}$$

where z_0 is the average height where the velocity vanishes, and is thus a measure of the roughness of the surface. Similarly, z_r is the lower limit of the integral of ϕ_h. We have from the definition of the friction velocity (1.23) that $u_* = \sqrt{-\overline{wu}}$, and therefore the surface fluxes can be determined directly from Equation (1.27). By evaluating (1.27) at some reference height z_{ref}, the surface fluxes can be predicted from the variables z_0, z_r, $U(z_{ref})$, $\Theta(z_r)$ and $\Theta(z_{ref})$.

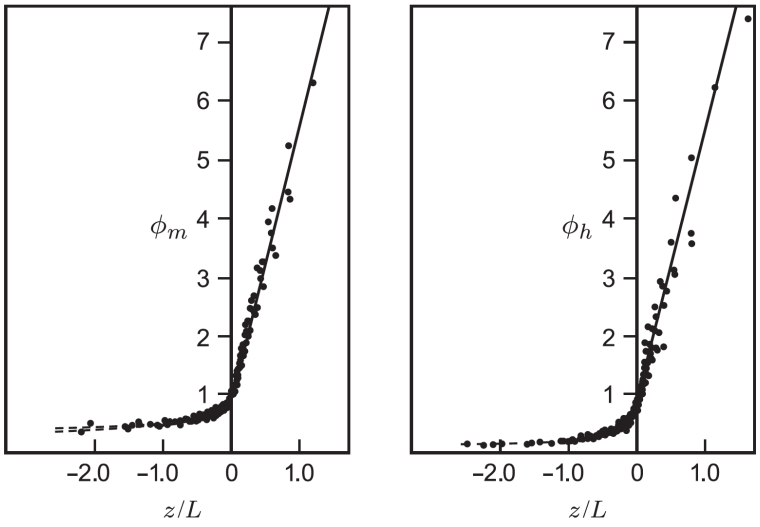


Figure 1.3: Shows plot of the functions ϕ_m (left) and ϕ_h (right), from Equation (1.26) together with data from the Kansas experiment. Figure source: [1].

1.3.4 Second order closure models

As discussed briefly in the introductory text of this section, models that use prognostic equations for all or some of the turbulent fluxes are called second order closure models. This term, however, does not imply, for *which* of the fluxes we include second order terms. To distinguish between different second order models, Mellor and Yamanda developed a new system for classifying turbulence parameterization models [8]. They denote a model that uses prognostic equations for all the turbulent fluxes in Equation (1.10)-(1.12) a *Level 4* model. The *Level 3, Level 2* and *Level 1* models then uses different degrees of approximations. Below, the turbulent flux budgets are shown, and at the end of this section, we briefly describe the Level 4 model and discuss the main implications of the different approximations leading to the lower level models.

Turbulent flux budgets

The process of getting the turbulent flux budgets is similar to determining the TKE equation in Section 1.1.4. Again, the derivations are quite cumbersome, so details are left out, but the

main steps are explained. To determine the budget of $\overline{u_i u_j}$, $\overline{\theta u_j}$ and $\overline{\theta^2}$, we use the relations

$$\frac{D_U}{Dt}u_i u_j = u_j\frac{D_U u_i}{Dt} + u_i\frac{D_U u_j}{Dt}, \qquad \frac{D_U}{Dt}\theta u_i = \theta\frac{D_U u_i}{Dt} + u_i\frac{D_U \theta}{Dt}, \qquad \frac{D_U}{Dt}\theta^2 = 2\theta\frac{D_U \theta}{Dt}.$$

We insert the expressions for $\frac{D_U}{Dt}u_i$ and $\frac{D_U}{Dt}\theta$ from Equation (1.14) and (1.15) in the equations above and use the zero-divergence requirement (1.13). By averaging, we obtain, after quite a bit of rewriting

$$\begin{aligned}
\frac{D_U}{Dt}\overline{u_i u_j} =& -\overline{u_j u_k}\frac{\partial U_i}{\partial x_k} - \overline{u_i u_k}\frac{\partial U_j}{\partial x_k} - 2(\epsilon_{ikm}\Omega_k\overline{u_m u_j} + \epsilon_{jkm}\Omega_k\overline{u_m u_i}) \\
&+ \frac{1}{\theta_0}(g\delta_{3i}\overline{\theta u_j} + g_j\overline{\theta u_i}) + \nu\nabla^2\overline{u_i u_j} \\
&- \frac{\partial\overline{u_i u_j u_k}}{\partial x_k} + \frac{1}{\rho_0}\overline{p\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)} - \frac{1}{\rho_0}\left(\frac{\partial\overline{p u_i}}{\partial x_j} + \frac{\partial\overline{p u_j}}{\partial x_i}\right) - 2\nu\overline{\frac{\partial u_i}{\partial x_k}\frac{\partial u_j}{\partial x_k}}, \qquad (1.28)\\
\frac{D_U}{Dt}\overline{\theta u_i} =& -\overline{\theta u_j}\frac{\partial U_i}{\partial x_j} - \overline{u_i u_j}\frac{\partial\Theta}{\partial x_j} - 2\epsilon_{ijk}\Omega_j\overline{\theta u_k} + \frac{1}{\theta_0}g\delta_{3i}\overline{\theta^2} \\
&- \frac{\partial\overline{\theta u_i u_j}}{\partial x_j} + \frac{1}{\rho_0}\overline{p\frac{\partial\theta}{\partial x_i}} - \frac{1}{\rho_0}\overline{p\frac{\partial\theta}{\partial x_i}} + \frac{\partial}{\partial x_j}\left(\nu\overline{\theta\frac{\partial u_i}{\partial x_j}} + \alpha\overline{u_i\frac{\partial\theta}{\partial x_j}}\right) - (\alpha+\nu)\overline{\frac{\partial\theta}{\partial x_j}\frac{\partial u_i}{\partial x_j}}, \qquad (1.29)\\
\frac{D_U}{Dt}\overline{\theta^2} =& -2\overline{u_i\theta}\frac{\partial\Theta}{\partial x_i} + \alpha\nabla^2\overline{\theta^2} - \frac{\partial\overline{\theta^2 u_i}}{\partial x_i} - 2\alpha\overline{\frac{\partial\theta}{\partial x_i}\frac{\partial\theta}{\partial x_i}}. \qquad (1.30)
\end{aligned}$$

All viscosity terms and the pressure gradient terms have been separated into two parts. The reason for this is that different parameterization assumptions will apply to them, see the description of the Mellor-Yamanda model later in this section. The interpretation of the terms on the right-hand side in Equation (1.28) is as follows: first and second terms are mean gradient production, third term is Coriolis production, fourth term is buoyant production, fifth and ninth term are viscous dissipation, sixth term is turbulent transport, and seventh and eighth terms are pressure gradient interaction. The interpretation of the terms in (1.29) and (1.30) is similar. Note that the terms in Equation (1.28) and (1.29) are arranged so that the last line contains all the terms that need to be parameterized, and the same goes for the last two terms in Equation (1.30).

Mellor-Yamanda model

The model description here is based on two articles by George L. Mellor [7] and by Mellor and Yamanda [8]. The model considers only a dry atmosphere, but was later generalized to take into account water vapor and liquid water [9]. Later, Nakanishi and Niino have made further developments on this model, and created the *MYNN* model, which will be described briefly in the following section.

The equations for the mean flow are essentially and the Reynolds averaged equations (1.10)-(1.12), except that Mellor and Yamanda does not consider radiation, so $R = 0$. In the Level 4 model, the equations (1.28)-(1.30) are used as prognostic equations for the turbulent fluxes. The tensor $\overline{u_i u_j}$ is symmetric and thus has six independent components. Further, $\overline{u_i \theta}$ has three components, whereas $\overline{\theta^2}$ is a scalar. Thus, this model consists of a total of 10 coupled differential equations in addition to the basic equations. To solve the equation system, however, one needs to parameterize the last 4 terms in Equation (1.28), the last 5 terms in Equation (1.29) and the last 2 terms in Equation (1.30). Below, we will go through the assumptions made by Mellor and Yamanda one by one.

It is assumed that pressure diffusional terms can be ignored, though, Mellor argues that this is questionable [7]

$$\overline{pu_i} = \overline{p\theta} = 0.$$

Mellor argues that Kolmogorov's *small-scale isotropy hypothesis* can be used to make following assumption about the viscous terms

$$2\nu \overline{\frac{\partial u_i}{\partial x_k}\frac{\partial u_j}{\partial x_k}} = \frac{2}{3}\frac{q^3}{\Lambda_1}\delta_{ij}, \quad (\alpha+\nu)\overline{\frac{\partial \theta}{\partial x_j}\frac{\partial u_i}{\partial x_j}} = 0, \quad 2\alpha\overline{\frac{\partial \theta}{\partial x_i}\frac{\partial \theta}{\partial x_i}} = 2\frac{q\overline{\theta^2}}{\Lambda_2}.$$

The idea is that the anisotropy associated with turbulence will vanish on smaller scales. Thus, each term is made proportional to an isotropic tensor, and the proportionality constant is formed by the quantities $q = \overline{u_i u_i}$, $\overline{\theta^2}$, and the length scales Λ_1 and Λ_2. Mellor argues that since a first order tensor cannot be isotropic, the right-hand side of the second equation above vanishes. Further assumption is made without noticeable mention[8]

$$\frac{\partial}{\partial x_j}\left(\nu\theta\overline{\frac{\partial u_i}{\partial x_j}} + \alpha\overline{u_i\frac{\partial \theta}{\partial x_j}}\right) = 0.$$

[8] In [7] Mellor carries the terms along until the final set of equations, but in [8] the terms have been neglected from the beginning, without reference to any assumption.

The third order terms are parameterized as follows[9]

$$\overline{u_i u_j u_k} = -q\lambda_1 \left(\frac{\partial \overline{u_i u_j}}{\partial x_k} + \frac{\partial \overline{u_i u_k}}{\partial x_j} + \frac{\partial \overline{u_j u_k}}{\partial x_i} \right),$$
$$\overline{\theta u_i u_j} = -q\lambda_2 \left(\frac{\partial \overline{\theta u_j}}{\partial x_i} + \frac{\partial \overline{\theta u_i}}{\partial x_j} \right),$$
$$\overline{\theta^2 u_i} = -q\lambda_3 \frac{\partial \overline{\theta^2}}{\partial x_i}.$$

Mellor assumed that these terms are proportional to the gradients of the second order terms, much in analogy to the concept of eddy diffusion, where the second order terms are proportional to gradients of first order terms. The proportionality constant is in each case the product of q and a length scale λ_1, λ_2 and λ_3, respectively.

The remaining two terms are parameterized as follows

$$\frac{1}{\rho_0} \overline{p \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)} = -\frac{1}{3l_1} \left(\overline{u_i u_j} - \frac{\delta_{ij}}{3} q^2 \right) + Cq^2 \left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right), \quad \frac{1}{\rho_0} \overline{p \frac{\partial \theta}{\partial x_i}} \frac{1}{\rho_0} = -\frac{q}{3l_2} \overline{\theta u_i},$$

where l_1 and l_2 are length scales, and C is a dimensionless constant. Inserting these parameterizations in Equation (1.28)-(1.30), the equation system is closed. We then need to determine the constants l_1, l_2, Λ_1, Λ_2, λ_1, λ_2, λ_3 and C. Mellor assumed that all length scales are proportional to one single *master length scale*, L. A discussion of the interpretation of this length scale and of several different proposals for functional forms of L is presented by Mellor and Yamanda [8], [9]. Without going further into detail, the form they use is presented here

$$L = \frac{\kappa z}{1 + \kappa z / l_0}, \quad \text{with} \quad l_0 = 0.10 \frac{\int_0^\infty z q dz}{\int_0^\infty q dz}, \tag{1.31}$$

where κ is the von Kármán constant. Referring to the results of yet another article, Mellor and Yamanda sets $\lambda_1 = \lambda_2 = \lambda_3 = 0.23L$. For the rest of the length scales, they define $l_1, l_2, \Lambda_1, \Lambda_2 = (A_1, A_2, B_1, B_2)L$, where the dimensionless proportionality constants then need to be determined. To determine the constants, Mellor restricts the analysis to the surface layer, where all fluxes are constant. From laboratory measurements of the turbulent fluxes, Mellor was then able to estimate the constants [7]

$$(A_1, A_2, B_1, B_2, C) = (0.92, 0.74, 16.6, 10.1, 0.08) \tag{1.32}$$

[9] It should be noted that there is a slight inconsistency in the notation between the articles. Therefore, the notation used here is a mixture of those.

Finally, we can write the equations for the Level 4 model by Mellor and Yamanda

$$\begin{aligned}\frac{D_U}{Dt}\overline{u_iu_j} =& -\overline{u_ju_k}\frac{\partial U_i}{\partial x_k} - \overline{u_iu_k}\frac{\partial U_j}{\partial x_k} - 2(\epsilon_{ikm}\Omega_k\overline{u_mu_j} + \epsilon_{jkm}\Omega_k\overline{u_mu_i}) + \frac{1}{\theta_0}(g\delta_{3i}\overline{\theta u_j} + g\delta_{3j}\overline{\theta u_i}) \\ &+ \nu\nabla^2\overline{u_iu_j} + \frac{\partial}{\partial x_k}\left[0.23qL\left(\frac{\partial\overline{u_iu_j}}{\partial x_k} + \frac{\partial\overline{u_iu_k}}{\partial x_j} + \frac{\partial\overline{u_ju_k}}{\partial x_i}\right)\right] \\ &- \frac{1}{3A_1L}\left(\overline{u_iu_j} - \frac{\delta_{ij}}{3}q^2\right) + Cq^2\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) - \frac{2}{3}\frac{q^3}{B_1L}\delta_{ij}, \end{aligned} \tag{1.33}$$

$$\begin{aligned}\frac{D_U}{Dt}\overline{\theta u_i} =& -\overline{\theta u_j}\frac{\partial U_i}{\partial x_j} - \overline{u_iu_j}\frac{\partial\Theta}{\partial x_j} - 2\epsilon_{ijk}\Omega_j\overline{\theta u_k} + \frac{1}{\theta_0}g\delta_{3i}\overline{\theta^2} \\ &+ \frac{\partial}{\partial x_i}\left[0.23qL\left(\frac{\partial\overline{\theta u_j}}{\partial x_k} + \frac{\partial\overline{\theta u_k}}{\partial x_j}\right)\right] - \frac{q}{3A_2L}\overline{\theta u_i}, \end{aligned} \tag{1.34}$$

$$\frac{D_U}{Dt}\overline{\theta^2} = -2\overline{u_i\theta}\frac{\partial\Theta}{\partial x_i} + \alpha\nabla^2\overline{\theta^2} + \frac{\partial}{\partial x_i}\left[0.23qL\frac{\partial\overline{\theta^2}}{\partial x_i}\right] - 2\frac{q\overline{\theta^2}}{B_2L}, \tag{1.35}$$

where again $q = \overline{u_iu_i}$, the constants are given by (1.32) and L is defined by (1.31). The process of deriving the Level 3, Level 2 and Level 1 model includes a scale analysis of all the terms in Equation (26)-(28), [8]. The approximations used in the Level 3 model leads to having only prognostic equations left for the variance terms $\overline{q^2}$ and $\overline{\theta^2}$, whereas diagnostic equations are used for all the covariances. Both the Level 2 and Level 1 models uses diagnostic equations for all variances and covariance and are thus first order closure models.

The conclusion made by Mellor and Yamanda was that the Level 3 model perform nearly as good as the Level 4 model, but it is much more efficient, since only 2 instead of 10 additional differential equations are needed. Therefore, it is the Level 3 model that was further developed to include water vapor and liquid water [9]. As described in Section 1.1.2, this generalization involves the use of *liquid water potential temperature* θ_l instead of θ and a prognostic equation for the total water content q_w. The Level 3 version then needs additional prognostic equations for the variance term $\overline{q_w^2}$ and the covariance term $\overline{\theta_l q_w}$. However, Mellor and Yamanda introduced a new Level 2.5 version, where only $\overline{q^2}$, i.e. TKE, is predicted with a prognostic equation, while all other second order terms are parameterized.

1.3.5 Mellor-Yamanda-Nakanishi-Niino model

Nakanishi and Niino has further improved the Mellor-Yamanda model, and their main contributions are estimating the closure constants based on large eddy simulation data and suggesting a new diagnostic expression for the turbulent length scale, L [10, 11]. The model has been tested against the original Mellor-Yamanda model and showed improved results [13]. The new master

length scale introduced by Nakanishi and Niino is defined as

$$\frac{1}{L} = \frac{1}{L_S} + \frac{1}{L_T} + \frac{1}{L_B}, \tag{1.36}$$

$$L_S = \begin{cases} \kappa z/3.7, & \zeta \geq 1 \\ \kappa z(1+2.7\zeta)^{-1}, & 0 \leq \zeta < 1 \\ \kappa z(1-100\zeta)^{0.2}, & \zeta < 0 \end{cases},$$

$$L_T = 0.23 \frac{\int_0^\infty z q dz}{\int_0^\infty q dz},$$

$$L_B = \begin{cases} q/N, & \partial\Theta_v/\partial z > 0 \text{ and } \zeta \geq 0 \\ \left(1 + 5(q_c/(L_T N))^{1/2}\right) q/N, & \partial\Theta_v/\partial z > 0 \text{ and } \zeta < 0 \\ \infty, & \partial\Theta_v/\partial z \leq 0 \end{cases},$$

where $\zeta = z/L_M$, and L_M is the Monin-Obukhov length defined in Equation (1.24). $N = ((g/\Theta_0)\partial\Theta_v/\partial z)^{1/2}$ is the Brunt–Väisälä frequency, which is the frequency of the oscillation due to an air parcel being displaced from its equilibrium [4, Ch. 3]. $q_c = \left((g/\Theta_0)\left(\overline{w\theta_v}\right)_0 L_T\right)^{1/3}$. The velocity scale q_c is similar to the convective length scale w_*, which is the typical vertical velocity related to convection in the PBL [1, Ch. 10].

This new turbulent length scale is constructed such that the shortest out of the three L_S, L_T and L_B will be dominant. L_S is the length scale in the surface layer and will only dominate close to the surface, since it scales with z. L_T depends on the turbulent structure of the PBL and is very similar to the master length scale in the original Mellor-Yamanda model. L_B is the length scale limited by buoyancy and is controlled by a combination of the local gradient of virtual potential temperature and the surface stability.

As well as with the Mellor-Yamanda model, the MYNN model exists in a Level 2.5 and a Level 3 version. Both are available in WRF, and both could therefore have been used in this study. However, it was difficult to find studies using the MYNN3 scheme, whereas there are several studies comparing the MYNN2.5 scheme to other PBL schemes in WRF [17, 18, 19]. The conclusions in these studies are similar; no PBL scheme is ideal for all scenarios. The MYNN2.5 scheme is, however, one of the best performing schemes in all these three studies. Therefore, the MYNN2.5 scheme has been selected to create the training data for this study, see Section 4.1.

Below, we show the parts of the MYNN2.5 PBL scheme, which are necessary to understand the development of the neural network based scheme described in Chapter 4. First, in addition to the mean field equations (1.10), (1.11), (1.20), (1.21), we define the TKE equation as it appears in the MYNN model [13]

$$\frac{\partial q^2}{\partial t} = -\frac{\partial}{\partial z}\left(\overline{wq^2} + 2\frac{\overline{wp}}{\rho_0}\right) - 2\left(\overline{wu} + \frac{\partial U}{\partial z}\overline{wv}\frac{\partial V}{\partial z}\right) + 2\frac{g}{\Theta_0}\overline{w\theta_v} - 2\varepsilon, \tag{1.37}$$

where $q = \sqrt{u^2 + v^2 + w^2}$ is the square root of twice the TKE. The first term on the right-hand side contain both turbulent transport and pressure transport of TKE, the second term is shear production, the third term is buoyant production, and the fourth term is the viscous dissipation rate. The turbulent fluxes are parameterized as

$$\begin{aligned} \overline{wu} &= -K_m \frac{\partial U}{\partial z}, & \overline{w\theta_l} &= -K_h \frac{\partial \Theta_l}{\partial z}, \\ \overline{wv} &= -K_m \frac{\partial V}{\partial z}, & \overline{wq_w} &= -K_h \frac{\partial Q_w}{\partial z}, \\ \overline{wq^2} + 2\frac{\overline{wp}}{\rho_0} &= -3K_m \frac{\partial q^2}{\partial z}, & \overline{w\theta_v} &= \beta_\theta \overline{w\theta_l} + \beta_q \overline{wq_w}, \end{aligned} \tag{1.38}$$

where K_m and K_h are the turbulent diffusivities for momentum and scalar quantities, respectively, and β_θ and β_q are functions related to condensation and evaporation processes, which links the flux of virtual potential temperature to the flux of conserved quantities. The reason why the flux of virtual potential temperature is not simply parameterized as $\overline{w\theta_v} = -K_h \frac{\partial \Theta_v}{\partial z}$, is that virtual potential temperature is not conserved under condensation and evaporation. For further details, see the formulation of the *partial-condensation* scheme by Sommeria and Deardorff [14]. K_m and K_h are both functions that depend on q, L, β_θ, β_q as well as variables related to local stability and wind shear. For a complete description, the reader is referred to either of [11, 13], but two of the variables, which the diffusivities depend on, are worth mentioning

$$G_m = \frac{L^2}{q^2}\left(\left(\frac{\partial U}{\partial z}\right)^2 + \left(\frac{\partial U}{\partial z}\right)^2\right), \qquad G_h = -\frac{L^2}{q^2}\frac{g}{\Theta_0}\left(\beta_\theta \frac{\partial \Theta_l}{\partial z} + \beta_q \frac{\partial Q_w}{\partial z}\right). \tag{1.39}$$

In addition, the dissipation term in the TKE equation is parameterized as

$$\varepsilon = \frac{q^3}{B_1 L}, \tag{1.40}$$

where L is the master length scale from Equation (1.36), and $B_1 = 24$ is a closure constant.

Chapter 2

The WRF model

The Weather Research and Forecast model, *WRF*, is an open source regional numerical weather prediction model widely used for research and weather forecasting. The description here is based on the official WRF user guides [20, 21]. The overall structure of the model is illustrated in Figure 2.1.

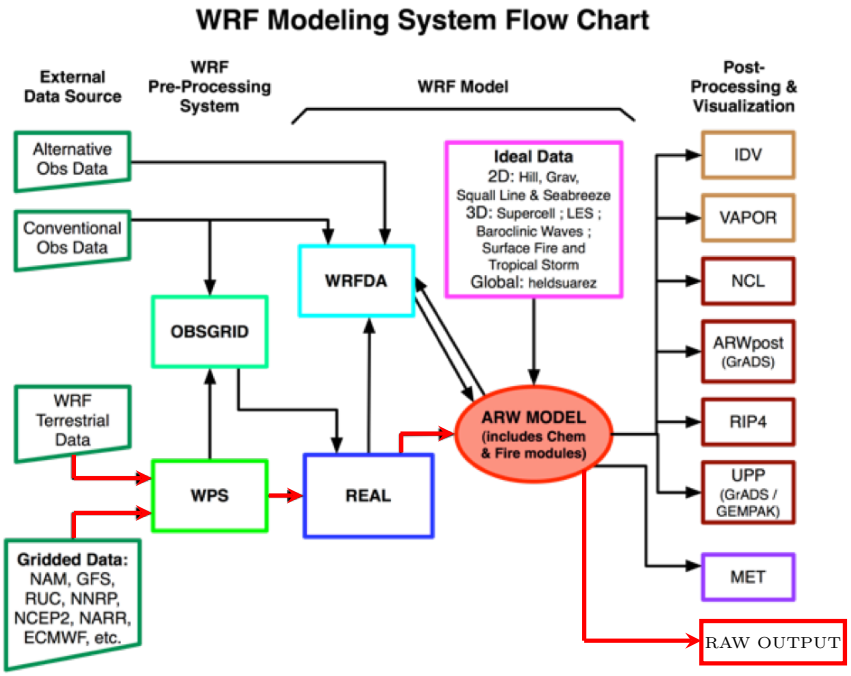


Figure 2.1: Flow chart illustrating the different elements of WRF. The flow chart shows elements that are not used in this study, so the relevant parts have been highlighted with the red arrows. Figure source: [20].

The red arrows in Figure 2.1 indicate which parts of the WRF system are used in this study. The WRF Preprocessing System, *WPS* prepares the data and interpolates it to a specified horizontal grid. *REAL* then interpolates the data to specified vertical levels, before it is fed to Advanced Research WRF, *ARW* as initial and boundary conditions.

The sections 2.1 and 2.2 describe the main components of WRF. The latter will go briefly through the main features of the ARW system and then describe how the PBL parameterization is implemented. Section 2.3 describes the model configurations used for the simulations.

2.1 WRF Preprocessing System

WRF comes with its own data preprocessing system *WPS*, which takes meteorological and terrestrial data as input and interpolates it to the horizontal grid points.

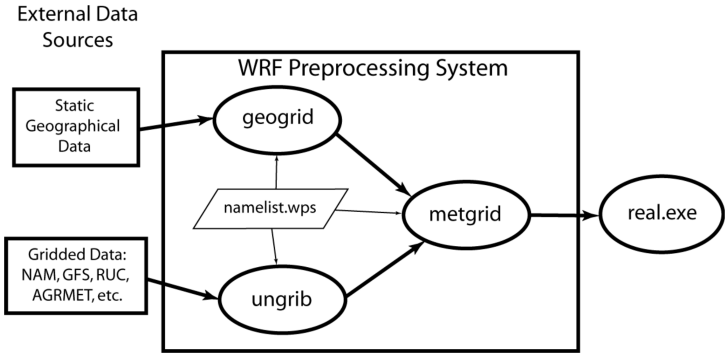


Figure 2.2: Flow chart illustrating the different elements of WPS. Figure source: [20].

Figure 2.2 show the different elements of WPS. First, one must provide files containing gridded meteorological data for initial and boundary conditions. For this purpose, we use operational final analyses from The Global Forecast System, *GFS*, produced by the National Center for Environmental Prediction, *NCEP* [25]. This dataset provides global tropospheric analyses and forecasts on $0.25^\circ \times 0.25^\circ$ grids for every sixth hour starting from 2015-07-08.

Next, the physical domain must be defined, which involves specifying the type of map projection. Several different types of projection are available in WPS, which are suitable for different areas on the globe. Here, we concentrate on the Lambert Conformal mapping, which is specifically developed for mid-latitudes.

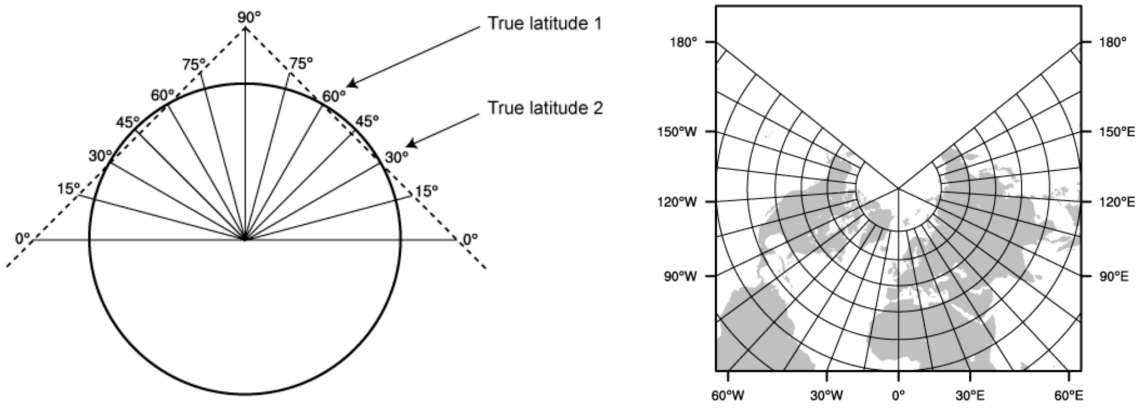


Figure 2.3: Illustration of Lambert conformal mapping. Figure source: [20].

The idea of Lambert conformal mapping is to project the surface of the earth onto a cone, which aligns with the rotation axis of the earth, as illustrated in Figure 2.3 (left). The specified true latitude(s) determine, where the cone intersects with the surface of the earth. If only one true latitude is specified, the cone will be a tangent to the earth at this latitude. The resulting grid is illustrated in Figure 2.3 (right).

The type of projection as well as the size and resolution of the horizontal grid are specified in the file `Namelist.wps`. In this file, one must also specify the dates and times corresponding to the reanalysis files used. The `geogrid.exe` program then uses the information from the `Namelist.wps` file together with files containing information about terrestrial data to create the simulation domain. The `ungrib.exe` program takes the reanalysis data, provided in GRIB format, unpacks it and writes it to an intermediate file format. Finally, the `metgrid.exe` program interpolates the data from the intermediate files to the simulation domain and creates NetCDF files, as input for the WRF `real.exe` program.

2.2 Advanced Research WRF

The Advanced Research WRF, *ARW* provides a range of idealized test cases as well as simulations using *real* weather data. For the latter, the program `real.exe` takes the NetCDF files created by the `metgrid.exe` program and interpolates the meteorological data to the vertical model levels used. The 3D grid is specified in the `Namelist.input` file, and this must correspond to the horizontal grid specified in the `Namelist.wps` file, i.e. the number of grid points and the spatial resolution must be the same. For the vertical model levels, WRF uses a terrain following vertical coordinate, η, defined as

$$\eta = \frac{p_h - p_{ht}}{\mu}, \quad \text{where} \quad \mu = p_{hs} - p_{ht}, \tag{2.1}$$

where p_h is the hydrostatic component of the pressure and p_{hs} and p_{ht} are the pressures along the surface and the top boundaries. η thus varies from 1 to 0, with $\eta = 1$ following the surface of the earth and $\eta = 0$ following a fixed top pressure surface, see illustration in Figure 2.4.

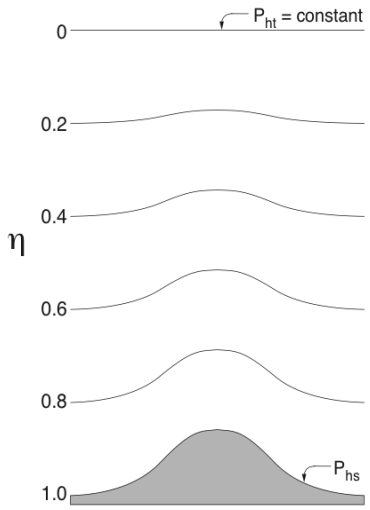


Figure 2.4: Illustration of the terrain following vertical coordinate, η. Figure source: [21].

The variable $\mu(x, y)$ represents the mass in an air column per unit area at location (x, y). WRF uses the flux form of the variables

$$\mathbf{V} = \mu\mathbf{v} = (U, V, W), \quad \Omega = \mu\omega = \mu\frac{\partial\eta}{\partial t}, \quad \Theta = \mu\theta, \tag{2.2}$$

where $\mathbf{v} = (u, v, w)$ and θ denote the mean field velocity and temperature fields, respectively. Similarly, ω is the vertical "velocity" parameter in η coordinates. The capital letter variables $\mathbf{V}$, Ω and Θ denote the flux form variables. Note that the notation used here should not be confused with the notation for the PBL equations derived Chapter 1, which will be used in the remaining chapters of the thesis.

ARW uses an Eulerian solver meaning that changes are computed for fixed positions in a stationary grid, instead of for air parcels following the mean flow (as the equations derived in Chapter 1). Further, the model is non-hydrostatic and fully compressible. The first means that vertical velocity is a prognostic variable instead of assuming hydrostatic balance (as in Equation (1.1)). The latter means that the "full" continuity equation describes the mass conservation instead of using the Boussinesq approximation (as in Equation (1.4)). The equations include forcing terms arising from physics parameterizations, which will be described 2.2.1.

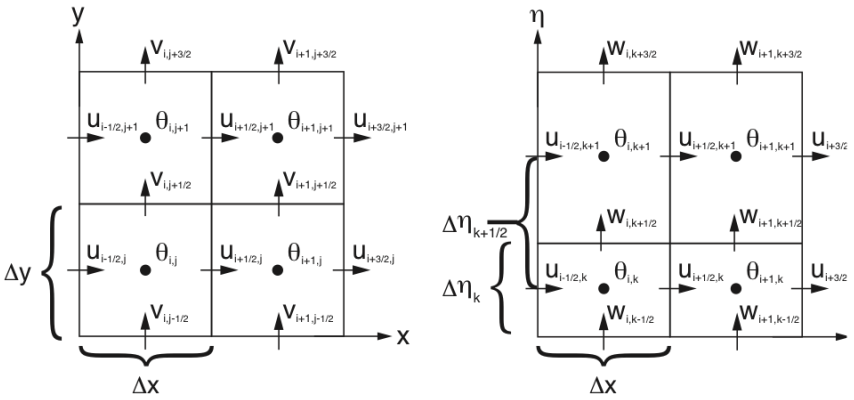


Figure 2.5: Spatial discretization, horizontal (left) and vertical (right). Figure source: [21].

For the spatial discretization, ARW uses an *Arakawa C grid*, which means that the velocity fields are defined staggered compared to the remaining physical variables, see Figure 2.5. Each velocity component is staggered only in the dimension of the component itself. The points where θ is defined are referred to as *mass points*, whereas the points where the velocities are defined are referred to as u, v and w *points*, respectively. In the horizontal dimensions, the grids are staggered regularly in both dimensions, while in the vertical, the distances between η surfaces are not constant throughout the atmosphere. The mass points in the vertical are also called *half* η levels, and are located halfway between the two neighboring *full* η levels. The "distance" between two η surfaces is measured in the η coordinate and should not be interpreted as the physical distance. The full η surfaces must be specified by the user in the `Namelist.input` file.

ARW uses a third-order Runge-Kutta time integration scheme for low-frequency modes, i.e. those important for meteorology. To ensure numerical stability, the faster modes such as acoustic waves are integrated over smaller time steps. Generally, for Eulerian models, there is a strong constraint on the duration of the time step. To avoid extrapolation, an air parcel, or a propagating wave, should not be allowed to travel longer than the distance between two neighboring grid points. In practice, the time step for the low-frequency modes, Δt, is constrained by the advective Courant number $C = u\Delta t/\Delta x$, where u is the advection speed, and Δx is the distance between two neighboring gridpoints. The maximum stable Courant number for the third-order Runge-Kutta scheme, depends on the order of the spatial discretization. The numerical stability, however, depends on the vertical resolution and the magnitude of the vertical velocity as well. The latter depends on the horizontal resolution, since finer resolutions permitting convective cells, typically $x \leq 5$ km, will generally produce larger vertical velocities. As a rule of thomb, the time step for the low-frequency modes, Δt, should not exceed $6 \times \Delta x$ seconds, where Δx is the spatial resolution measured in km. For a horizontal resolution of $\Delta x = 10$ km, this gives $\Delta t = 60$ seconds. This time step must be specified in the `Namelist.input` file, while the time

step for the fast modes is set automatically to a fraction of Δt. For further details on the stability constraint, as well as generally about the temporal and spatial discretization, see [21, Ch. 3].

In general, when interpolating weather data to a discrete grid, the fields will not be balanced. To reduce this initial imbalance, ARW provides a *digital filtering initialization*, DFI [21, Ch. 5.3]. The ARW DFI works by assuming that all noise from initial imbalance is of higher frequency than the important, meteorological modes. Therefore, in principal, it just works as a low-pass filter removing all frequencies higher than a certain cutoff frequency, however, in practice, the DFI involves forward and backward integration around the initial time. Configurations related to the digital filter must be specified in the `Namelist.input` file.

2.2.1 Physics parameterizations

In addition to the dynamical model, ARW provides parameterization models for all non-resolved physics [21, Ch. 8]. This includes PBL parameterization, surface-land interaction, radiation, cumulus parameterization and microphysics. For each of the physics parameterizations, ARW provides a range of different schemes allowing for many different setup. One option is to use a pre-defined "physics suite", where the different physics parameterizations have been selected to be suitable for a specific weather problem. An example is the *CONUS* suite, which is designed to work well for forecasts covering the contiguous United States. One can easily combine parts of the physics suite with explicitly defined physics parameterizations. As mentioned earlier, the dynamical equations include forcing terms, which arise from the physics parameterization. In practice, each physics parameterization computes the tendencies for all relevant physical variables, e.g. the tendency for potential temperature due to turbulence is of the form $\partial\Theta/\partial t|_{PBL} = -K_h\partial^2\Theta/\partial^2 z$, where the expression for K_h of course depends on the PBL scheme used.

Section 1.3 described how the equations for different approaches to parameterizing the turbulent fluxes. However, it was not described how such a parameterization is implemented in discrete numerical model. Here is therefore a brief description of how the PBL parameterization is implemented in the WRF setup. Before the PBL scheme is called, the horizontal velocities are interpolated to the θ points. Hence, all physical variables, except vertical velocity, are defined in the θ points, i.e. the half η levels. To compute the tendencies in the θ points, however, we need to know the turbulent fluxes in and out of each grid box, i.e. the fluxes at the grid box walls (full η levels). Since the lowest full η level is following the surface, this needs to be treated differently from the remaining levels. Therefore, in practice, ARW first calls a separate surface flux scheme, which computes the surface fluxes and passes them as input the PBL scheme. The

PBL scheme then computes the turbulent diffusivities in all remaining full η levels, except the top level, where they are assumed to be zero (this is illustrated in Figure 4.2 in Section 4.3). Finally, the PBL scheme solves the diffusion equation for each relevant physical variable and then outputs the tendencies.

2.3 Model setup

As mentioned in the introduction, this study focuses on regional forecasts for Scandinavia, and a small domain covering most of Scandinavia and the British Islands are used for the creating the training dataset, see Section 4.1. A larger model domain covering most of Europe is used for testing the neural network based PBL scheme, see Section 5.1. It is assumed that the physics parameterizations in the CONUS physics suite, will be suitable for the weather in these domains, since they cover latitudes similar to contiguous United States. The CONUS physics suite is therefore used for physics parameterizations, except the surface flux scheme and the PBL scheme for which the MYNN schemes are used.

We use a 10-km horizontal resolution. One argument for this relatively coarse resolution is to avoid the necessity of *nesting*. The model domain is already nested into its lateral boundary data, meaning that is "fed" with information about the physical variables at the boundaries. But ARW provides the option of additional nesting, where a smaller domain with a higher resolution can be nested in the parent domain. The point of nesting is both to avoid noise from interpolating coarse lateral boundary conditions to a fine grid and to save some computations by only using the high resolution grid for the region of interest [20, Ch. 7]. The resolution of the reanalysis data used for initial and boundary conditions is $0.25^\circ \times 0.25^\circ$, which approximately corresponds to $\sim 28\text{km} \times 28\text{km}$. Thus, with a 10-km horizontal resolution we avoid a large ratio between the resolutions of the lateral boundary conditions and the horizontal domain grid.

The vertical resolution, on the other hand, is important for resolving the boundary layer dynamics. Therefore, we specifically use a high vertical resolution in the lowest part of the atmosphere and a relatively coarse resolution far away from the surface. One can either manually specify all the η levels or use ARW's built-in algorithm to automatically generate the vertical levels based on configurations specified in the `Namelist.input` file. The latter option was used in this study. We use a total of 41 full η levels, the distance between the two lowest levels was set to 10 m, and the maximum allowed thickness was set to 1000 m. Additionally, a stretching factor needs to be specified, which determines how "quickly" the distance between the η levels grow. For details, see [20, Ch. 5]. With these configurations, the first full η level above the ground is located at $\sim 10\text{m}$, and 14 out of the 41 levels are located within the lowest kilometer of the atmosphere.

In the `Namelist.input` file, it is also possible to specify, which variables one wishes to extract

from WRF and how often they should be written to the output NetCDF file. Examples of the files `Namelist.wps` and `Namelist.input`, corresponding to the configurations described above, are shown in Appendix A.

Chapter 3

Artificial neural networks

A neural network is a specific type of machine learning model, which, as the name suggests, can be thought of as a (very simplified) mathematical representation of information processing in the brain. Today, neural networks exist in many different forms developed to solve different types of problems. Here, the description is restricted to the specific type of network *architecture* used in the study, the *feedforward* neural network. This is the simplest type of neural network, but it is well suited for a wide range of regression problems. The descriptions in this chapter are mainly based on following textbooks on the subject [28, 26] and [27, Ch. 5]. First, Section 3.1 presents a mathematical description of the feedforward neural network, and then Section 3.2 describes the basic theory of *training* neural networks.

3.1 The feedforward neural network

The feedforward neural network is essentially a sequence of *layers*, each consisting of a number of *neurons* or *nodes*. The layers are fully connected, meaning that each node is connected to all the nodes in the neighboring layers. Neural networks are called *deep*, if they consist of two or more *hidden layers*. An illustration of a simple deep neural network is shown in Figure 3.1.

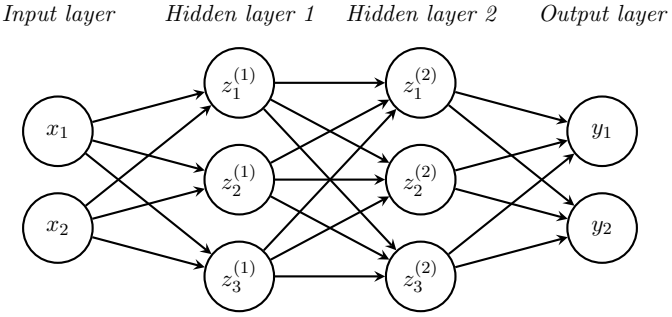


Figure 3.1: Simple feedforward neural network with two input variables, two output variables, and two hidden layers, each with three nodes. The arrows show which nodes are connected and the direction of the information flow in the network.

Each node is represented by a number, which depends on the values of the nodes in the previous layer. Thus, the information is only *fed forward* to the next layer. The very first layer of nodes contains the input variables arranged in a vector $z_i^{(0)}$, and the nodes in all subsequent layers are computed as

$$\begin{aligned} z_j^{(n)} &= h\left(a_j^{(n)}\right), \quad \text{where} \\ a_j^{(n)} &= w_{ji}^{(n)} z_i^{(n-1)} + b_j^{(n)}. \end{aligned} \tag{3.1}$$

The superscript $^{(n)}$ implies that the vector $z_i^{(n)}, n = 1, 2, 3, ..., N$ contains the values of the nodes in the n'th layer. The elements of the vector $a_i^{(n)}$ are linear functions of the nodes in the preceding layer, where the constants $w_{ji}^{(n)}$ are called the *weights*, and the constant $b_j^{(n)}$ is called the *bias*. The subscript $_{ji}$ implies that $w_{ji}^{(n)}$ is the constant we multiply with $z_i^{(n-1)}$ to compute $z_j^{(n)}$.

The function $h(\cdot)$ is called the *activation* and is a nonlinear and differentiable function. The role of the activation is to allow the neural network to learn nonlinearities, since if the activations were linear functions, the neural network, regardless of number of layers and neurons, would reduce to a linear regression model. The activation does not have to be the same for all layers, and often the activation for the output layer will differ from the one used in the remaining layers. If the purpose is to make a classification model, the activation function is typically constructed so that the output vector consists of positive numbers summing up to 1. Hence, each element of the output vector can be interpreted as the probability of a given output. A typical example of this is a model for classifying handwritten digits. Given a dataset of labelled examples of handwritten digits, the neural network then predicts the probability of the digit belonging to each of the ten classes $0, 1, 2, ..., 9$.

For regression models, on the other hand, the activation function for the output layer is typically the identity function $h(a_i) = a_i$, allowing the output variables to vary freely. The problem studied in this project is an example of a regression problem, where a set of turbulent quantities are assumed to depend on the mean field variables. The strength of a neural network is that it does not assume the functional form but can in principal emulate any function. Hence, for regression problems, a neural network with $N - 1$ hidden layers[1], the output values are $z_j^N = w_{ji}^{(N)} z_i^{(N-1)} + b_j^{(N)}$. Combining this relation with Equation (3.1), and denoting the input and output vectors x_i and $\hat{y}_i$, the complete feedforward neural network can be written as the

[1]The reason for using $N - 1$ instead of N *hidden* layers is that the output layer then is the Nth layer.

following recurrence relation

$$\begin{aligned} \hat{y}_j &= w_{ji}^{(N)} z_i^{(N-1)} + b_j^{(N)}, \\ z_j^{(n)} &= h\left(a_j^{(n)}\right) \quad \text{for} \quad n = 1, 2, ..., N-1 \ , \quad \text{where} \\ a_j^{(n)} &= w_{ji}^{(n)} z_i^{(n-1)} + b_j^{(n)}, \\ z_i^{(0)} &= x_i. \end{aligned} \tag{3.2}$$

From Equation (3.2), we see that the number of biases grow linearly with the total number of nodes in the network. On the other hand, the number weights needed to predict the values of a layer is the number of elements in the weight matrix, i.e. the product of the number of nodes in the layer itself and the preceding layer. As an example, the network in Figure 3.1 has 27 model parameters. However, if the number of nodes in the hidden layers is increased by a factor of 10, then the total number of model parameters becomes 1080, which is an increase by a factor of 40. It can be deduced that for deep neural networks, where hidden layers are much larger than the input and output layers, the total number of weights is approximately proportional to the square of the number of nodes per layer in the hidden layers.

As mentioned, the activation is what enables the neural network to learn nonlinearities, and therefore it is important to discuss its role and interpretation in greater detail. As an example, consider an activation that is simply a step function

$$h(a) = \begin{cases} 0 & \text{if} \quad a < 0 \\ 1 & \text{if} \quad a \geq 0 \end{cases} .$$

This is somewhat similar to a very simple conceptual model of a brain, where a neuron can either send an impulse or be silent. This example, although not of much practical use, leads to a nice interpretation of the weights and biases. Since all neurons can only send signals of the same strength, the role of the weights is then to determine the importance of the signal from each of the neurons in the preceding layer. The bias, on the other hand, affect the overall sensitivity of the neuron, determining how likely it is to send an impulse to the subsequent layer. In practice, the step function is impractical to use, since its derivative is zero everywhere except at $a = 0$, where it is not defined. This can make optimization in a large parameter space very inefficient. A typical activation function is the logistic *sigmoid* function

$$h(a) = \frac{1}{1 + e^{-a}}. \tag{3.3}$$

The sigmoid function also varies between 0 and 1, but it has a derivative that is different from

zero. Thus, the interpretation of the weights and biases are almost the same as with the step function, but the weights and biases can be optimized using gradient descent, which eases the training. However, a disadvantage of the sigmoid function is that it "saturates" for large numeric values of a, i.e. it converges to a constant value. This means that the derivatives go to zero when the weights become large, and this can potentially slow down the training significantly. This makes it harder to train deep neural networks in particular [28]. Therefore, a popular alternative activation is the $ReLU$ function (rectified linear unit)

$$h(a) = \begin{cases} 0 & \text{if} \quad a < 0 \\ a & \text{if} \quad a \geq 0 \end{cases} . \tag{3.4}$$

The sigmoid and ReLU activations described above are shown in Figure 3.2, together with two other popular activation functions, the hyperbolic tangent function, *tanh* and the *leakyReLU*.

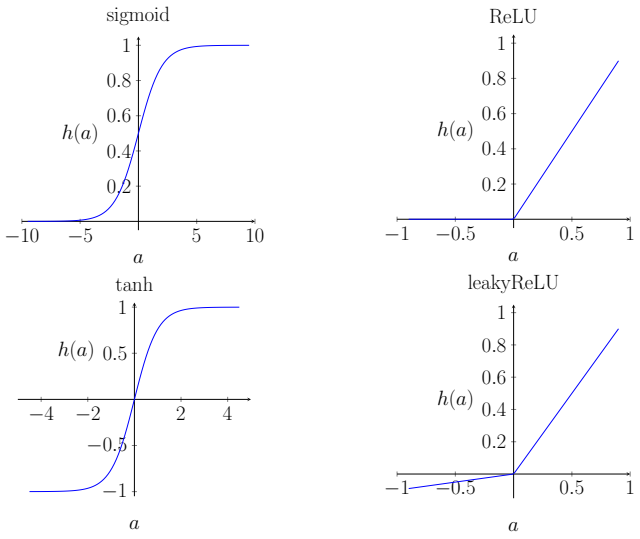


Figure 3.2: Four different activation functions.

The leakyReLU is very similar to the normal ReLU function, but instead of having a constant value for $a < 0$, it has a slightly positive slope. This is supposed to prevent problems related to the ReLU function's gradient being zero for $a < 0$. However, generally the activation function, as well as the number of layers, and number of nodes in each layer are *hyperparameters*, which need to be optimized to best solve the specific problem.

3.2 Training the network

Assume, we have a training set of M input vectors $\{\mathbf{x}_m\}$ with corresponding target vectors $\{\mathbf{y}_m\}$, for $m = 1, 2, ..., M$. Note that M is the number of data points and thus unrelated to the dimensions of the input and output vectors. $\mathbf{x}_m$ and $\mathbf{y}_m$ are the m'th input and output vectors, which respectively contain the relevant input and output variables for the m'th data point. To exemplify this, consider the specific case of a turbulence parameterization model: One could imagine a training dataset consisting of a large number of samples, e.g. $M \sim 10^6$. Each target vector could then consist of the diffusivities K_m and K_h for a point in the 3D grid of the model domain at a specific time, and the input vector could contain variables related to stability and wind shear, e.g. $\partial\Theta_v/\partial z$ and $(\partial U/\partial z)^2 + (\partial V/\partial z)^2$.

We now denote the neural network from Equation (3.2) as

$$\hat{\mathbf{y}}_m = f(\mathbf{x}_m, \mathbf{W}),$$

where all the weights and biases, $w_{ij}^{(n)}$ and $b_i^{(n)}$, are stacked in the vector $\mathbf{W}$ for compact notation. We then define the *loss function*, which is a measure of the distance between the target values and the predictions of a neural network with model parameters $\mathbf{W}$

$$E(\mathbf{W}) = \frac{1}{M}\sum_m e(\mathbf{y}_m, \hat{\mathbf{y}}_m),$$

where $e(\cdot)$ is some function measuring the error of a single prediction. For regression problems, common loss functions are the mean squared error or the mean absolute error

$$E(\mathbf{W}) = \frac{1}{M}\sum_m \|\mathbf{y}_m - \hat{\mathbf{y}}_m\|^2, \tag{3.5}$$

$$E(\mathbf{W}) = \frac{1}{M}\sum_m \|\mathbf{y}_m - \hat{\mathbf{y}}_m\|. \tag{3.6}$$

Since both Equation (3.5) and (3.6) are differentiable and the neural network consists only of differentiable functions, we can analytically compute the derivative of the loss function with respect to each of the model parameters. Hence, the model can be optimized using gradient descent by computing $\frac{\partial E}{\partial W_i}$ and then iteratively adding a small adjustment to the model parameters

$$W_i^{new} = W_i - \epsilon\frac{\partial E}{\partial W_i}, \tag{3.7}$$

where the *learning rate* ϵ is a positive real number[2]. The algorithm used for computing the

[2]The actual size of ϵ depends on the choice of optimization algorithm, the method of data normalization, and the problem in general.

gradient is called *back-propagation* and is essentially just application of the chain rule for partial differentiation [28]. In the following section, we will discuss some challenges with using gradient descent and describe a popular alternative approach *stochastic gradient descent*, or simply *SGD*.

3.2.1 Stochastic gradient descent

Intuitively, using gradient descent to iteratively update the weights and biases using Equation (3.7) may seem like a good approach. However, in practice the method has several limitations. For a large data set, it is generally computationally demanding to compute the gradient of the loss function, since it depends on all data points. Therefore, this method can be very slow. In addition, since the loss function is generally nonlinear and non-convex, optimizing a neural network is a complex task and the gradient-based learning will often benefit from some stochastic element allowing it to more easily escape local minima and saddle points. For these reasons, stochastic gradient descent is a popular alternative to the standard gradient descent based learning. It essentially means that only a subset, or *mini-batch*, of the training data is used, when the gradients are computed. This diminishes the computational cost of each iteration while introducing a stochastic element, since the gradient is estimated based on a random subset of the data. The SGD algorithm thus updates the weights

$$W_i^{new} = W_i - \epsilon \frac{\partial E_B}{\partial W_i}, \tag{3.8}$$

where E_B is the loss computed based on the mini-batch of size B

$$E_B(\mathbf{W}) = \frac{1}{B} \sum_{m=1}^{B} e(\mathbf{y}_m, \hat{\mathbf{y}}_m).$$

The SGD algorithm then iterates over all the mini-batches, until the model has *seen* all the data points. We call the number of iterations needed to encounter all data points exactly one time an *epoch*. The number of epochs needed to find a sufficiently good minimum of the loss function depends on the specific problem and is therefore yet another hyperparameter. Typically, the easiest way to evaluate whether a model has trained for long enough, is to look at the *learning curve* and visually determine if the model is still learning. In Figure 3.3, a sketch shows examples of how learning curves for two different models might look. The example shows how the losses decrease as function of number of epochs but also illustrates how two different models can learn at different rates. In practice, this could of course be related to the learning rate parameter ϵ, but it could just as well be due to different choices of other hyperparameters. The thick solid lines show the loss on the training data, while the thin solid lines show the loss on an independent validation dataset. It is common practice to evaluate the validation loss continuously and plot

it together with the loss on the training data. This makes it easy to visualize whether the model overfits to the training dataset. In the case of overfitting, the *training loss* will continue decreasing, while the *validation loss* stabilizes or even starts increasing again. In the sketch in Figure 3.3, model 2 is clearly still learning, while the learning curve for model 1 is close to convergence.

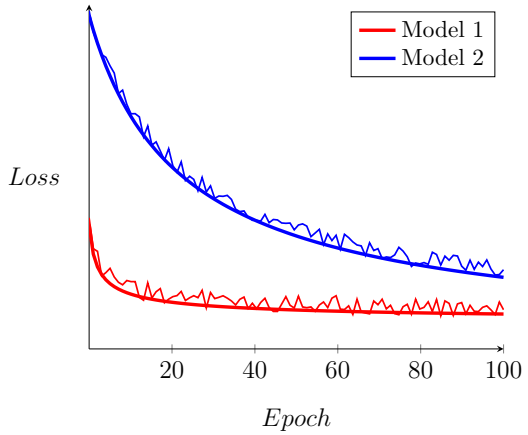


Figure 3.3: Sketch of learning curves for two different models. The thick solid lines show the loss on the training data for each of the two models, while the thin solid lines show the loss on the validation data.

The validation data is thus used to evaluate the model during training, and it is also common practice to select the "best" model as the model that performs best on the validation dataset, rather than selecting the model after the last update of the weights. Therefore, it is a good idea to introduce a third independent *test* dataset for estimating the error on the predictions of the selected "best" model. This is necessary because the model is selected based on the performance on the validation dataset, and, hence, this can no longer be used to give an independent and unbiased estimate of the performance [26].

The SGD algorithm is the basis of a range of different optimization algorithms, which combine the concepts of stochastic gradient-based learning with other elements, such as a *momentum* parameter and adaptive learning rates. The momentum algorithm introduces a velocity parameter v and a new hyperparameter $\alpha \in [0, 1[$, which determine how quickly the contributions of the previous gradients will decrease. In other words, α determines how easy it is for the gradient to

change direction from iteration to iteration.

$$W_i^{new} = W_i + v, \tag{3.9}$$
$$v = \alpha v^{old} - \epsilon \frac{\partial E_B}{\partial W_i}.$$

Note that if $\alpha = 0$, Equation (3.9) reduces to (3.8). The point of the momentum is to reduce some of the noise related to the stochastic gradients. Different variations of Equation (3.9) combined with different approaches to make an adaptive learning rate, has resulted in many different optimization algorithms. An example is the Adaptive Moment Estimation, or simply the *Adam* optimizer [29]. Any optimizer introduces a new set of constants that must be tuned by the user to get the best result. This means that the efficiency of an optimization algorithm partly depends on the user's familiarity with the specific algorithm and its hyperparameters. The Adam optimizer, however, is considered an all-round good choice, because it is fairly robust when just using the default hyperparameters, except for the initial learning rate [26, Ch. 8]. If this assumption is accepted, Adam is quite easy to work with, since it only requires tuning of one hyperparameter, namely the learning rate.

Worth mentioning is also the concept of learning rate scheduling, meaning that the learning rate is manually set to vary in some way during the training. One approach is to decrease the learning rate when the model is getting close to a sufficiently good minimum. In that case, a smaller learning rate will help fine-tune the model parameters [26, Ch. 8]. A different approach is to let the learning rate varies periodically throughout the training. Smith suggests a cyclic triangular learning rate schedule, where the learning rate vary periodically in a triangular pattern [30]. The motivation for this idea is that occasional high learning rates will help the model "escape" from local minima and saddle points.

As indicated in this chapter, optimizing a neural network is more than just determining the values of the model parameters. More importantly, it consists of determining the optimal combination of hyperparameters. For some hyperparameters, one may rely on the experience of other studies, but, generally, testing the different options is best, since it may differ from problem to problem.

All neural networks have been trained in Python [33] using the Keras library [35], which is a high-level neural networks API[3], using Tensorflow as backend [34].

[3] Application programming interface

Chapter 4

Development and optimization of the model

When solving physical problems using machine learning, there are often many different possible approaches, and in the beginning one will likely test many different ideas, and this trial and error can be important to get onto the right track. Therefore, it is also impossible to give a comprehensive description and documentation of all the ideas, which have been tested during this study. Instead, some of the different possible approaches will be described, and we will try to discuss their pros and cons to motivate some of the choices made. Other parts of the process will of course be described in greater detail.

In Section 4.1, we briefly describe how the training dataset was generated. In Section 4.2, we discuss different options for the overall model construction, especially in terms of choice of model output. Next, in Section 4.3, we describe the process of determining the input variables for the model. Finally, in Section 4.4 the model training and hyperparameter optimization are described.

4.1 Creating the dataset

The datasets for training the neural networks was generated using the WRF model with the configurations described in Section 2.3. The MYNN scheme was used for both the surface layer scheme and the PBL scheme [13]. To extract the relevant variables from WRF, it was necessary to introduce a set of new variables in the WRF Registry and add each of these to the argument lists of all relevant Fortran subroutines. For a description of the WRF Registry, see [20, Ch. 8]. These new variables were assigned their values directly in the MYNN module just before and after the computation of the diffusivities. For details on the MYNN Fortran module and how this is implemented in the WRF code, see [22]. The relevant variables are described in the

Sections 4.2 and 4.3.

The domain, shown in Figure 4.1, is covering most of the British islands and Scandinavia, and the area covers a range of different surface types, ensuring large variation in surface conditions for the PBL scheme and surface flux scheme, see Appendix B. The combination of a relatively small geographical area and a horizontal resolution of 10 km allows for efficient data generation. However, since a proper computer resource for training the neural networks has not been available for this project[1], it is not necessarily of much use to create large amounts of data, since this means the training of the neural networks becomes slow.

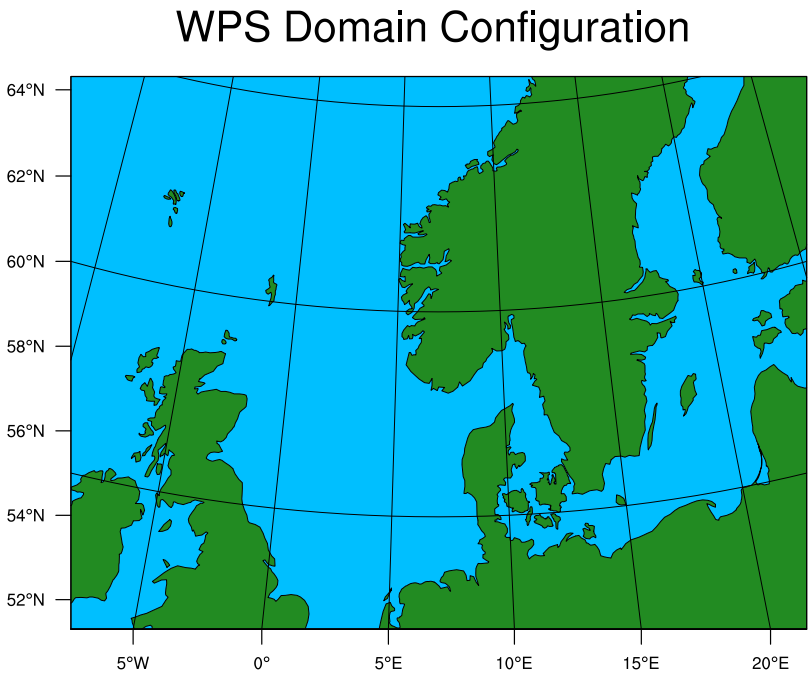


Figure 4.1: Domain used for generation of training data.

How to sample the dataset is not trivial and requires knowledge about both the specific problem and the type of machine learning model used. Essentially, a machine learning model learns to emulate the statistics of the training data, and therefore the dataset should be constructed in a way such that its variability resembles that of the actual physical system. One way to ensure natural variability in the dataset, is to randomly sample the data points from simulations with randomly chosen initial times. However, randomly sampling the dataset might require many different simulations before the weather extremes are represented. So, to limit the amount of

[1] All the neural networks have been trained using a standard laptop.

data, a pseudo-random data sampling was used to maximize variability in the dataset with only a few simulations. Six days were chosen, two in summer months, two in winter months and two transitional. The specific days were not analyzed in detail but some were chosen specifically from periods with extreme hot or cold weather. This way of sampling, of course, means that the variability in the dataset might not be a realistic representation of the physical system. However, if it is necessary to limit the size of the dataset, one would imagine that it is more important to have the extremes well represented. For further details about the simulations, see Appendix B.

To make sure that the diurnal cycle is covered, for every simulation, we sample from a 24-hour period. Since the model is run with 60-second time steps and a domain size of 200×150, this gives us potentially $200 \times 150 \times 24 \times 60 \approx 4 \cdot 10^7$ profiles, and with a vertical resolution of 41 full η levels, we have more than one and a half billion data points from each simulation. However, some data points are most likely highly correlated and might not contribute with new information. Therefore, only a small subsample from each simulation is used for the training dataset. Data is exported from WRF once each hour, and for each of these time steps, 500 randomly selected profiles are added to the training dataset. It is common practice to reserve somewhere between 10% and 50% dataset for validation, depending on the complexity of the problem and the amount of data available [28]. And as described in Section 3.2.1, it is a good idea to have a third independent *test* dataset for estimating the error on the final model.

Since in this study, the final test will be implementing the model in WRF, one could argue that this additional test dataset, might not be necessary. However, in Section 4.4.5 we shall see that it does provide some useful information. And since in this case, the problem is not lack of data, two additional datasets of same size as the training dataset are subsampled from the simulated data. These will serve as validation data and test data, respectively. The choice of 500 profiles may seem arbitrary, however, tests suggested that fewer data points lead to poor representation of some of the rarer cases, such as the statically unstable model levels, and more data points did not seem to contribute significantly to the quality of the final model. From each of the 6 simulations, the data was sampled as described above, giving a total of $6 \times 500 \times 24 \times 39 = 2808000$ data samples for both training, validation and test. 39 is the number of model levels, where the PBL scheme predicts the turbulent quantities. This is the number or full η levels minus two, because a different scheme is used for the lowest level, and the diffusivities at the top level are assumed to be zero. For an illustration, see Figure 4.2 in Section 4.3.

To avoid effects from the boundaries, no data is sampled from the first 100 km from the boundaries. Likewise, to avoid unphysical behavior due to the model *spinning up*, we run each simulation for 30 hours and start sampling after 6 hours. To shorten the spin-up time and minimize noise from unbalanced initialization, WRF's digital filter is applied in each of the

simulations as described in Section 2.3.

4.2 Determining model output

Since neural networks are essentially able to emulate any nonlinear function, there are several different options to consider, when constructing this new PBL scheme. In particular, one need to decide what the model should predict. An obvious option is the tendencies due to turbulence for all relevant physical variables. In this case, the PBL scheme would consist solely of the neural network, and there would be no need for combining it with other methods. Alternatively, the neural network could predict the turbulent fluxes for each physical variable, or the turbulent diffusivities, just like most traditional PBL schemes. In these two latter cases, the neural network would need to be combined with other methods for computing the tendencies.

We start by considering a model that predicts the tendencies directly. First, note that the tendencies are the result of a combination of turbulent transport and a conversion to/from energy associated with the mean field variables. This means that somehow, one needs to keep track of the energy to make sure general conservation laws are respected. This could be done by using turbulent kinetic energy as an additional prognostic variable, and then constraining the total loss of energy from the mean fields to correspond to the imbalance in the TKE budget. This is, although not impossible, not a trivial task. Since the tendencies are also results of transport, the local tendencies are not necessarily directly related to the local TKE budget terms.

Therefore, a model predicting the tendencies directly needs some non-local variables as input, i.e. from a number of neighboring model levels. One easy way to ensure that all the necessary information is available would be to make a model that takes the whole vertical column as input and predicts the whole column of tendencies. However, this will make the input and output parameter spaces very large and thus the problem will be much more complex than with a local approach. Hence, we believe a better option is a model that takes local atmospheric variables as input and predicts local turbulent fluxes or diffusivities.

Further, we believe there are several advantages predicting the diffusivities instead of the fluxes. First, only two diffusivities are needed, one for momentum flux, and one for the flux of all scalar quantities. Hence, all information about the local effects of turbulence is contained in those two variables, whereas if the model should predict the fluxes (or even tendencies), it would need to be predicted explicitly for all the relevant physical variables. Predicting the diffusivities also makes it easy to compare the scheme to existing PBL parameterization schemes, since these are all essentially based on some form of eddy diffusivity. It also enables us to keep the implicit solver for the diffusion equation. This is particularly interesting, since this Crank-Nicholson type solver is unconditionally stable [6], and thus it will reduce the risk of numerical instability,

when the parameterization is implemented in WRF. Actually, we deemed it highly risky, from a numerical stability point of view, to build a machine learning model, which predicted the turbulent fluxes or the tendencies directly. Finally, early experiments suggested that the diffusivities were easier for the neural network to learn.

In addition to the diffusivities K_h and K_m, a couple more outputs are needed, if TKE should be kept as prognostic variable. Recall from Section 1.3.5 that the dissipation term in the TKE equation is parameterized as

$$\varepsilon = \frac{q^3}{B_1 L},$$

where q is the square root of twice the TKE, just like it is defined in the MYNN scheme, $B_1 = 24$ is a closure constant, and L is the turbulent length scale. Similarly, the buoyant production term depends on the flux of virtual potential temperature, which is computed as

$$\overline{w\theta_v} = \beta_\theta \overline{w\theta_l} + \beta_q \overline{wq_w},$$

where the β_θ and β_q are variables computed by the partial-condensation scheme, which is part of the MYNN PBL scheme, θ_l is the liquid-water potential temperature, and q_w is total water content.

To solve the TKE equation, these terms need to be estimated somehow. For the latter, one could of course approximate the buoyant production term as $-\overline{w\theta_v} = K_m \frac{\partial \Theta_v}{\partial z}$. However, if we want the neural network to emulate the MYNN scheme, the correction to the buoyancy term should be taken into account. Thus, to be able to solve the TKE equation, our neural network also predicts the turbulent length scale L and the buoyant production term B_p defined as

$$B_p = -\frac{g}{\Theta_0}\overline{w\theta_v} = -\frac{g}{\Theta_0}\left(\beta_\theta \overline{w\theta_l} + \beta_q \overline{wq_w}\right).$$

4.3 Determining model input

Determining the correct input variables is of course crucial for getting the best model, so before we start tuning the hyperparameters, it is a good idea to determine which inputs the model needs. As described in Section 1, turbulence depend strongly on the wind shear and the static stability. Hence, the neural network as a minimum needs some information about the wind and temperature fields. In addition, since the MYNN scheme has TKE as a prognostic variable, this is of course an obvious choice as an additional variable. In the original MYNN scheme, the variables related to shear and stability are G_m and G_h as defined in Equation (1.39). The variable G_h, however, depend on the variables β_θ and β_q, so this requires the use of the partial-

condensation scheme. In addition, both G_m and G_h scale with L^2/q^2. Since L is an output from the neural network, we can of course not give it as an input. Therefore, we define two new variables B and S, which are inspired by G_m and G_h but depend only on known mean field variables. Together with q, these are inputs to what we define as the *base model*

$$q = \sqrt{\overline{uu} + \overline{vv} + \overline{ww}}, \qquad B = -\frac{g}{\Theta_0}\frac{\partial \Theta_v}{\partial z}, \qquad S = \left(\frac{\partial U}{\partial z}\right)^2 + \left(\frac{\partial V}{\partial z}\right)^2, \tag{4.1}$$

where q is the square root of twice of the TKE, while B and S are related to the buoyancy and wind shear, respectively. We then define the base model

$$K_h, K_m, L, B_p = f_b\left(q, B, S\right), \tag{4.2}$$

where f_b is a not yet further defined neural network, that takes q, B and S as inputs and predicts K_h, K_m, L and B_p. All variables are defined in the same horizontal grid point and same model level.

Several other variables are considered as possible inputs: Surface sensible heat flux Q_0, surface friction velocity u_*, height above the surface z, potential temperature Θ, temperature T, water vapor content Q_v and cloud water content Q_c. Except for the surface fluxes, all these input variables are also defined in the same model levels as the outputs. The surface fluxes and the height above the surface are important for the turbulent length scale close to the surface, as discussed in Section 1.3.5. The reason for considering Θ, T, Q_v and Q_c, as inputs is that the buoyant production term B_p is affected by condensation processes. The Clausius-Clapeyron relation implies that the saturation vapor pressure e_s is dependent on the temperature T, while the actual water vapor pressure e is linked to Q_v via the equation of state [4, Ch. 3]. Together these two variables indicate how close the air parcel is to saturation. The presence of liquid water Q_c is of course also relevant due to the possibility evaporation. It is less intuitive, why the potential temperature should be relevant. However, as mentioned earlier, the MYNN scheme uses a partial-condensation scheme [14] to compute β_θ and β_q, which determines the flux of virtual potential temperature. In [14], Deardorff and Sommeria derives the expressions for β_θ and β_q using the variables Θ_l and Q_w, and they end up with expressions dependent on both these variables. Hence, referring to the definition of Θ_l in Equation (1.19), we see that the buoyancy production term, as it is defined in the MYNN model, must depend on both Θ, T, Q_v and Q_c. Table 4.1 below, shows an overview of the variables tested.

Table 4.1: The variables for the base model as well as the 7 additional variables tested on top of the base model.

	Variable	Explanation	Unit
Base model variables	q	*Square root of twice 2TKE*, Eq. (4.1)	$[m/s]$
	B	*Buoyancy parameter*, Eq. (4.1)	$[s^{-2}]$
	S	*Shear parameter*, Eq. (4.1)	$[s^{-2}]$
Additional variables	z	*Height above surface*	$[m]$
	Q_0	*Surface sensible heat flux*	$[W/m^2]$
	u_*	*Friction velocity*	$[m/s]$
	Θ	*Potential temperature*	$[K]$
	T	*Temperature*	$[K]$
	Q_v	*Water vapor mixing ratio*	[kg/kg]
	Q_c	*Liquid water mixing ratio*	[kg/kg]

There could potentially be other relevant input variables. For example, it could be interesting to test the effect of surface latent heat flux, since as described in Chapter 1, the flux of virtual potential temperature is the important quantity for the overall static stability of the PBL. The turbulent flux of virtual potential temperature at the surface can be estimated as

$$\left(\overline{w\theta_v}\right)_0 \approx \frac{1}{\rho c_p} Q_0 + \frac{0.61\Theta}{\rho L_{water}} Q_{LH} \approx \frac{1}{\rho c_p}\left(Q_0 + 0.07 Q_{LH}\right),$$

where Q_0 and Q_{LH} denote the surface fluxes of sensible and latent heat, respectively, and L_{water} is the specific latent heat for condensation. Thus, even when the latent heat flux is large, its contribution to the flux of virtual potential temperature would probably be insignificant in most cases. Further, one could add information from the neighboring model levels. This, however, will make it necessary to treat the boundaries to the surface and at the top level differently and was therefore not considered in this study.

Recall from Section 2.2 that all physical mean field variables in WRF, except vertical velocity, are defined in the mass midpoints of the grid boxes, i.e. the half η levels. However, the turbulent diffusivities K_h and K_m as well as the turbulent length scale L and the buoyant production B_p should be computed in the full η levels, i.e. the interfaces between grid boxes. The lowest full η level is the interface between the ground and the lowest grid box and the turbulent fluxes there are computed by an independent surface flux scheme, while the turbulent fluxes at all other η levels are computed by the PBL scheme. An illustration of this is shown in Figure 4.2

The importance of the different variables was tested by adding each of them as input to the base model, training each model until the value of the loss function converged, and then comparing the loss values to determine which variable improved the result the most. That variable is then added to the inputs, and this model becomes the new benchmark, when adding

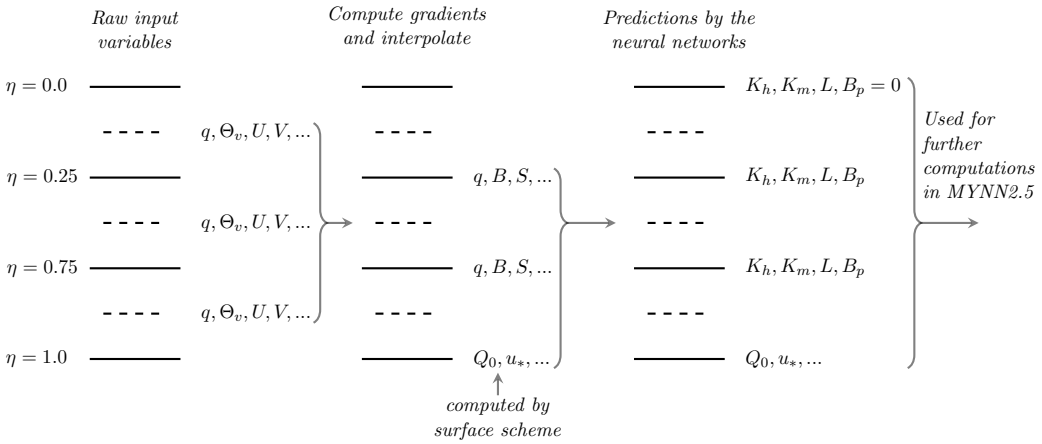


Figure 4.2: Illustration how the neural network scheme will interact with the WRF model. The sketch shows a WRF domain with 4 η levels (the solid lines) and three half η levels (the dashed lines). The half η levels are the mass midpoints of the grid boxes, while the full η levels are the interfaces between grid boxes. The three different columns show at which levels, the different variables are defined, and the arrows indicate, which variables are used to compute the next variables in the next column. Above each column, the title indicates, how the variables are obtained.

the next variable. The process is then repeated, such that all the remaining variables are added one at the time, and the best model is chosen. After each iteration, the improvement is thus expected to be smaller and after some time, we might even see that adding more variables does not have any effect. Since the relevant input variables might not be the same in different physical scenarios, we do this for two separate cases, namely statically stable and unstable model levels. The stability criterion used is simply the sign of the buoyancy parameter B, which is positive for unstable conditions and negative for stable conditions.

Large amounts of data can cause the training to be slow and thus the convergence time to be long, so this can be a very inefficient approach. Therefore, we train only on a subset consisting of 100000 randomly selected samples from the training dataset. Similarly, 100000 randomly selected samples are used for validation, so that we only compare the model performances on data that were not part of the training dataset.

Further, we need to design a model appropriate for this problem, and since we have not yet discussed the design of the neural network, this may seem arbitrary. Some of the choices of hyperparameters, such as optimizer and activation are based on what is generally accepted to work well. The remaining hyperparameters are chosen based on initial trial and error. Only the learning rate is manually tuned to avoid inefficient learning. The method for this described in Section 4.4.1 [2]. The hyperparameters for this setup are shown in Table 4.2, however, explanations for some of them are not presented until later in this chapter.

[2]Here, only the linear learning rate scanner is used and not the learning rate schedule, see Section 4.4.1.

Table 4.2: The model setup for determining input variables.

Optimizer	Activation	Pre- and postprocessing	Loss function	Batch size	Model size
Adam	ReLU	Logarithmic scaling	Mean squared error	1024	2 layers with 50 nodes in each.

The same model architecture and hyperparameters are used for the stable and unstable case. Each model is trained for 200 epochs, since this seemed to be sufficient to see, which model converges to the lower value of the loss function.

4.3.1 Results

Figure 4.3 shows the loss on the validation data during the training of 200 epochs for eight different models. As described above, a total of 7 variables are tested on top of the base model, and the process is done iteratively, so that each time a new variable is added, all the candidates are tested, and only the one with the lowest loss in the validation set is accepted. In Figure 4.3, only the models with the new accepted variables are shown. Note that the y-axis is logarithmic to make it easy to see the different convergence values.

We see that especially adding liquid water Q_c and height above the ground z seems to reduce the errors significantly. Also the remaining variables do seem to have some impact on the model performance, except the temperature, i.e. the two bottom curves (pink and gray) converge to roughly the same value.

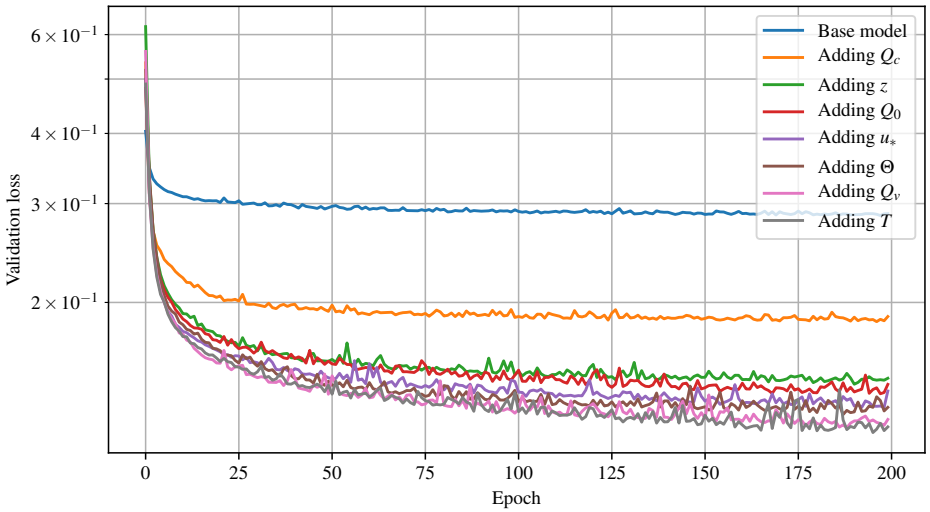


Figure 4.3: Learning curves showing validation loss for models for stable samples with different input variables. The base model has q, B and S as inputs, and the remaining models each add a new variable in addition to the input variables of the previous model (the model *above* in the legend). Thus, the last learning curve correspond to the model using all input variables.

Figure 4.4 is similar to Figure 4.3 but for the unstable samples. It is interesting to note that the order of which variables has the largest impact is not the same as for the stable case. However, the conclusion is the same: all variables do seem to improve the result, except the temperature.

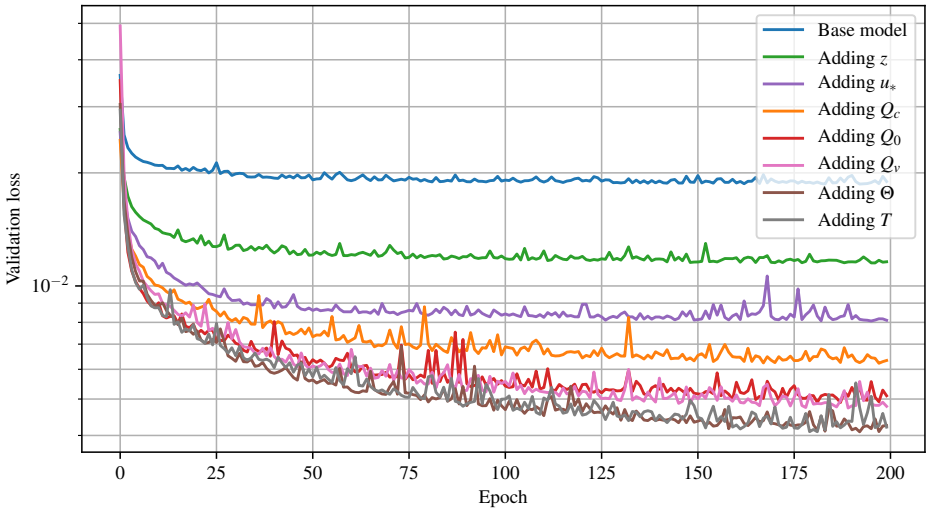


Figure 4.4: Similar to Figure 4.3 but for unstable samples. The color of each learning curve is selected such that it matches the learning curve for the same variable in Figure 4.3.

4.3.2 How to conclude based on the results

First, based on the two figures, we conclude that there is no reason to differentiate between stable and unstable model levels when it comes to the choice of input variables. Further, the analysis show that all the tested input variables, except for temperature, do seem to contribute with valuable information. Therefore, a natural first approach was to include all these variables as inputs for the neural networks.

However, the predictions of a neural network are generally only useful, when it is exposed to examples, which are within the variability of the training dataset. As described in Section 4.1, the dataset is based on only six simulations, and is constructed to have the extremes well represented rather than covering all types of weather. The hope was of course that the neural networks would be able to interpolate between the extremes, but it cannot be ruled out that the network learns to falsely correlate certain boundary layer dynamics with the specific weather scenarios in the training data. If this is the case, then the model might behave in an unexpected way when encountering different weather situations. Especially the potential temperature and the mixing ratios for water vapor and liquid water depend on the specific weather situation. Further, we mainly expect these variables to contribute to a correction to the buoyancy production term, which should only be relevant in situations when condensation or evaporation occur. Hence,

these variables should mainly contain information, which is irrelevant for the majority of the samples. Therefore, it is not unlikely that these variables lead to overfitting.

As described in Section 3.2.1, monitoring of the learning curves is a good way to make sure that a model does not overfit to features in the training data, which are not present in the validation data. Here, however, the risk is a different type of overfitting, where the model can overfit to features present in both the training data and validation data. This is an indication that the two datasets are not independent, which is of course true since both datasets are sampled from the same simulations. This will be discussed further in Chapter 6.

As mentioned, the first approach was to include all variables except temperature just as the results above suggested. The following Sections 4.4.1-4.4.5 describe how this model was optimized. However, when the model was implemented in WRF, the results were far from satisfying, and this lead to the hypothesis that the neural networks had overfitted. Therefore, as an attempt to improve the results of the model, a new model was constructed without the input variables, which were believed to cause the overfitting. This model, described in detail in Section 4.4.6, only takes q, B, S, z, u_* and Q_0 as inputs. Several of the results obtained in the sections 4.4.1-4.4.5 are assumed to be valid for this model, but some adjustments are made.

4.4 Training and optimizing the model

Hyperparameter optimization is a complex task, since, generally, the hyperparameters are interdependent. However, varying too many hyperparameters at once, can make the process both inefficient and intransparent, while varying too few, one might not find the optimal combination. As described in Chapter 3, common hyperparameters are learning rate, batch size, optimization algorithm, activation function, loss function, as well as the size and architecture of the neural network. Some of these hyperparameters are strongly interdependent, such as learning rate, batch size, optimization algorithm and loss function. Therefore, the task of finding the best hyperparameters involves finding a minimum in a high dimensional parameter space.

Generally, there are two approaches to this optimization process: grid search and Bayesian optimization. The latter covers a range of different optimization algorithms, which estimate a functional relation between the values of the hyperparameters and model performance. This so-called object function is unknown and initially assumed uniform but is then gradually altered as different models are tested [31].

To compare two models with different hyperparameters, in principal, both models need to be trained until their losses have converged to a minimum value. Therefore, no matter which method is used, this will be a time-consuming process. To ease this process, the model optimization was separated into three steps, which will be assumed mutually independent

- Finding the optimal data processing, data categorization and loss function.
- Examining the relation between model performance and network size.
- Optimizing activation and batch size.

In addition, the learning rate is manually tuned to each combination of hyperparameters as described in Section 4.4.1. For all three steps, grid search is used to find the optimal hyperparameter combination. While using an intelligent algorithm to find the best model seem tempting, one problem with the Bayesian optimization algorithms is that they often require many iterations to find the best combination of hyperparameters. Since this may require many hours of training on a standard laptop, it was not feasible in this project. In addition, sometimes we are not only interested in the best model, but rather the relation between model performance and efficiency. In this case, grid search in the specific parameters related to model size is more interesting.

Section 4.4.1 describes a quick method for estimating the optimal learning rate for any given combination of hyperparameters. The Sections 4.4.2 and 4.4.3 explains what is meant by data categorization and data processing. In Section 4.4.4, the remaining hyperparameters are described, and in Section 4.4.5 the results of the optimization are shown. The model optimization is repeated for the model using fewer input variables in Section 4.4.6, and finally in Section 4.4.7, the best models from each optimization process are compared.

4.4.1 Learning rate and optimizer

It is generally accepted that the learning rate is one of the most important hyperparameters, since a too small learning rate means that the model risks to *get stuck* in local minima, and with a too large learning rate the model will never learn anything. Using an optimization algorithm with an adaptive learning rate such as Adam should, in theory, help, but even Adam does depend on the start value of the learning rate as discussed in Section 3.2.1.

As suggested by Smith [30], a linear learning rate *scanner* is used to estimate the optimal learning rate for each model run. With this method, only a few epochs are needed to estimate the optimal learning rate, which means that the learning rate can be optimized without training the model to convergence. This is very useful, since it allows to quickly determine the optimal learning rate for any combination of hyperparameters to ensure efficient training. The idea is to linearly increase the learning rate after the training on each mini-batch and then obtain a relation between learning rate and loss. The optimal learning rate is then assumed to be located, where the slope of the learning curve is steepest.

However, as discussed in Section 3.2.1, there is not necessarily one optimal learning rate and one might consider using a learning rate schedule to make the training more efficient. To

demonstrate the effect of the learning rate schedule, two different models are trained. The first model is trained with Adam using the optimal learning rate found using the learning rate scanner, and the second is trained with Adam using a triangular cyclic learning rate schedule.

The models are trained on the entire training dataset, both stable and unstable samples. Aside from the learning rate, the remaining hyperparameters are kept constant, and they are the same as those described for *model 1* in Section 4.4.5.

The optimal learning rate is found by using the linear learning rate scanner for one epoch, varying the learning rate from 10^{-6} to 10^{-2}. Figure 4.5 shows the loss as function of the learning rate, and here we clearly see that there is a range of learning rates, $\sim 4 \cdot 10^{-5}$ to $\sim 2 \cdot 10^{-4}$, for which the loss decreases significantly. However, using learning rates far outside of this interval seem to be inefficient. Thus, the optimal learning rate should be located somewhere in that interval. A simple algorithm is used to estimate lr_{opt}, by finding the learning rate corresponding to the mean value of the highest and lowest loss. Using this mid-point might not find the steepest part of the slope, but it ensures that the learning rate chosen is well inside the steep part of the learning curve and has turned out to be the most robust way of estimating lr_{opt}.

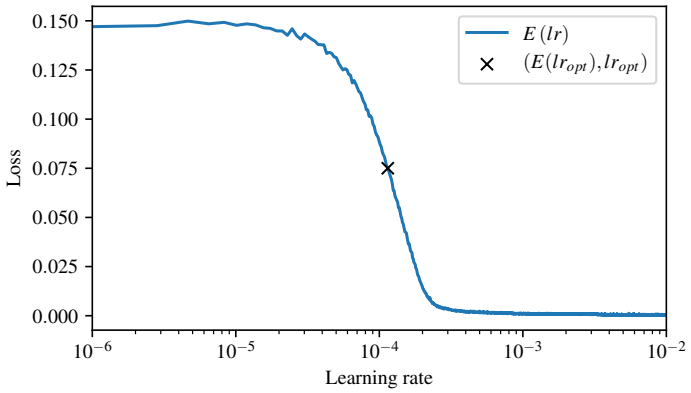


Figure 4.5: Loss as function of learning rate. The relation is obtained using a linear learning rate scanner and is used to estimate the optimal learning rate.

The cyclic learning rate schedule makes the learning rate vary between lr_{base} and $\gamma^i lr_{max}$, where i is the iteration number, i.e. the training of the i'th mini-batch, and hence γ^i is an exponentially decaying factor. lr_{base}, lr_{max} and γ are constants that need to be specified. In addition, one must specify a *stepsize* δ, which is the number of iterations it takes to increase from lr_{base} to lr_{max}. The stepsize is recommended to be somewhere between 2 and 10 epochs [30], and for this problem, 5 epochs seemed appropriate. The maximum value is suggested, as a rule of thumb, not to be more than a factor of two away from the "optimal" learning rate, or alternatively, one can estimate lr_{base} and lr_{max} from Figure 4.5 as the learning rates, where the efficient learning

seem to start and stop, for example $lr_{base} = 4 \cdot 10^{-5}$ and $lr_{max} = 2 \cdot 10^{-4}$. However, in our case, a much higher maximum value makes the model converge even faster. The constants are thus set to $lr_{base} = 1/2 lr_{opt}$ and $lr_{max} = 10\gamma^i lr_{opt}$, respectively. From Figure 4.6, we see that the learning rate schedule, makes the loss value oscillate quite heavily, but the low values are significantly lower than when using Adam with lr_{opt}. γ is set based on visually estimating when the learning curve to has flattened *significantly*. After about 500 epochs, corresponding to 50 cycles, the slope of the learning curve is almost flat, and γ is set so that, at this point, the amplitude will be 10% of its initial amplitude. Since the stepsize δ is the number of iterations corresponding to half a cycle, it follows that

$$\gamma^{2 \cdot 50\delta} 10 lr_{max} = lr_{max} \iff \gamma = 10^{\frac{-1}{100\delta}},$$

Using a batch size of 1024 and having 2808000 data points, we get $\delta = 5 \cdot 2808000/1024 \approx 13711$ iterations, which, using the above relation, gives $\gamma \approx 0.999998$.

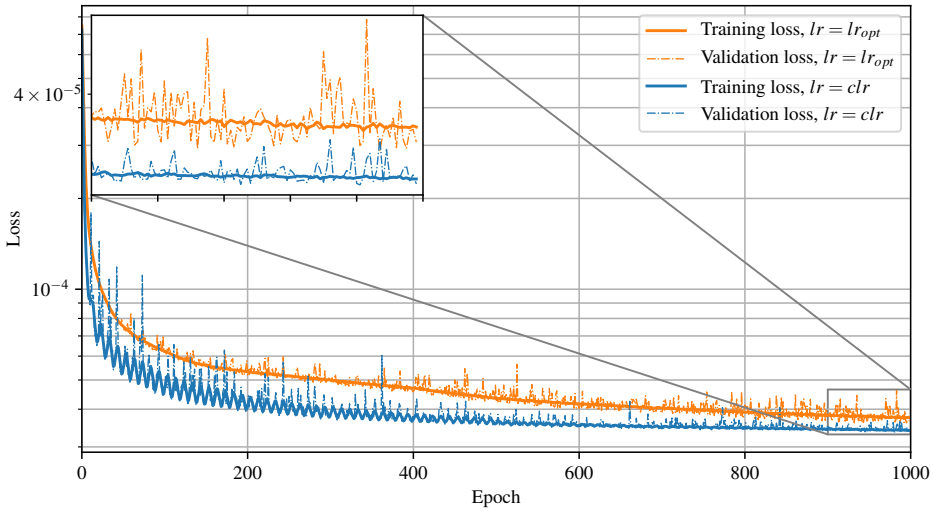


Figure 4.6: Learning curves for two identical models trained using different learning rate strategies. The orange curve show the loss of a model trained with Adam using the learning rate lr_{opt}, whereas the blue curve show the loss of a model trained with Adam using a cyclic learning rate schedule, *clr*.

From Figure 4.6, we see that after the 1000 epochs, the model trained with Adam using lr_{opt} is still learning, whereas the model trained with Adam using the cyclic learning rate seems to have converged. Thus, this method seems to speed up the training significantly. All models described in the following sections have been trained using Adam with the cyclic learning rate schedule, *clr*, where the lr_{base} and lr_{max} are found as described above.

4.4.2 Categorizing the data

The main idea is to categorize the samples in the training dataset based on some physical condition, and then use a different model for each category. If one of these categories is clearly underrepresented in the dataset, a model trained on all samples may have difficulties learning how to "handle" the underrepresented samples. In addition, if the data can be separated in a way that makes the problem mathematically less complicated, it will require a less complex neural network to learn the problem. Thus, this could benefit both accuracy and efficiency.

As described in Section 1, the physical behavior of an unstable atmosphere is fundamentally different from that of a stable atmosphere, due to the changing role of buoyancy forces. In addition, out of the $\sim 2.8 \cdot 10^6$ samples in the training dataset, only $\sim 3.4 \cdot 10^5$, about 12%, are unstable. Therefore, it seems natural to categorize the data depending on the sign of the buoyancy parameter B.

Further, only $\sim 7.3 \cdot 10^5$ out of the $\sim 2.5 \cdot 10^6$ stable samples, about 29%, has a TKE value above the minimum value. This is of course due to the fact that most of the free atmosphere is stable and non-turbulent, however, one can question the importance of the two thirds of the dataset, where no turbulence is present. Therefore, we suggest to further separate the stable samples into two categories depending on the presence of turbulence. For the samples with TKE, a neural network will be trained to predict the turbulent quantities, and for those without, the turbulent quantities will be set to some minimum values, except the buoyancy production term that will be parameterized using the vertical gradient of the virtual potential temperature, as described in Section 4.2.

4.4.3 Pre- and postprocessing

Preprocessing the data is generally necessary to ensure efficient training of neural networks. If the values of the different input variables differ by orders of magnitude, the gradient of the loss function becomes highly non-uniform, which can slow down or even completely stop the training [28]. Therefore, it is generally advised that all input variables are scaled to be mean-centered, i.e. the mean value is subtracted from all data points. Further, if the distributions of the variables are approximately normally distributed, then all data points should be divided by the standard deviation of the variable, and otherwise it should be divided by the maximum absolute value, such that all data points are in the interval $[-1, 1]$ [28]. In addition, one can imagine that if the values of the output variables differ by orders of magnitude, the loss functions in Equations (3.5) and (3.6) will "focus" mainly on the variables with large values. For that reason, it is a good idea also to scale the outputs to have values in similar ranges. Thus, a typical data

normalization would be

$$x_{scaled,i} = \frac{(x_i - x_{mean})}{\max(|x|)} \tag{4.3}$$

$$y_{scaled,i} = \frac{y_i}{\max(|y|)} \tag{4.4}$$

In the Equations (4.3) and (4.4), x_i and y_i denote one of the input and output variables, respectively, while the index i runs over all the samples in the training dataset. The scaling is performed independently for each of the input and output variables. Note that the mean value is not subtracted before scaling the outputs. This is because the scaling of the outputs is merely to ensure that all variables are of the same order of magnitude, such that they are equally weighted when the loss is computed. In the Figures 4.7 and 4.8, respectively, the scaled input and output variables described in the sections 4.2 and 4.3 are shown.

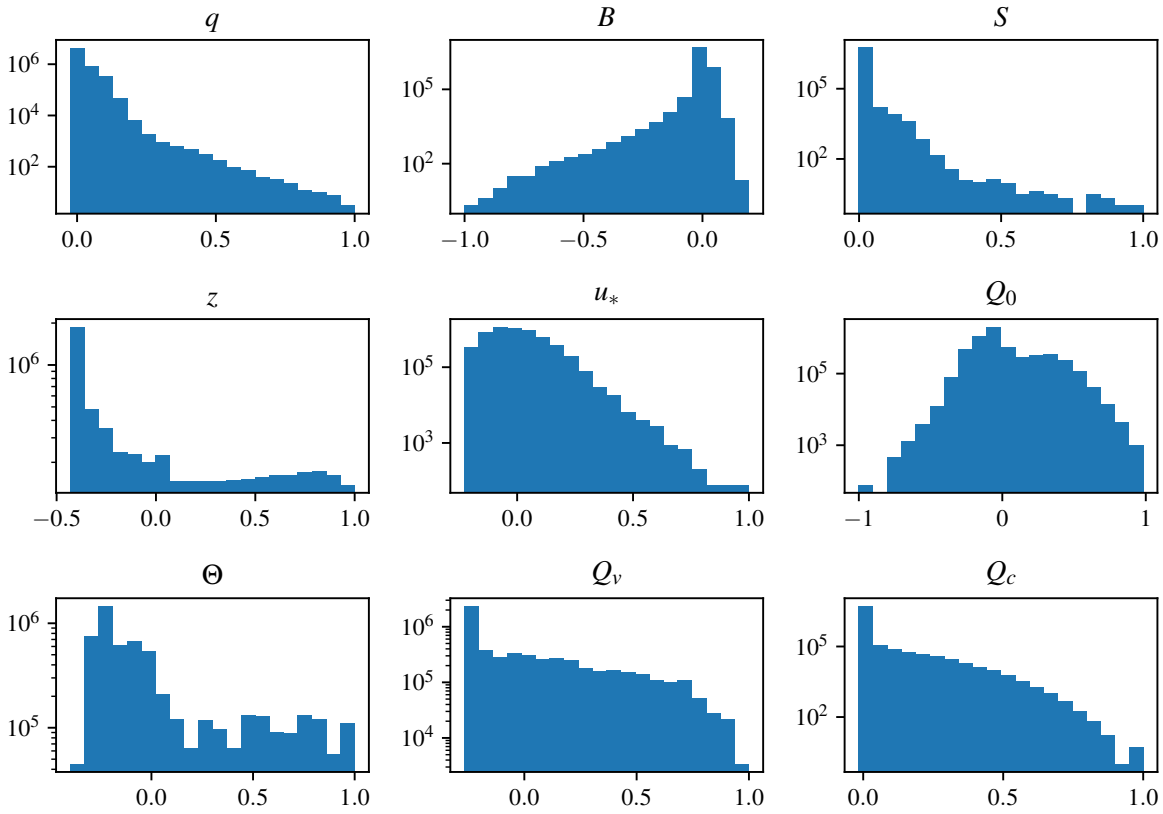


Figure 4.7: Distributions of all the input variables, after the linear scaling from Equation (4.3) is applied. The histograms are based on all, both the stable and unstable, samples from the training dataset.

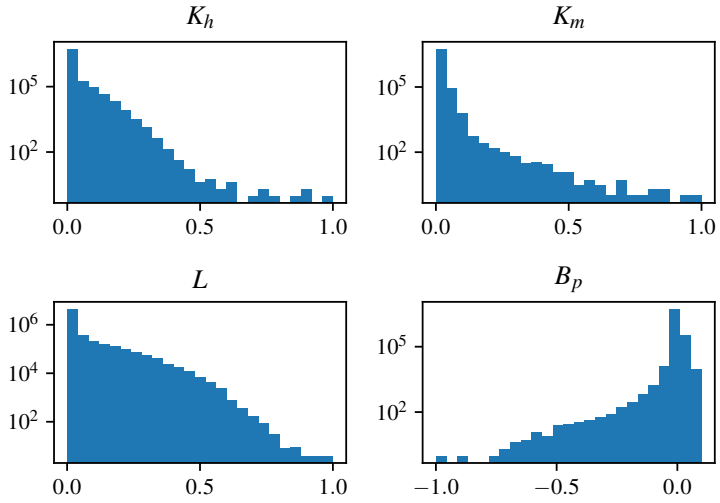


Figure 4.8: Distributions of all the output variables, after the linear scaling from Equation (4.4) is applied. The histograms are based on all, both the stable and unstable, samples from the training dataset.

From the Figures 4.7 and 4.8, we see that for several of both the input and output variables, the majority of the data points are located in a relatively narrow interval, while a small fraction of the data points is spread out in a long *tail*. Note the logarithmic y-axes on both figures. Thus, referring to the discussion above, this large difference in orders of magnitude may cause the learning to be inefficient. One solution is to use a nonlinear scaling such as a logarithmic function. The logarithm does have the properties we are looking for; however, it only allows scaling of positive values. Hence, for variables with both negative and positive values such as Q_0, B and B_p, the scaling cannot be applied directly. The Q_0 distribution, however, does not have nearly as long tails as the other variables, so for Q_0 we will simply omit the logarithmic scaling. Similarly, we will not use the logarithmic scaling on the Θ. Although, all temperature values are positive, the temperature values do not cover several orders of magnitude, so there would be no point in using a nonlinear scaling.

In Section 4.4.2, we discussed separating the dataset in stable/unstable samples depending on the sign of the buoyancy parameter B. In this case, the logarithmic scaling can be applied directly for B by using the absolute value. One would imagine that the output variable B_p, i.e. the buoyant production term in the TKE equation, had the same sign as the buoyancy parameter B, but this is not the case for all samples. In Figure 4.9, the output variable distributions are shown for the two different cases. Although these cases with the "wrong" sign are rare, they prevent the use of logarithmic scaling. For the stable samples, $\sim 1.7 \cdot 10^4$ out of the $\sim 2.5 \cdot 10^6$, or $\sim 0.7\%$ of the samples has B_p values with the "wrong" sign, while for the unstable case it is only 91 samples out of the $\sim 3.4 \cdot 10^5$, or $\sim 0.03\%$. Since the fraction of these samples is

so small, and because the numerical values of these production terms are also relatively small, these values are simply to set to a "minimum" value with the "correct sign", such that the logarithmic scaling can be applied.

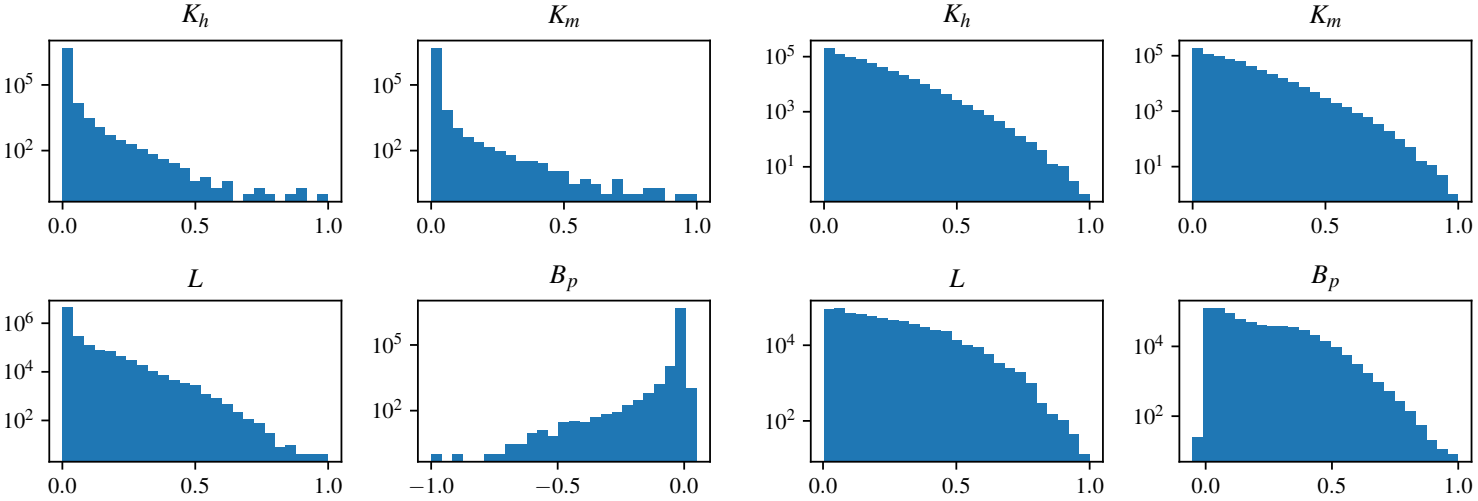


Figure 4.9: Distributions of output variables for all stable samples (left) and unstable samples (right), after the linear scaling from Equation (4.4) is applied. Note that each of the datasets are divided by its own maximum absolute value. The histograms are based on all samples from the training dataset.

To demonstrate the effect of the logarithmic scaling, the distributions for the unstable samples are shown in the Figures 4.10 and 4.11, while the distributions for the stable samples are shown in Appendix C.

After the logarithmic scaling of the inputs, the scaling from (4.3) is applied, while for the outputs the logarithmic scaling is applied after (4.4) is applied. Now, one might notice that several of the variable distributions still has relatively long tails. However, before the tails represented extreme values, where turbulence is typically important, whereas now the tails represent physical values very close to zero. Thus, these "outliers" are cases where the exact physical value is not really of importance.

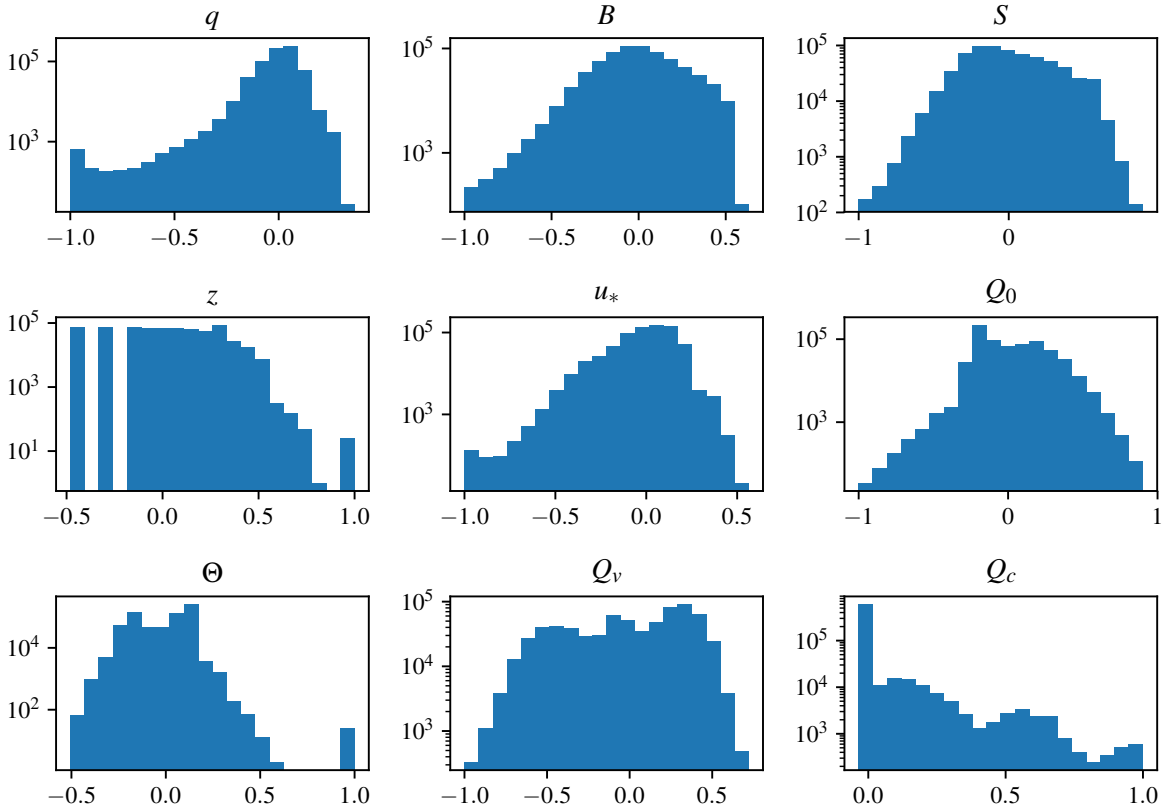


Figure 4.10: Distributions of all the input variables. First the logarithmic scaling is applied and then the linear scaling from Equation (4.3). The histograms are based on all unstable samples from the training dataset. Similar histograms for the stable samples are shown in Appendix C.

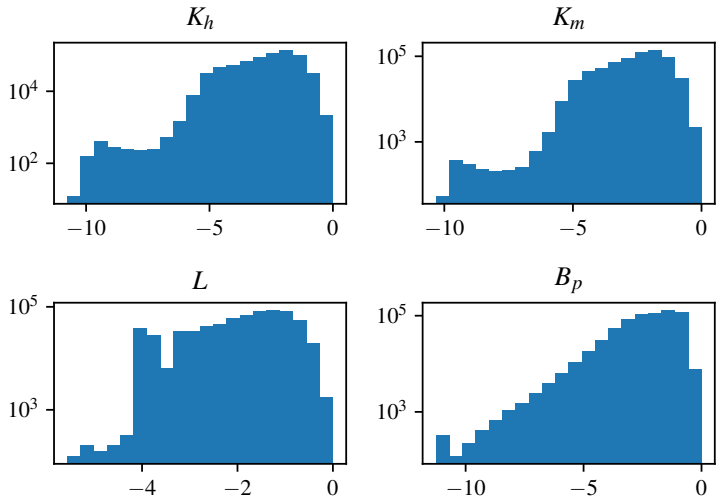


Figure 4.11: Distributions of all the output variables. First the linear scaling from Equation (4.4) is applied, and then the logarithmic scaling. The histograms are based on all unstable samples from the training dataset. Similar histograms for the stable samples are shown in Appendix C.

4.4.4 Other hyperparameters

Once the optimal data processing is found, we are ready to optimize the remaining hyperparameters.

Loss function

The first thing tested is the loss function, since we want to make sure that the optimal metric is used to compare the models in the further optimization process. Generally, *mean squared error* or *mean absolute error*, Equations (3.5) and (3.6), work well for regression problems. But when we use a nonlinear scaling of the outputs, we have a new choice to make: should we minimize the error on the scaled values or should we compute the physical values first and minimize the error on those?

Network size

The optimization of the network size might involve a trade-off between accuracy and computational cost, since the smaller the network gets, the less it will be able to learn, while on the other hand a very large network might be computationally heavy. Larger network, however, are also more likely to overfit.

We want to examine this relation between model performance and network size and maybe find an "upper limit", where adding more model parameters either leads to overfitting or just does not improve the model performance. 9 different models are trained: with 1, 2 and 3 hidden layers, and with 25, 50 and 100 nodes in each layer.

Batch size

The batch size is the number of samples that are used to compute the gradient of the loss function. Thus, the smaller the batch size, the more noise is introduced in each iteration as consequence of incorrect estimation of the gradient. The batch size also changes the number of iterations per epoch, and therefore different batch sizes most likely result in different numbers of epochs before convergence. In this study, batch size did not seem to play an important role for the model performance, which is why the first part of the optimization was done with a fixed batch size. However, to make sure that as many options as possible are tested, we here compare 4 models trained with different batch sizes.

Activation

The activation function is, as described in Section 3, what enables the network to learn nonlinearities. Therefore, one would imagine that the choice of activation function is important for

the model performance. To test this, we train four models with different activation functions. The different activation functions tested are ReLU, leakyReLU, sigmoid and tanh.

It should be noted that all activations are tested in a model setup, that has been found in the optimization process described above, using the activation function ReLU. Hence, it cannot be excluded that a model with another activation function would perform better in a network of different size, and we can therefore only conclude which of the activation functions that work best in combination with this specific network architecture.

4.4.5 Model optimization

As described, the model optimization will be done in three steps. However, for all combinations of hyperparameters, the learning rate will be tuned to the specific model, before the training begins, and the cyclic learning rate schedule will be used to ensure efficient training.

Step 1: Finding the optimal data processing, data categorization and loss function.

To test the effects of the data categorization, the different pre- and postprocessing approaches and the impact of the loss function, six different models are trained. In Table 4.3, the model setups are listed. The models are constructed such that model 1 is based on the simplest, most "naive" approach. For model 2, 3, etc., different features are then gradually added.

Table 4.3: First column is the model number. Second column tells us, how the data is categorized: *None* means one model is used for all samples. *Stability* means that two different models are used for stable/unstable samples. *Stability and TKE* means stable samples are further divided into samples with/without TKE. Third column shows the data processing, and the fourth column shows the loss function used. *mse* is mean squared error, while *mae* is mean absolute error.

	Data categorization	**Pre- and postprocessing**	**Loss function**
Model 1	None	Linear scaling	*mse* on physical values
Model 2	Stability	Linear scaling	*mse* on physical values
Model 3	Stability	Logarithmic scaling	*mse* on log-scaled values
Model 4	Stability and TKE	Logarithmic scaling	*mse* on log-scaled values
Model 5	Stability and TKE	Logarithmic scaling	*mse* on physical values
Model 6	Stability and TKE	Logarithmic scaling	*mae* on physical values

For the remaining hyperparameters, i.e. network size, batch size and activation, we use what initial trial and error suggested works well for this specific problem. This is a batch size of 1024 samples, a network with 2 layers with 50 nodes in each, and the ReLU activation function. The number of epochs was estimated visually from the slope of the learning curves. All models in

both step 1, 2 and 3 are trained for 1500 epochs. However, the "best" model is not considered the final model but instead the model out of the 1500, which has the lowest validation loss.

In Section 4.3, we compared the convergence values of the validation loss to determine, which model had the lowest error on the prediction. Now, the actual loss values cannot be compared, since the models use different scaling of the output variables and are optimized using different loss functions. Therefore, to compare the performance of the models, all outputs are rescaled to the physical values, and a set of statistical error measures is computed

$$
\begin{aligned}
l_1 &= \frac{\sum_{m=1}^{M} \left| \hat{\psi}_m - \psi_m \right|}{\sum_{m=1}^{M} |\psi_m|}, \\
l_2 &= \sqrt{\frac{\sum_{m=1}^{M} \left(\hat{\psi}_m - \psi_m \right)^2}{\sum_{m=1}^{M} \psi_m^2}}, \\
r &= \frac{\sum_{m=1}^{M} \left(\hat{\psi}_m - \overline{\hat{\psi}} \right) \left(\psi_m - \overline{\psi} \right)}{\sqrt{\sum_{m=1}^{M} \left(\hat{\psi}_m - \overline{\hat{\psi}} \right)^2} \sqrt{\sum_{m=1}^{M} \left(\psi_m - \overline{\psi} \right)^2}}.
\end{aligned} \tag{4.5}
$$

Note that the sum is not over the input and output vectors $\mathbf{y}_m$ and $\hat{\mathbf{y}}_m$ as in Equation (3.5) and (3.6). Instead, Equation (4.5) computes the statistics of each individual output variable, denoted ψ. The hat still denotes the predicted value, and the overbar means the average value of the variable. The l_1 and l_2 measures are essentially the normalized mean absolute error and root mean square error, $rmse$, which allows for easier comparison between the different variables. r is the Pearson correlation coefficient.

Since the physical quantities of interest are the fluxes rather than the diffusivities, the heat and momentum fluxes are estimated by multiplying the diffusivities with the gradients. Instead of computing the turbulent fluxes for each of the wind components, the magnitude of the kinematic momentum flux is considered $K_m\sqrt{S} = K_m\sqrt{(\partial U/\partial z)^2 + (\partial V/\partial z)^2}$. And instead of the actual heat flux, we consider the flux of virtual potential temperature by computing $K_h B \propto \overline{w\theta_v}$. In addition, the diffusion term from the TKE Equation, $\varepsilon \propto q^3/L$ is computed. In Table 4.4, the evaluation of the models from Table 4.3 is shown. The independent test dataset is used for the comparison.

In Table 4.4, we see that each new "feature" added from model 1 to model 4 improves the performance of the model, and by far the largest improvement is applying the logarithmic scaling (from model 2 to 3). It is very interesting to note the difference between the performance of model 3 and model 4. Whereas model 3 uses one neural network for all the stable samples, model 4 only uses a neural network for the stable samples with non-zero TKE, while for the remaining samples K_h, K_m, and L are set equal to some minimum values. These minimum

values are set to the mean values of the variables in the part of the data set with no TKE. The buoyant production term is parameterized as $B_p = K_h B$. The two models use the exact same neural network for the unstable samples. To understand why the result is better when not using a model for the zero-TKE samples, we should examine the performances of model 3 and model 4 on the stable samples with/without TKE. In Table 4.5, the *rmse* for model 3 and 4 on these two datasets are shown.

Table 4.4: Performance of the models tested in the first step of the model optimization. The model specifications are explained in Table 4.3. For each variable and each error measure, the best performance is highlighted with red color, and the worst is highlighted with blue.

	Stat	K_h	K_m	L	B_p	$-K_h B$	$K_m\sqrt{S}$	q^3/L
Model 1	l_1	0.2713	0.2861	0.1261	0.2835	0.4481	0.9068	0.2849
	l_2	0.2986	0.3154	0.1598	0.2303	1.011	5.289	4.877
	r	0.9508	0.9438	0.9841	0.9733	0.6927	0.1277	0.3667
Model 2	l_1	0.2275	0.3206	0.1120	0.2261	0.5495	1.218	0.2263
	l_2	0.2250	0.2762	0.1476	0.1915	0.6040	8.798	0.7186
	r	0.9727	0.9580	0.9864	0.9820	0.8598	0.2382	0.8736
Model 3	l_1	0.06222	0.04390	0.1168	0.05586	0.07348	0.02881	0.07455
	l_2	0.1188	0.1136	0.1843	0.1082	0.1712	0.06836	0.1198
	r	0.9924	0.9930	0.9793	0.9943	0.9853	0.9978	0.9927
Model 4	l_1	0.05847	0.04058	0.1124	0.04618	0.06845	0.02351	0.05988
	l_2	0.1096	0.08420	0.1555	0.06163	0.1595	0.03453	0.1123
	r	0.9935	0.9962	0.9850	0.9981	0.9872	0.9994	0.9939
Model 5	l_1	0.08943	0.07602	0.1212	0.09739	0.1960	0.4034	0.1119
	l_2	0.1191	0.09794	0.1545	0.1364	0.2993	0.7136	0.1742
	r	0.9924	0.9948	0.9851	0.9907	0.9627	0.9837	0.9844
Model 6	l_1	0.04779	0.03089	0.09572	0.03554	0.05976	0.01854	0.05179
	l_2	0.1053	0.06758	0.1360	0.05390	0.1589	0.01868	0.09625
	r	0.9941	0.9975	0.9887	0.9985	0.9873	0.9999	0.9954

Table 4.5: Performance of model 3 and model 4 on two different subsets of the data: stable samples without TKE, and stable samples with TKE. Instead of the statistical measures from Equation (4.5), the *rmse* is compared here, since the order of magnitude of the error is very different on the two datasets.

	K_h	K_m	L	B_p	$-K_h B$	$K_m\sqrt{S}$	q^3/L
Model 3, **TKE** ~ 0	7.09E-4	9.07E-4	8.11E-2	3.47E-8	4.44E-8	1.74E-8	1.95E-6
Model 4, **TKE** ~ 0	1.64E-3	4.84E-3	0.299	3.28E-7	3.34E-7	3.29E-7	1.23E-5
Model 3, **TKE** > 0	3.02	2.37	6.69	1.00E-4	1.73E-4	1.52E-3	4.31E-2
Model 4, **TKE** > 0	2.65	1.63	5.37	5.22E-5	1.61E-4	7.37E-4	4.03E-2

As expected, we see from Table 4.5 that model 3 of course performs significantly better on the samples without TKE, as the *rmse* is almost an order of magnitude larger for all variables for

model 4. However, the errors are of very small magnitude in both cases, since the variables are essentially just very close to zero. For the samples with TKE, on the other hand, model 4 is significantly better than model 3. Thus, the benefit from separating the dataset according to TKE value is larger than the loss in accuracy by setting the low values to constants. And if one takes into account the computational saving by only using a neural network for about a third of the model levels, model 4 is clearly preferable.

Comparing the last three models in 4.4, which differ only in choice of loss function, we see that model 6 performs best for all variables, regardless of which statistical measure we look at. Thus, the mean absolute error of the physical values is the loss function that give the best model performance.

Step 2: Examining the relation between model performance and network size.

To optimize the network size, 9 different models are tested. Except for the model size, all 9 models are identical model 6 in step 1. Since the same datasets, scaling and loss function are used for all models, the convergence values of the losses can now be compared directly. In Figure 4.12, the learning curves for all 9 models are plotted together for stable samples (left) and unstable samples (right). Both the loss on the training set and validation set are shown. For both the stable and unstable samples, we clearly see that the performance improves, when more layers or nodes are added to the network. Note, however, that for the largest networks,

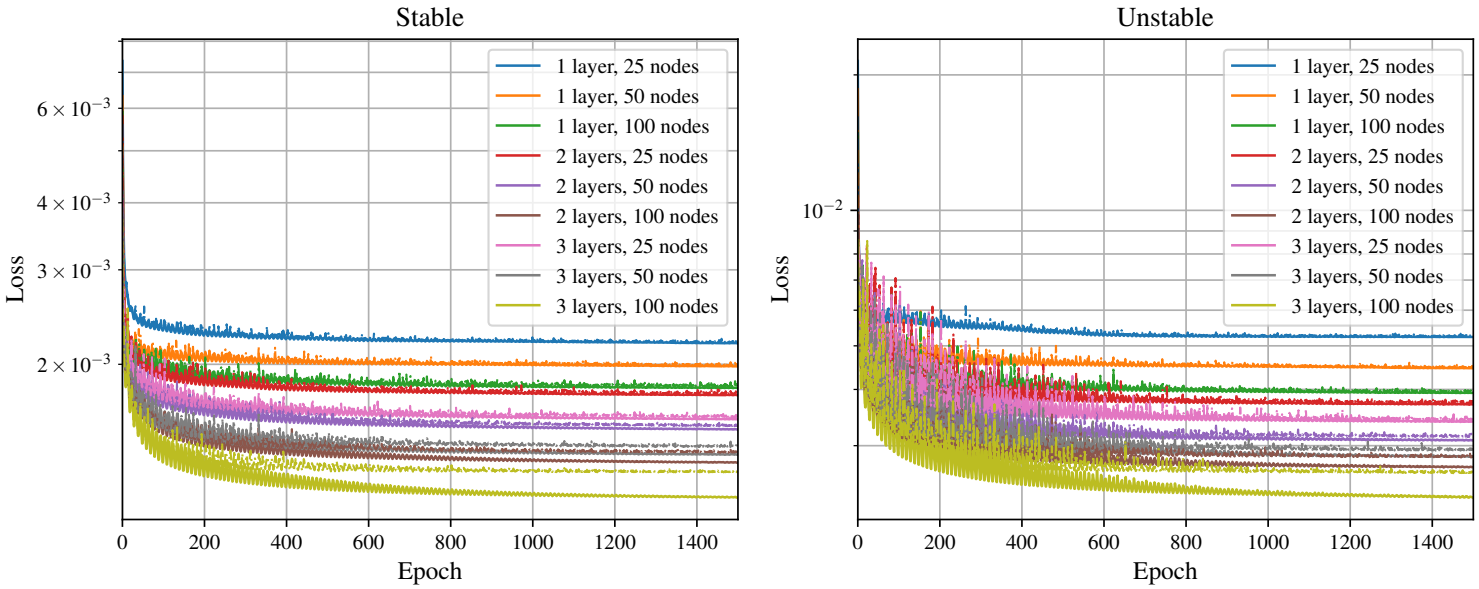


Figure 4.12: Learning curves for models with different numbers of layers and nodes, trained on stable samples (left) and unstable samples (right). The losses on the training set is drawn as the thick solid lines, and the legends show the corresponding model setup. The thinner dashed lines show the loss on the validation set.

the training loss continues to decrease, while the validation loss stabilizes, e.g. the networks with $2-3$ layers and 100 nodes per layer. This is a sign of overfitting, since the networks start learning patterns that are present only in the training data. Although the larger networks do perform better on the validation data, overfitting should of course be avoided.
The model with 3 layers and 25 nodes and the model with 2 layers and 50 nodes both seem to be good choices for a well performing model that does not overfit to the training data.

The number of layers and number of nodes, however, does not directly indicate the number of computations needed to make a prediction. Therefore, in Figure 4.13, the loss is plotted as function of the number of model parameters. The exact time it takes to make a prediction will depend on how the model is implemented. The computations needed depend on the structure of the network as well, since the weights are multiplied with the nodes, whereas the biases are added to the nodes. In addition, vectorization might not be equally efficient for different matrix sizes. However, as a rough approximation, the prediction time is assumed to scale with the number of model parameters. The losses are computed on the test dataset, and the exact values therefore might differ from those in Figure 4.12. The models compared in Figure 4.13 are those that performed best on the validation set. Hence, the risk of overfitting to the training data should be reduced.

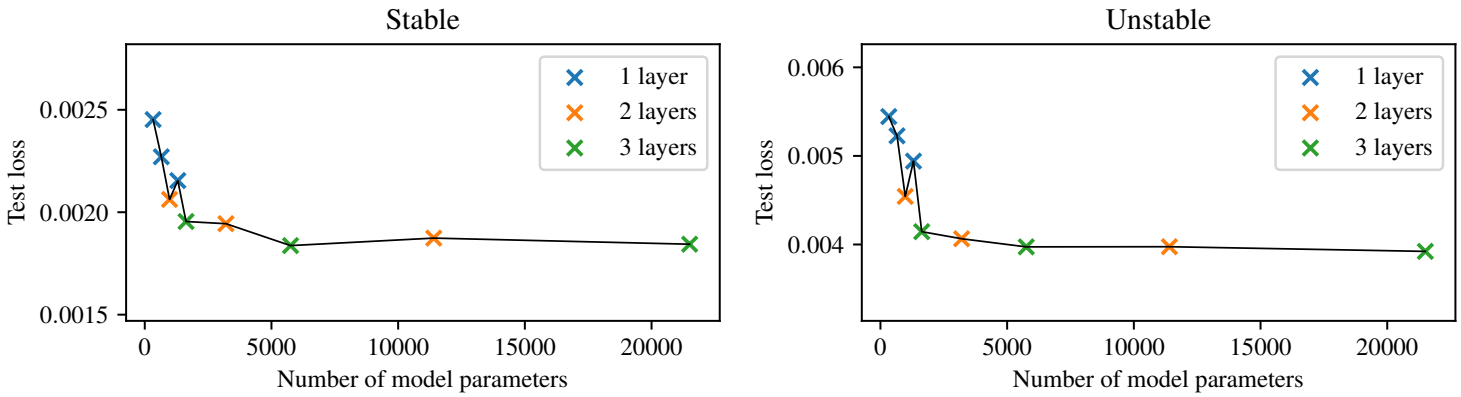


Figure 4.13: Loss as function of the number of model parameters, for stable samples (left) and unstable samples (right). The colors indicate the number of layers in the model.

It looks like the models with only 1 layer does not perform as well as those with 2 and 3 layers, even when they have a comparable number of model parameters. When evaluated on the test data, the largest models actually perform either similar to or slightly worse than the smaller models, although they performed better on both the training and validation datasets. This is a very interesting result: it means that when the model starts to overfit to the training data, it is not sufficient to choose the model that performs best on the validation data to avoid overfitting. We will return to the discussion in Chapter 6. Regardless what the explanation is, it is clear

that the large networks are not suitable for the problem.

The models with 3 layers and 25 nodes and the model with 2 layers and 50 nodes perform quite similarly, so the model with 3 layers and 25 nodes is a good compromise between efficiency and accuracy for both the stable and unstable samples.

Step 3: Optimizing batch size and activation.

Before the model is implemented, different batch sizes are tested to examine, whether this has an impact for the model performance. In Figure 4.14, four models with different batch sizes are tested. In addition to 1024, which has been used so far, the batch sizes 256, 512 and 2048 are tested.

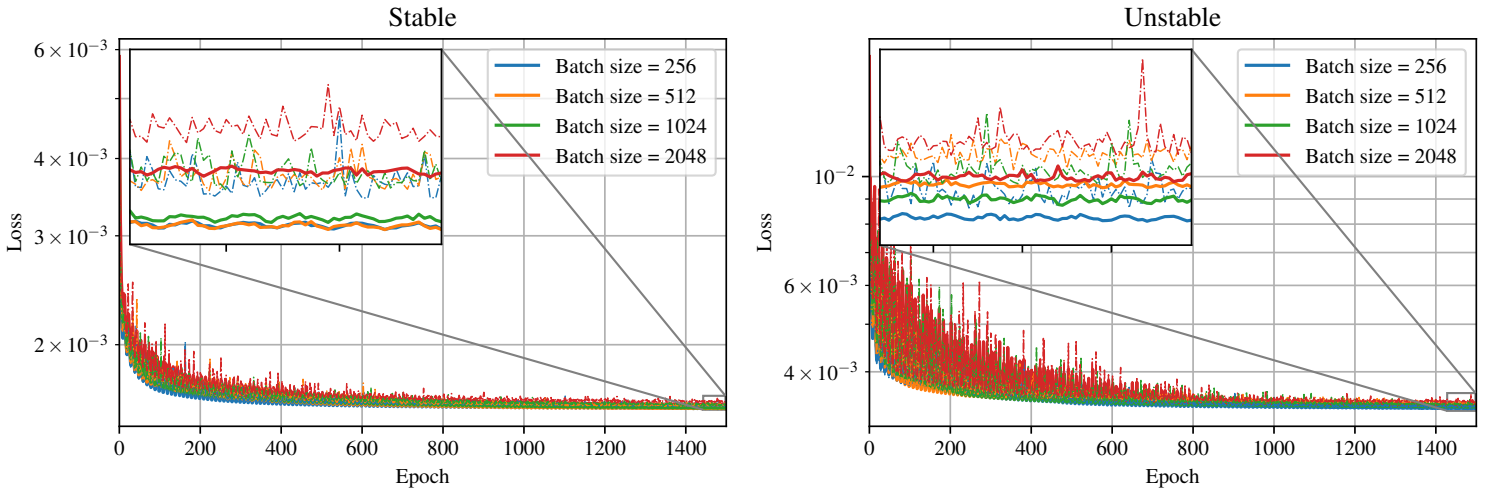


Figure 4.14: Learning curves for models trained with different batch sizes, for stable samples (left) and unstable samples (right). The losses on the training set is drawn as the thick solid lines, and the legends show the corresponding batch size. The thinner dashed lines show the loss on the validation set.

From Figure 4.14, we see that all models converge to roughly the same loss values. Zooming in, however, we do see a small difference, and in both cases the model trained with batch size of 2048, does not seem to perform as well as the others. For the unstable samples (right), the spread is a bit larger, but it does not seem to be systematic. Hence, nothing indicates that a different batch size will improve the result significantly.

Similarly, in Figure 4.15, we show the results of four models trained with different activation functions. We see that the ReLU and leakyReLU activations perform slightly better than the sigmoid and tanh activations. The leakyReLU may perform slightly better than the regular ReLU function, however, the difference is very small, and since the model training cannot be expected to be identical every time, nothing definitive can be concluded.

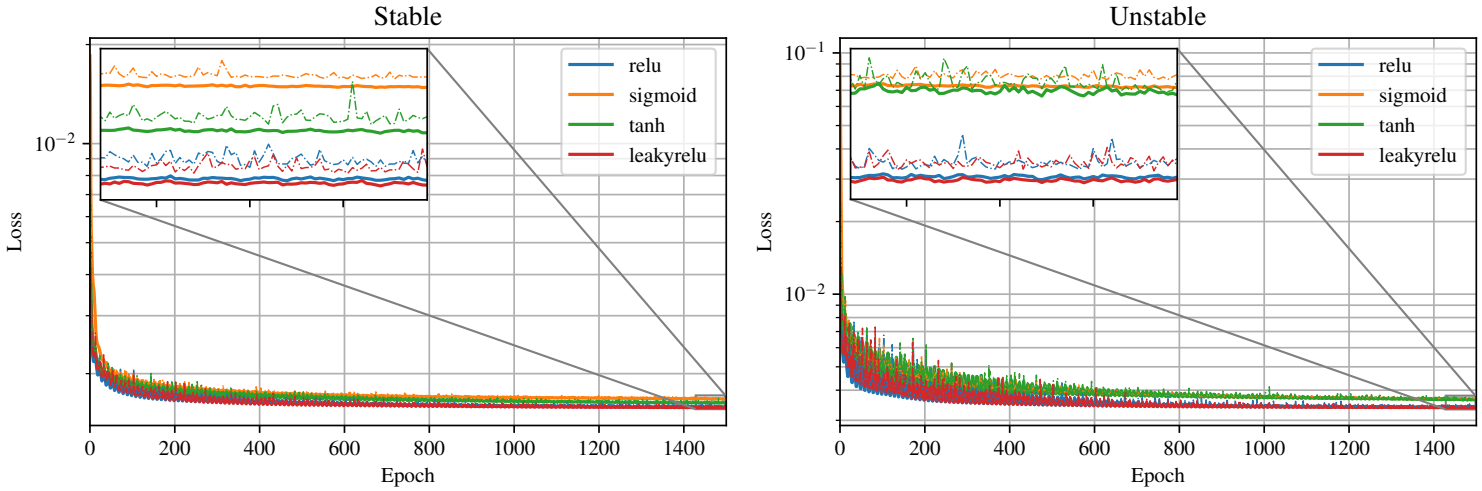


Figure 4.15: Learning curves for models with different activation functions, for stable samples (left) and unstable samples (right). The losses on the training set is drawn as the thick solid lines, and the legends show the corresponding activation. The thinner dashed lines show the loss on the validation set.

To sum up, the optimal model for both stable and unstable samples is a neural network consisting of 3 layers with 25 nodes in each layer. The activation function is the ReLU function, and the loss function is the mean absolute error evaluated on the physical values, i.e. not the direct output from the network but the exponential function of the output. In addition, the model was trained using Adam with the cyclic learning rate schedule and a batch size of 1024 samples.

4.4.6 Developing a model with fewer input variables

As discussed earlier, a model using fewer input variables is developed as an attempt to avoid potential overfitting to poorly represented input variables. Therefore, for this model, Θ, Q_v and Q_c are omitted from the input variables. Since the physical problem is unchanged, it will be assumed that most of the results obtained in Section 4.4.5 are applicable for this model, too. Thus, we use the same data categorization, data processing, loss function, batch size and activation function. However, since the model has fewer input variables, the overall complexity of the problem is reduced, and the relation between model performance and network size must be expected to be different. Therefore, the model is tested with the same 9 network sizes as described in step 2 in Section 4.4.5.

In Figure 4.16, the learning curves for all 9 models are plotted together for stable samples (left) and unstable samples (right). Both the loss on the training set and validation set are shown. Again, the number of epochs necessary was estimated visually, but now 1000 epochs seemed appropriate. The best model is still considered the one that performs best on the validation data. The result is very similar to that shown in Figure 4.12 for the model including

more input variables. Note again that the largest networks tend to overfit.

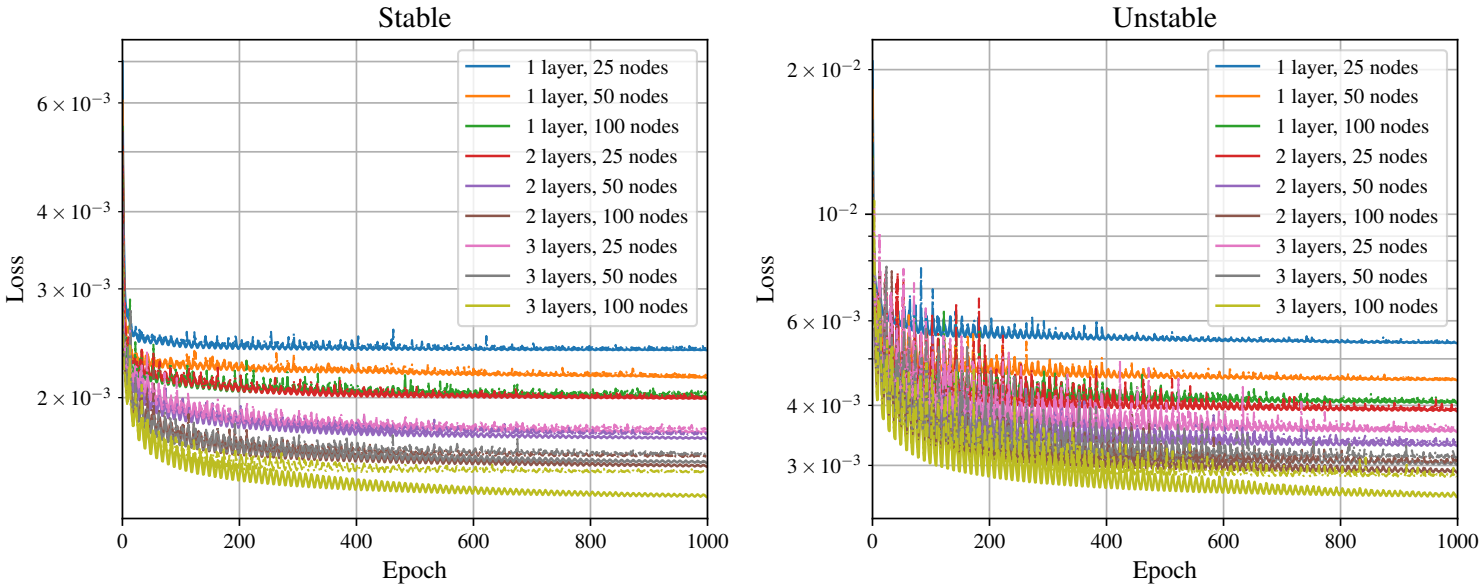


Figure 4.16: Learning curves for models with different numbers of layers and nodes, trained on stable samples (left) and unstable samples (right). The losses on the training set is drawn as the thick solid lines, and the legends show the corresponding model setup. The thinner dashed lines show the loss on the validation set. Despite the similarity with Figure 4.12, note that the models here use fewer input variables.

For both the stable and unstable samples, the models with 3 layers and 25 nodes and with 2 layers and 50 nodes again converge to lower loss values than the smaller models. However, for the stable samples, both models seem to overfit slightly to the training data.

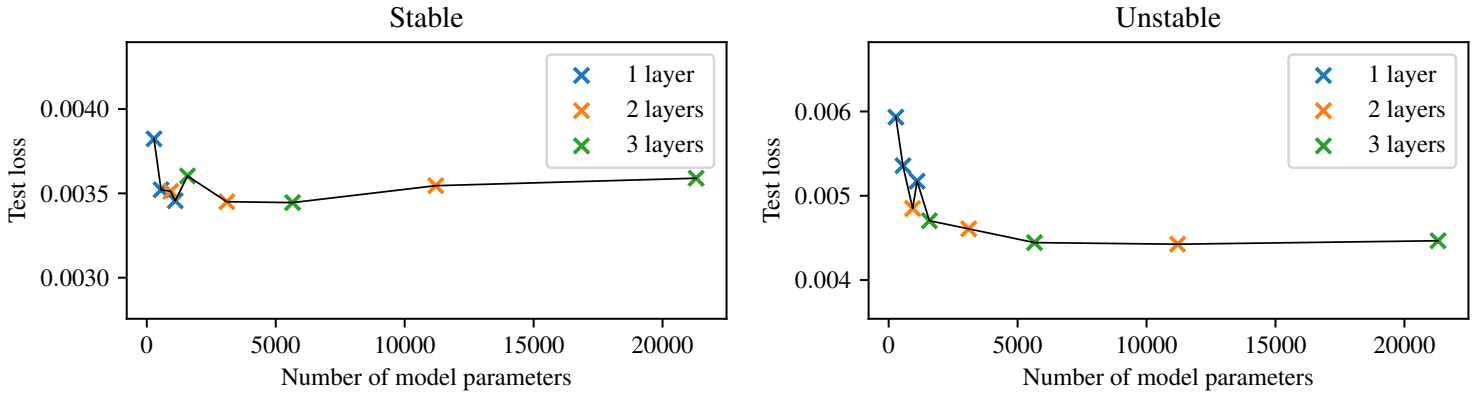


Figure 4.17: Loss as function of the number of model parameters, for stable samples (left) and unstable samples (right). The colors indicate the number of layers in the model. Despite the similarity with Figure 4.13, note that the models here use fewer input variables.

In Figure 4.17, the test loss for each model is plotted as function of the number of model parameters, which we will use as a measure of the prediction time. Again, we see that the largest models perform worse on the test data compared to some of the smaller models, suggesting that these are not suitable for the problem.

Based on the results shown in Figure 4.17, the best model for the stable samples seems to be the network with 1 layer and 100 nodes, whereas the best model for the unstable samples seems to be the network with 3 layers and 50 nodes. The latter, however, overfits to the training data indicating that it might not be the best choice. This is seen from Figure 4.16, where the training loss (gray solid line) converges to a lower value than the validation loss (gray dashed line). Instead, the model with 3 layers and 25 nodes is chosen for the unstable samples as a good compromise between efficiency and accuracy, while avoiding overfit.

For comparison, a smaller model is chosen as well, where the network with 1 layer and 50 nodes is used for the stable samples, and the network with 2 layers and 25 nodes is used for the unstable samples.

4.4.7 Model comparison

The three models that has been selected as the "optimal" models in the preceding sections are compared here, before they are implemented in WRF. Table 4.6 lists the different model specifications. Note that the names: model 1, 2 and 3 has nothing to do with the earlier numbering in of the models. The models are compared using the test dataset and the error measures from Equation (4.5). The results are shown in Table 4.7.

Table 4.6: The table show different model setups for the three "best" models described in the preceding sections.

	Input variables	**Model size for stable samples**	**Model size for unstable samples**
Model 1	q, B, S, z, u_*, Q_0, Θ, Q_v, Q_c	3 layers with 25 nodes in each.	3 layers with 25 nodes in each.
Model 2	q, B, S, z, u_*, Q_0	1 layers with 50 nodes in each.	2 layers with 25 nodes in each.
Model 3	q, B, S, z, u_*, Q_0	1 layers with 100 nodes in each.	3 layers with 25 nodes in each.

Table 4.7: Performance of the models described in Table 4.6. For each variable and each error measure, the best performance is highlighted with red color, and the worst is highlighted with blue.

	Stat	K_h	K_m	L	B_p	$-K_hB$	$K_m\sqrt{S}$	q^3/L
Model 1	l_1	0.04880	0.03095	0.09773	0.03650	0.06123	0.02009	0.05082
	l_2	0.1060	0.06904	0.1458	0.05529	0.1614	0.05026	0.09857
	r	0.9940	0.9974	0.9868	0.9985	0.9869	0.9987	0.9953
Model 2	l_1	0.06696	0.04308	0.1474	0.07612	0.09900	0.03718	0.1118
	l_2	0.1317	0.08510	0.1946	0.09490	0.2183	0.1232	0.1612
	r	0.9906	0.9960	0.9762	0.9956	0.9760	0.9933	0.9874
Model 3	l_1	0.06659	0.04091	0.1448	0.07704	0.1034	0.03888	0.1097
	l_2	0.1287	0.08459	0.1922	0.1001	0.2064	0.09866	0.1450
	r	0.9911	0.9961	0.9769	0.9952	0.9785	0.9961	0.9894

Not surprisingly, the model including Θ, Q_v and Q_c as inputs performs significantly better than the two other models. Further, model 3 performs better than model 2 for most variables, which is expected since model 3 had a lower loss values on the test data.

In addition, in Figure 4.18 and 4.19, examples of predictions of the neural networks are plotted. The two examples selected, are those in test dataset with the most extreme surface stability conditions. Figure 4.18 thus shows an example with statically stable surface conditions, while Figure 4.19 shows an example with unstable surface conditions. Note, however, that in the first case, the shear production near the surface is more than a magnitude larger than in the second case.

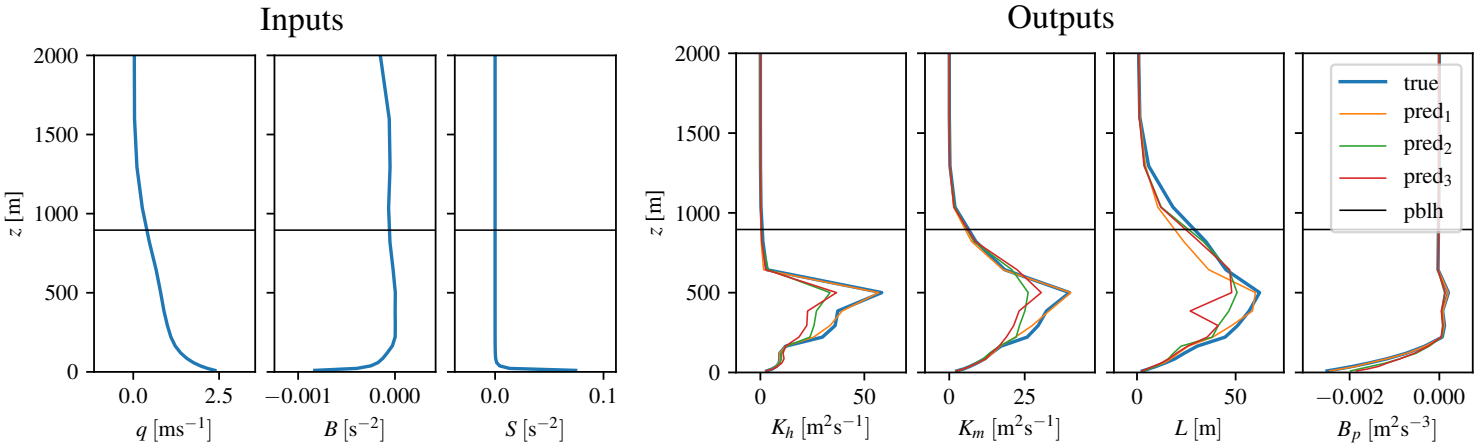


Figure 4.18: Example of predictions by the three neural networks. The three plots to the left show the profiles of the three inputs, q, B and S, while the four plots to the right show the output variables K_h, K_m, L and B_p. Both the "true" values and the predictions by the three networks are shown. The specific example shown, is the air column from the test dataset with the smallest value of the surface sensible heat flux, $Q_0 \approx -140\text{W/m}^2$, i.e. most statically stable surface conditions. The value of the friction velocity for this case is $u_* \approx 1\text{m/s}$. The planetary boundary layer height, *pblh*, is showed as well. The location is 64°31'N10°41'E (over the ocean very close to Norway's west coast), and the time is 2017.01.07 06 UTC.

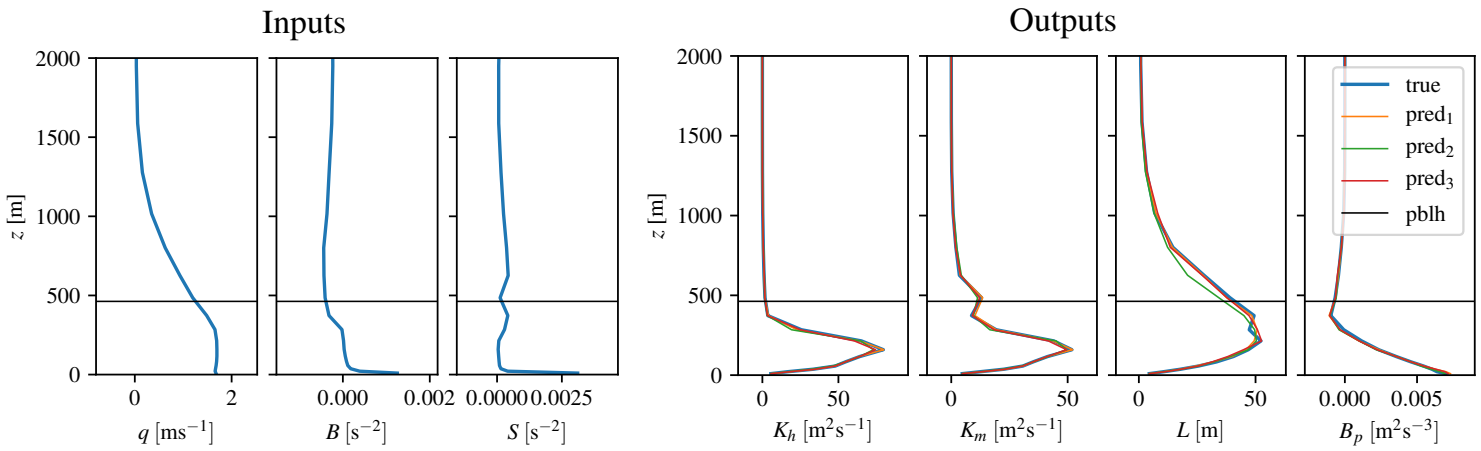


Figure 4.19: Example of predictions by the three neural networks. Similar to Figure 4.18, but the example shown here is the one from the test dataset with the largest value of surface sensible heat flux $Q_0 \approx 254\text{W/m}^2$. The value of the friction velocity for this case is $u_* \approx 0.59\text{m/s}$. The location is 53°71'N9°65'E (over Northern Germany), and the time is 2018.07.26 11 UTC.

The Figures 4.18 and 4.19 indicate that model 1 perform best, especially for the stable case, where model 2 and 3 underestimate the diffusivities and L. For the unstable case, all three models predict a profile very similar to the "true" values (the MYNN predictions). It should be stressed that nothing general can be concluded based on the two examples. When selecting only two examples from a large dataset, there is a considerable risk of selecting examples that does show something general. The point of showing the examples, however, is not to conclude which scheme performs best but merely to visually demonstrate abilities of the neural network models. A few additional randomly selected examples are shown in Appendix D.

Finally, in Figure 4.20, the predicted values are plotted as function of the true values for all three models. The plots are based on the independent test dataset. Note the logarithmic axis on the colorbar. For K_h, the correlation plots and the r^2 values are quite similar for the three models, however, there seem to be a cluster of data points, for which all three models underestimate the values. For K_m, the correlation plots and the r^2 values look quite good for all three models. However, for the predictions of L, we see a much larger variance, indicating that the neural networks might not have all the necessary information available. Although, the r^2 values are somewhat higher for model 1, the correlation plots for L have almost the same "shape" for the three models. This suggests that the three models have problems with same categories of data points. Hence, the large deviations do not seem to be related to the presence of the input variables Θ, Q_c and Q_v. Instead, this could be because only local variables are selected as inputs to the model. Recall from the expression of the turbulent length scale in Equation (1.36), that L also depends on the turbulent structure of the vertical profile, suggesting that some non-local information would be useful. For B_p, all three models seem to give good predictions for the

statically unstable samples (positive values), while for the stable samples, model 2 and model 3 does appear to give slightly worse predictions. This may very well be related to the missing input variables Θ, Q_c and Q_v. However, the B_p predictions of model 2 and 3 still look reasonable and both have r^2 values exceeding 0.99.

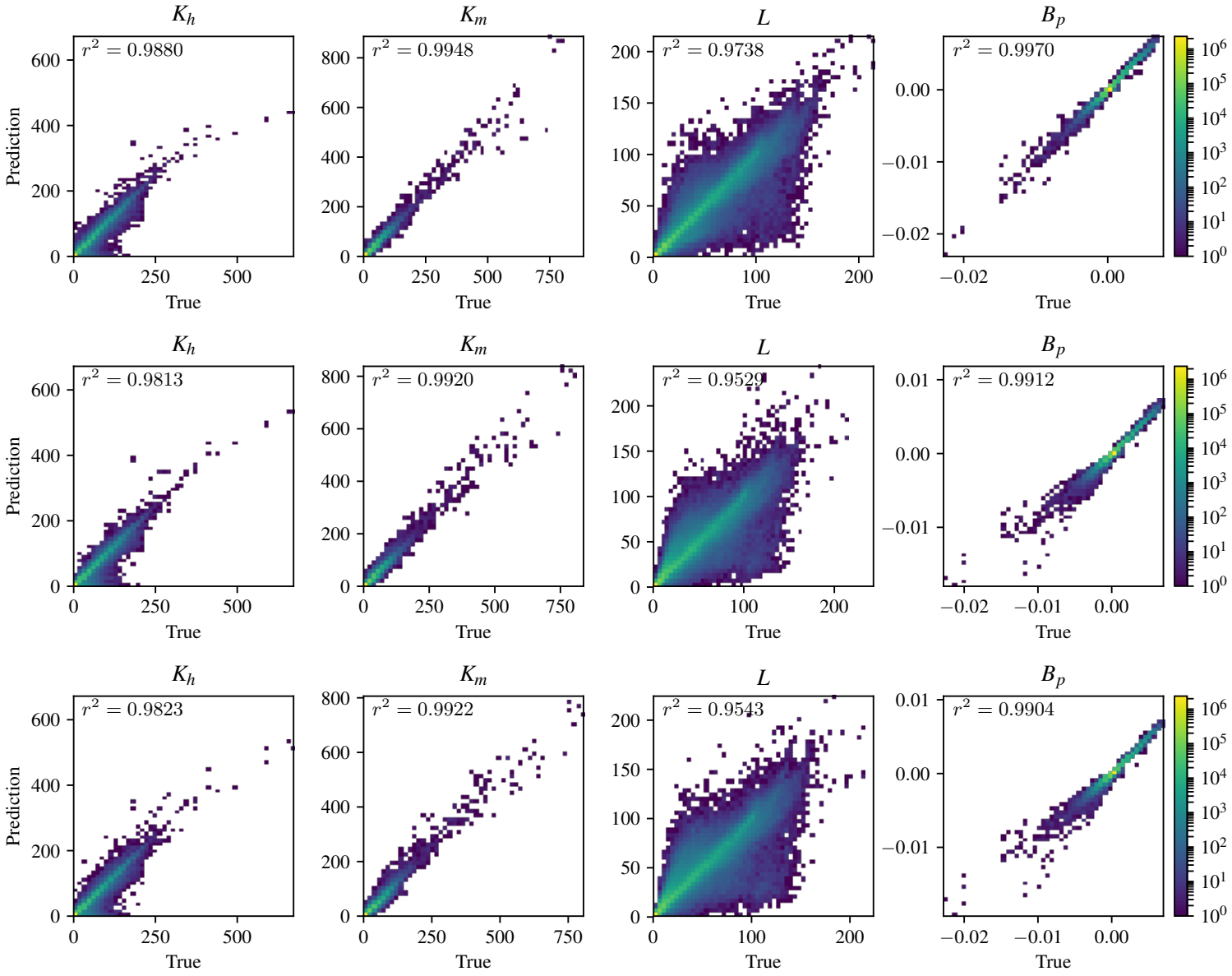


Figure 4.20: Correlation plots showing predictions as function of true values. The plots are constructed as 2D histograms to be able to see the density of data points in different locations (note that the colorbar axis is logarithmic). The first row are predictions by model 1, the second row by model 2, and the third row by model 3. The plots are based on predictions on the independent test dataset, and the errors and correlation values therefore correspond to those shown in Table 4.7.

Chapter 5

Implementation and test

To evaluate the performance of the neural networks, it is not enough to test them on an independent dataset sampled from new simulations with WRF. This would only give an estimate of the errors after one time step, but if the neural networks have systematic errors, it may cause feedback mechanisms and induce model drift. We will discuss this further in Chapter 6. Therefore, the three neural network based PBL schemes have been implemented in WRF and tested against some of the existing PBL scheme options in WRF.

In Section 5.1, we will describe the methods used to compare the different PBL schemes. Next, Section 5.2 briefly describes how the neural network based schemes are implemented in WRF. In Section 5.3, all three neural network based PBL schemes are compared to simulations made with the MYNN scheme. The predictions of the scheme that performs best are then compared to both the MYNN scheme and to two other PBL schemes in Section 5.4. Finally, in Section 5.5, we compare the computational cost of the different PBL schemes.

5.1 Method for model comparison

We will use the term ANN PBL schemes (artificial neural networks) for the neural network based parameterizations. Two simulations have been performed with each of the ANN schemes as well as with the MYNN scheme. Assuming that the MYNN scheme predicts the truth, we can estimate the "error" of the predictions for each scheme. Although this assumption might generally not be true, it is justified in our case, since the neural networks try to imitate the MYNN scheme.

The simulations were performed with the same setup as for the data generation described in Section 2.3. The domain, however, was extended to cover most of Europe, to examine the limitations of the ANN schemes, see Figure 5.1. Two different weather scenarios were selected as test cases, one during summer and the other during winter (initial times: 2017.01.11 and 2018.08.02 at 06 UTC). None of these dates were used for the generation of training data, since we want to get a truly independent estimate of the performance.

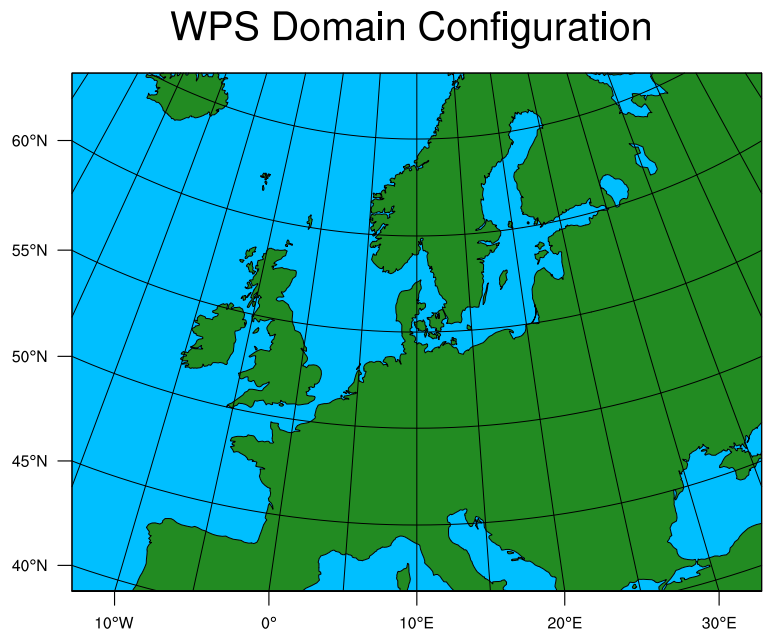


Figure 5.1: Domain used for testing the neural network based PBL schemes.

The PBL scheme is particularly important for the prediction of surface energy balance, and therefore we will look at the surface fluxes of sensible heat and latent heat as indicators of each scheme's performance. Also the friction velocity u_* and the planetary boundary layer height, *pblh*, will be considered, since the friction velocity is the square root of the kinematic surface momentum flux, and the *pblh* indicate whether the scheme captures the overall boundary layer dynamics (described further in Section 5.4). In addition, we will look at the 2m temperature and the 10m wind, since these depend on the accumulated effects of the surface fluxes

In Section 5.3 and 5.4, we will visually compare the predictions by showing maps of the different variables described above. In addition, we compute the root mean square error, $rmse$, and the pattern correlation coefficient, r

$$rmse = \sqrt{\sum_i \sum_j \left(\hat{\psi}_{ij} - \psi_{ij}\right)^2}, \qquad r = \frac{\sum_i \sum_j \left(\hat{\psi}_{ij} - \overline{\hat{\psi}}\right)\left(\psi_{ij} - \overline{\psi}\right)}{\sqrt{\sum_i \sum_j \left(\hat{\psi}_{ij} - \overline{\hat{\psi}}\right)^2}\sqrt{\sum_i \sum_j \left(\psi_{ij} - \overline{\psi}\right)^2}}, \tag{5.1}$$

where ψ_{ij} and $\hat{\psi}_{ij}$ are the "true" value and the predicted value of some variable, valid at the location with horizontal indexes i, j. The $rmse$ is a measure of the magnitude of the average "error", whereas r indicates how similar the overall features in the two fields are, i.e. it considers the spatial pattern of the anomalies rather than the magnitude. However, one must be careful when interpreting $rmse$ and r. Running two different numeric models for a chaotic system will inevitably give different results. So, how do we evaluate the model, when we know that its

prediction will eventually diverge from the "target" field? In addition to visually evaluating whether the predictions look physically realistic, one can try to determine whether the solutions diverge *too fast*. For this purpose, the same simulations are also performed with two different PBL scheme options in WRF. By comparing the MYNN scheme to these two other PBL schemes, we get an estimate of the general "level of agreement" between the already existing PBL scheme options. The two additional PBL schemes used are the MYJ scheme [16], which is similar to the MYNN scheme in the sense that it is based on the original Mellor-Yamanda papers and has TKE as a prognostic variable, and the YSU scheme [15], which is a first order closure model. Further, in Section 5.4, a few examples of profiles of temperature and wind speed will be compared visually to examine whether the ANN scheme produces physically realistic profiles.

It should be noted that several of the PBL scheme options in WRF are only compatible with specific surface flux schemes. Therefore, the simulations performed with the MYJ and YSU schemes each uses a different surface flux scheme, which is compatible with the specific PBL scheme.

5.2 Implementing neural networks in WRF

In Chapter 4, we developed three models that use neural networks to predict the diffusivities. However, to construct a complete PBL scheme, the tendencies also need to be computed. Further, the neural networks need turbulence kinetic energy as input, and this variable is not described in the initial state. However, the original Fortran module for the MYNN scheme[1] already contains subroutines for computing the tendencies and estimating the TKE for the first time step. Thus, the easiest way to construct the ANN scheme was to use the Fortran module for the MYNN scheme as the basis and then substitute the relevant subroutines with new subroutines making predictions with the neural networks. In the original MYNN module, there is one subroutine computing the diffusivities and the turbulent length scale, and another subroutine computing β_θ and β_q (the partial-condensation scheme). Thus, in the ANN scheme, the neural networks substitute both of these subroutines, while the remaining code is kept as the original with a few minor changes.

As described in Section 3.1, making predictions with a neural network simply consists of a sequence of matrix products followed by activations. Thus, the weight matrices and bias vectors must be defined in the Fortran module, and then they need to be assigned the specific values found during training of the neural networks. We test two different functions for computing matrix products in Fortran; the first is Fortran's intrinsic `matmul()` function, and second is the `sgemm()` function from the *OpenBLAS* library (Basic Linear Algebra Subprograms). This library provides optimized functions for linear algebra operations [23].

[1] The source code can be found on WRF's official GitHub-page [22]

5.3 Comparison of the three ANN schemes

First, we want to find the best of the three models found in the previous Chapter. The neural network based PBL schemes will be denoted ANN_1, ANN_2 and ANN_3, where the numbers corresponds to the model numbers 1, 2 and 3 in Section 4.4.7.

As described in Section 5.1, two simulations are performed for each PBL scheme, with initial times 2017.01.11 06 UTC and 2018.08.02 06 UTC, respectively. Each simulation is runs for 24 hours. Figure 5.2 shows plots of predictions (from the first simulation) of the surface fluxes of sensible heat and latent heat, the friction velocity and the planetary boundary layer height. The predictions are valid at 2017.01.12 06 UTC, i.e. 24 hours after the initial time. The left column shows the predictions of the MYNN scheme, while the three other columns show the difference between the prediction and the "true" value (predictions by the MYNN scheme). The differences are computed as $pred_{\text{ANN}} - pred_{\text{MYNN}}$, such that positive anomalies mean that the scheme predicts too high values.

From Figure 5.2, we see that the predictions of the ANN_1 scheme differs much more from the MYNN scheme, than the two other ANN schemes. Recall from Section 4.4.7 that model 1 was the model using Θ, Q_v and Q_c as additional input variables. Thus, the results support the hypothesis that model 1 have overfitted to the training dataset and therefore behaves unexpectedly in a new weather scenario. The sign of the anomalies indicate that the surface fluxes are generally underestimated. It should of course be noted that the surface fluxes are *not* computed by the neural networks directly, but by the surface flux scheme, which is identical in the four simulations. However, the surface fluxes depend on the atmospheric state near the surface, and therefore the surface fluxes do depend strongly on the PBL scheme as well. The anomaly fields of ANN_2 and ANN_3 are quite similar, and it looks like the errors are of approximately the same magnitude.

One can imagine that systematically underestimating the surface fluxes will lead to large anomalies in the surface fields of temperature, moisture and wind. In Figure 5.3, the 2m temperature and 10m wind fields are shown, also 24 hours after the initial time. For the wind fields, both wind speed and direction are shown. As expected, the predictions of the ANN_1 scheme show large anomalies in the temperature and wind fields, especially in the areas where the surface fluxes have large anomalies. In Figure 5.3, the anomaly plots for ANN_2 and ANN_3 are also noticeably different. The latter definitely have larger anomalies after the 24 hours, which is quite interesting, since model 3 (using the largest neural networks) performed better on the test dataset than model 2, see Section 4.4.7. Thus, these results suggest that the larger neural networks in model 3 have somewhat overfitted to the training dataset, although to much lesser extent than model 1. This of course raises the question of whether the neural networks in model 2 might also to some extent have overfitted? We will return to this question in the discussion in Chapter 6.

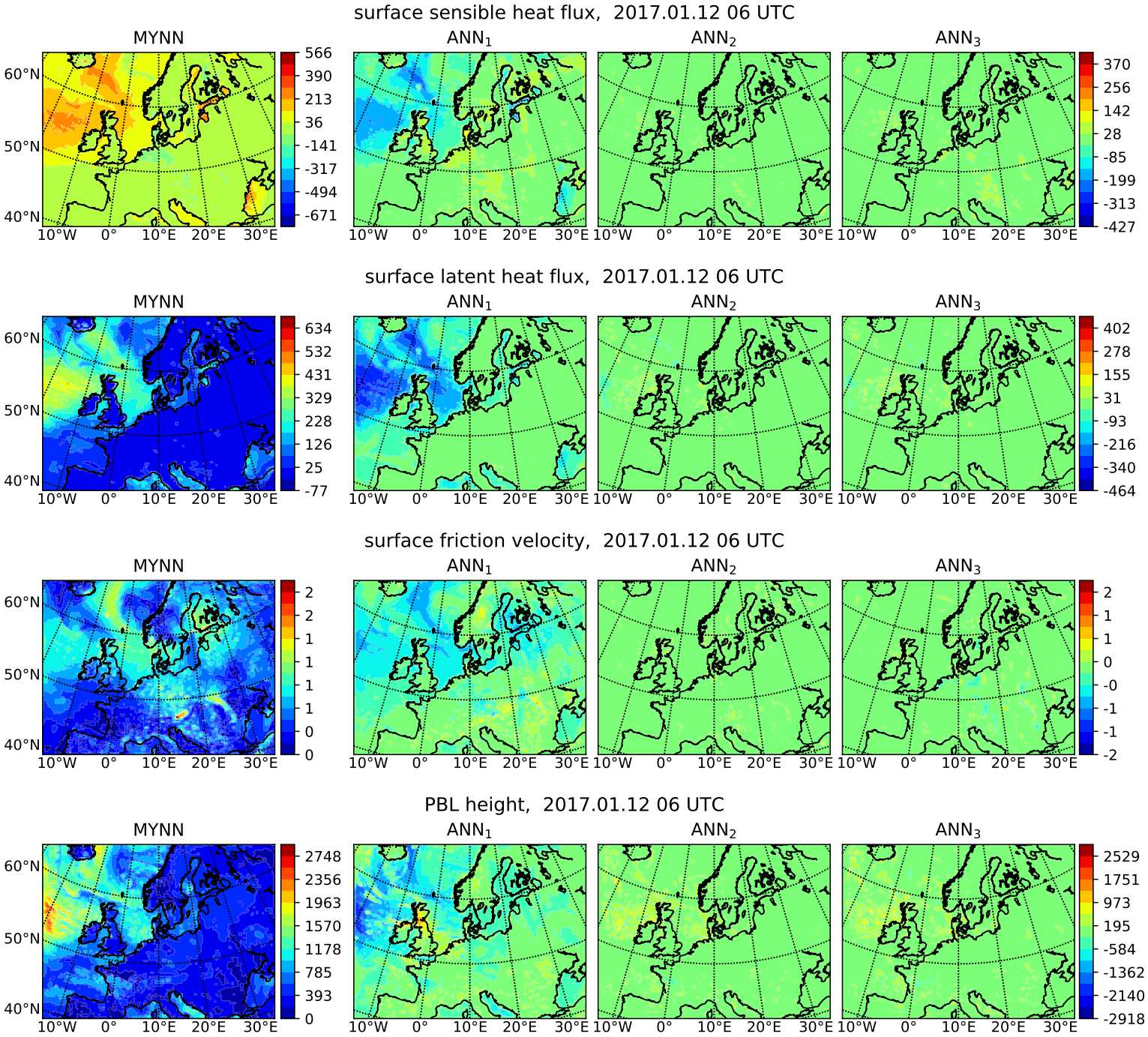


Figure 5.2: Predictions of surface sensible heat flux (first row), surface latent heat flux (second row), surface friction velocity (third row) and pblh (fourth row). First column show plots of the predictions of the MYNN scheme, while the three next columns shows the anomaly fields for the three ANN schemes, computed as $pred_{\text{ANN}} - pred_{\text{MYNN}}$. All predictions are valid at 2017.01.12 06 UTC, i.e. 24 hours after the initial time. The names ANN1, ANN2, ANN3 correspond to the model numbers in Section 4.4.7.

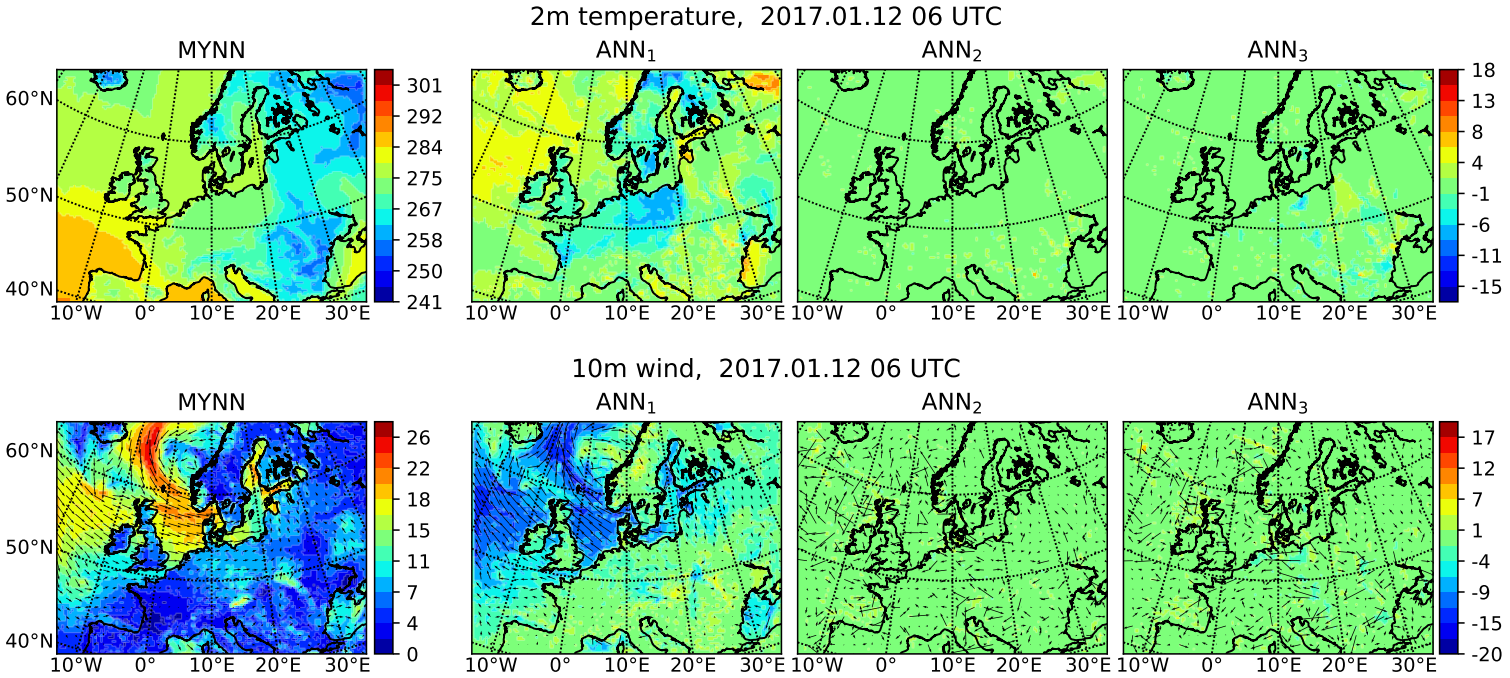


Figure 5.3: Similar to Figure 5.2, except that first row shows predictions of 2m temperature, and second row shows predictions of 10m wind.

As described in Section 5.1, the *rmse* and r of the predictions are computed for all three ANN schemes, for all time steps. In Figure 5.4, these measures are shown as function of number of hours after initial time. As expected, the predictions of the ANN_1 scheme have much higher *rmse* and lower r values compared to the other schemes. However, notice that the errors are large (and the correlations are low) already from the beginning and then almost constant throughout the simulation. This is not the case for the other schemes, where the errors increase (and the correlations decrease) as time goes. The ANN_2 scheme performs slightly better than the ANN_3 scheme for most variables, having both lower *rmse* and higher correlation with the MYNN scheme's prediction. For the 2m temperature and the 10m wind, WRF also outputs the initial fields, and Figure 5.4 shows that the errors of the predictions are non-zero even before the simulation begins, for all three ANN schemes. This is because of WRF's digital filtering initialization, DFI, which is applied before the simulation is started.

For the second test case, initiated at 2018.08.02 06 UTC, the simulation with the ANN_1 scheme was interrupted/crashed within the first 3 hours of simulation. Figure 5.5 show the 2m temperature for the initial time and the predictions after 1 and 2 hours for the MYNN scheme and all three neural networks. As before, the first column show the predictions of the MYNN scheme, while the other columns show anomalies. Here, we clearly see that the ANN_1 scheme causes the surface temperatures to grow to unphysical values, and 2 hours into the simulations

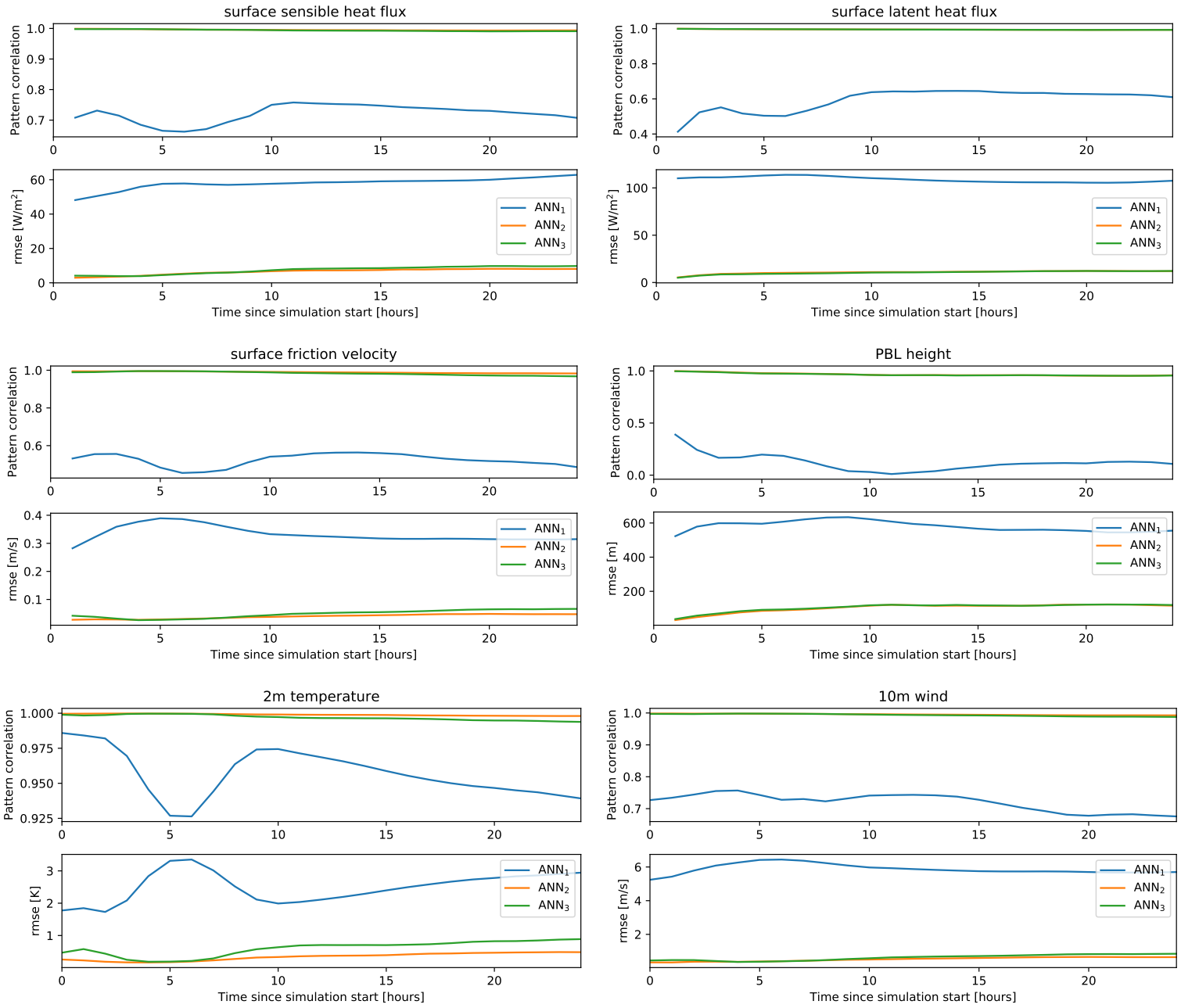


Figure 5.4: The plots in show the root mean square error, *rmse* and the pattern correlation, r as function of number of hours since the initial time. The titles indicate for which variable the measures are computed, and the legends show for which PBL scheme. Each title applies to the two plots below, showing r and *rmse*, respectively. The plots in this figure are based on the simulations initiated at 2017.01.12 06 UTC.

it is locally up to 49K higher than the true temperature. It is not difficult to imagine that when the increasing temperatures are fed as input to the neural networks, it will gradually cause a higher degree of extrapolation, which might result in some unpredictable positive feedback loop. This is discussed further in Chapter 6. Note again that even the initial temperature fields are significantly different because of the DFI.

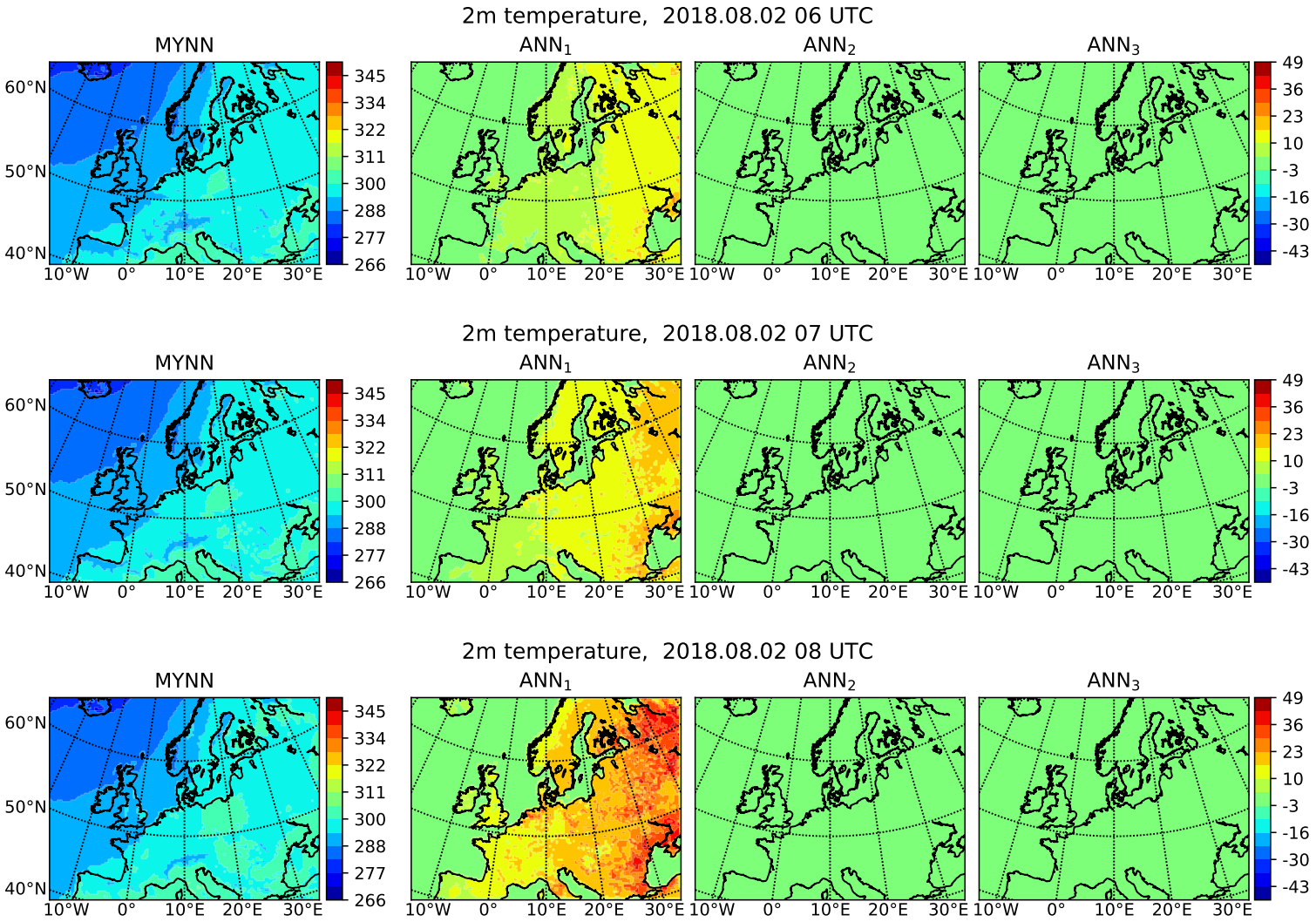


Figure 5.5: Predictions of surface sensible heat flux (first row), surface latent heat flux (second row), surface friction velocity (third row) and pblh (fourth row). First column shows plots of the predictions of the MYNN scheme, while the three next columns show the anomaly fields for the three ANN schemes, computed as $pred_{\text{ANN}} - pred_{\text{MYNN}}$. All predictions are valid at 2017.01.12 06 UTC, i.e. 24 hours after the initial time. The names ANN_1, ANN_2, ANN_3 correspond to the model numbers in Section 4.4.7.

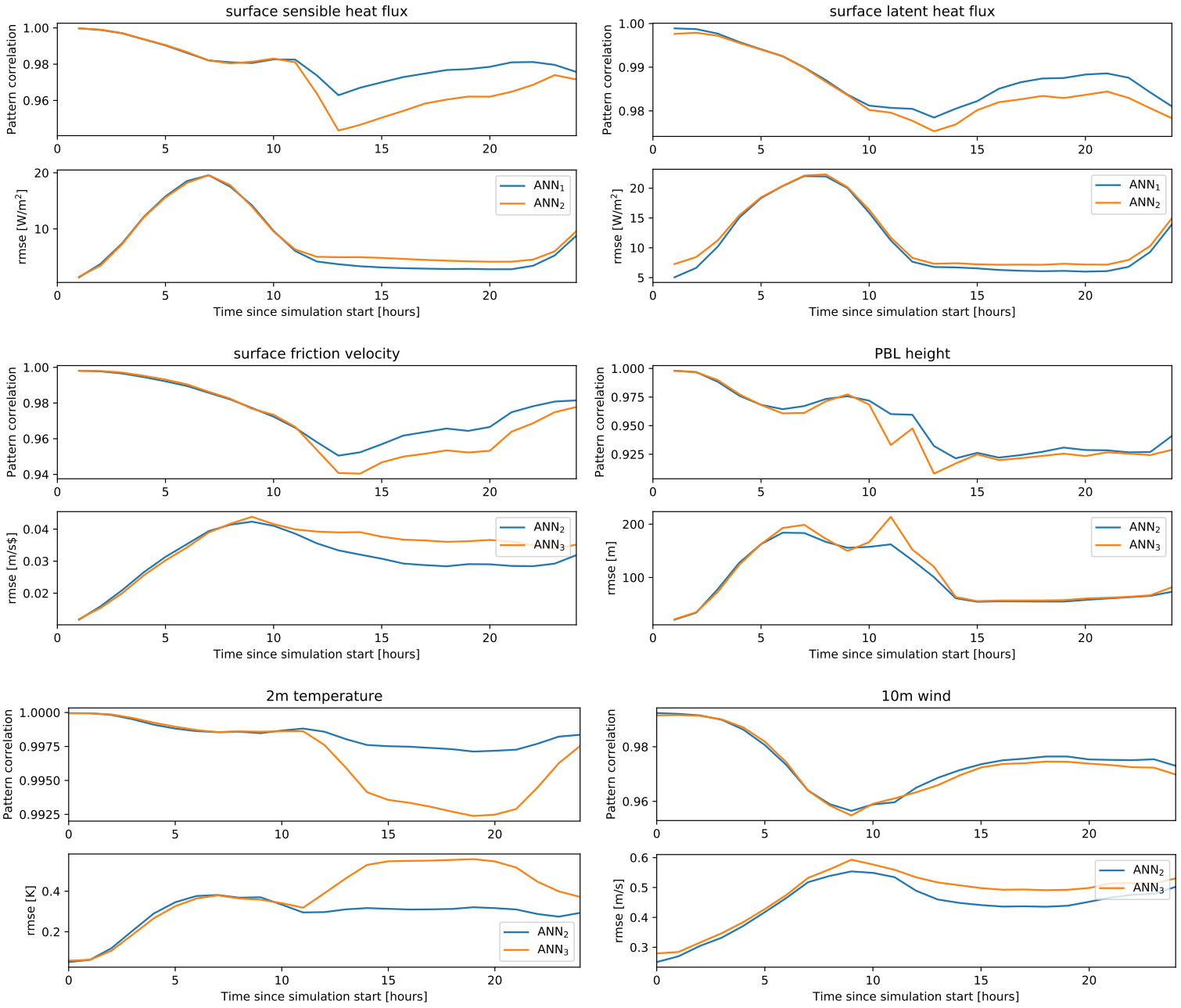


Figure 5.6: Similar to 5.4, except that the plots are based on the simulations initiated at 2018.08.02 06 UTC.

For the ANN_2 and ANN_3 schemes, plots of the predicted variables after 24 hours are shown in Appendix E, but the results are quite similar to those shown in the Figures 5.2 and 5.3. Figures 5.6 show the *rmse* and r as function of time for the second simulation for ANN_2 and ANN_3 scheme.

From both of the Figures 5.4 and 5.6, we see that the ANN_2 scheme performs slightly better than the ANN_3 scheme. As expected, the predictions of both schemes have high correlations and low errors in the beginning, and as time goes, the errors increase, and the correlations decrease. However, there is a clear dependency on the diurnal cycle as well, especially apparent for the surface heat fluxes in Figure 5.6. This is natural, since the surface heat fluxes in the summer are much higher during daytime than during night. Thus, the magnitude of the $rmse$ must be expected to be larger during daytime as well. Based on the two test cases, we conclude that the ANN_2 scheme is the best performing. Therefore, this is the scheme chosen for further tests in the following section.

5.4 Evaluation of the best ANN scheme

Now, we show the results of the additional simulations performed with the YSU scheme and the MYJ scheme. The same test cases as in the previous section are used, but the simulations are extended to run for 72 hours. The Figures 5.7-5.12 show plots of the predictions of the variables described in Section 5.1 after 12 and 72 hours. The figures are similar to those in the previous section, except now each figure show the predictions of just one variable. The plots show the predictions for the simulations initiated at 2017.01.11 06 UTC, while the predictions for the simulations initiated at 2018.08.02 06 UTC are shown in Appendix E.

From the Figures 5.7-5.12, we see that the predictions of the ANN scheme generally have smaller anomalies than the two other schemes. The predictions of MYJ scheme have the largest anomalies, and especially for the surface heat fluxes it differs quite significantly from the MYNN scheme. The results for the second simulation, shown in the Figures E.3-E.8 in Appendix E, show similar tendencies. It must be stressed that we cannot know, which of the PBL schemes is closest to the actual "truth". Therefore, we repeat that the anomalies of the predictions of the YSU scheme and the MYJ scheme cannot be interpreted as errors. The predictions of the these schemes are merely used to investigate, whether the anomalies of the predictions of the ANN scheme are of "reasonable" magnitude.

Further, in the Figures 5.13 and 5.14, we show the values of $rmse$ and r as function of time, for the two different test cases. These results show that the predictions of the ANN scheme have both higher pattern correlation and lower $rmse$ for all the variables, which corresponds well with the results in the Figures 5.7-5.12.

Thus, we have shown that the ANN scheme's predictions of fluxes and mean field variables close the surface are very similar to those obtained by using the existing PBL scheme options in WRF. Further, when comparing the ANN scheme to the YSU and MYJ schemes, it is the one that give predictions closest to the MYNN scheme, proving that the neural networks have learned to emulate the behavior of the scheme quite well.

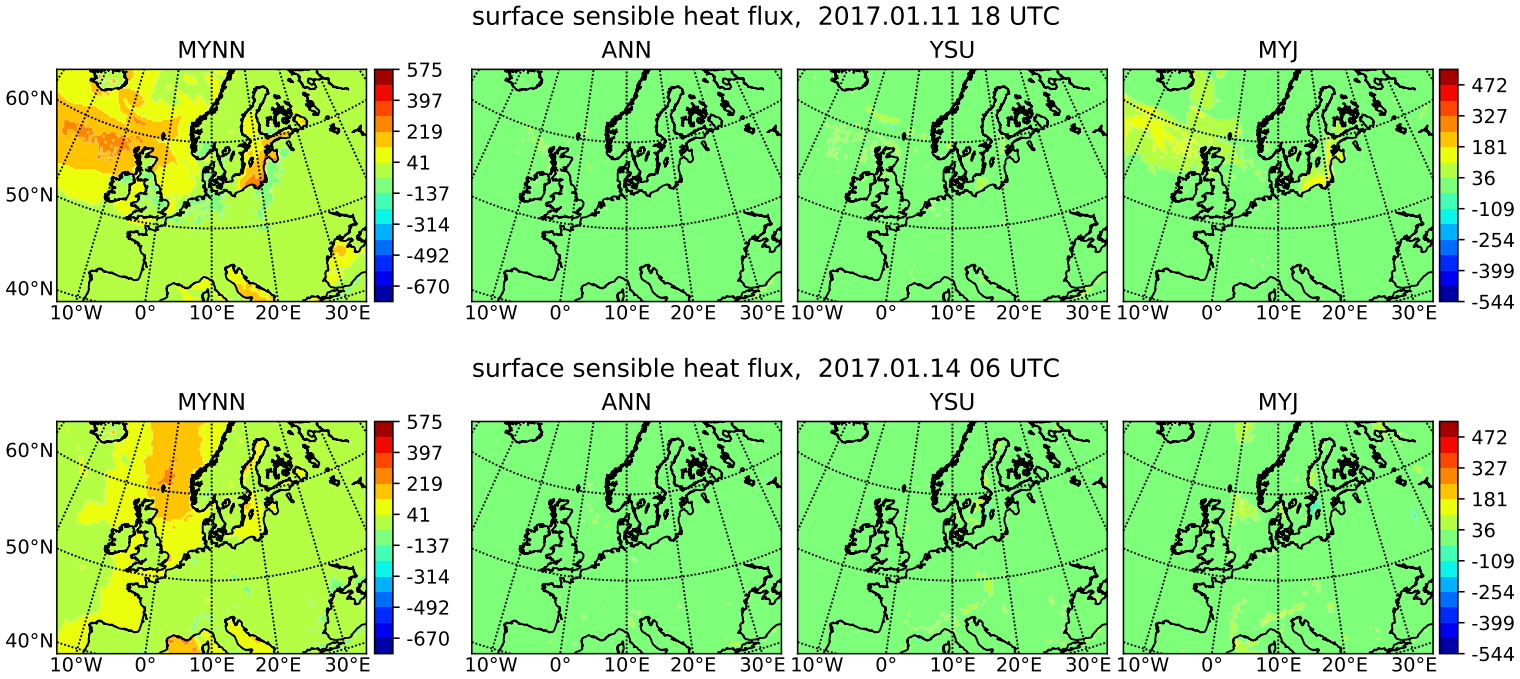


Figure 5.7: Predictions of surface sensible heat flux after 12 hours (first row), and after 72 hours (second row). First column shows plots of the predictions of the MYNN scheme, while the three next columns show the anomaly fields for the ANN scheme, YSU scheme and the MYJ scheme.

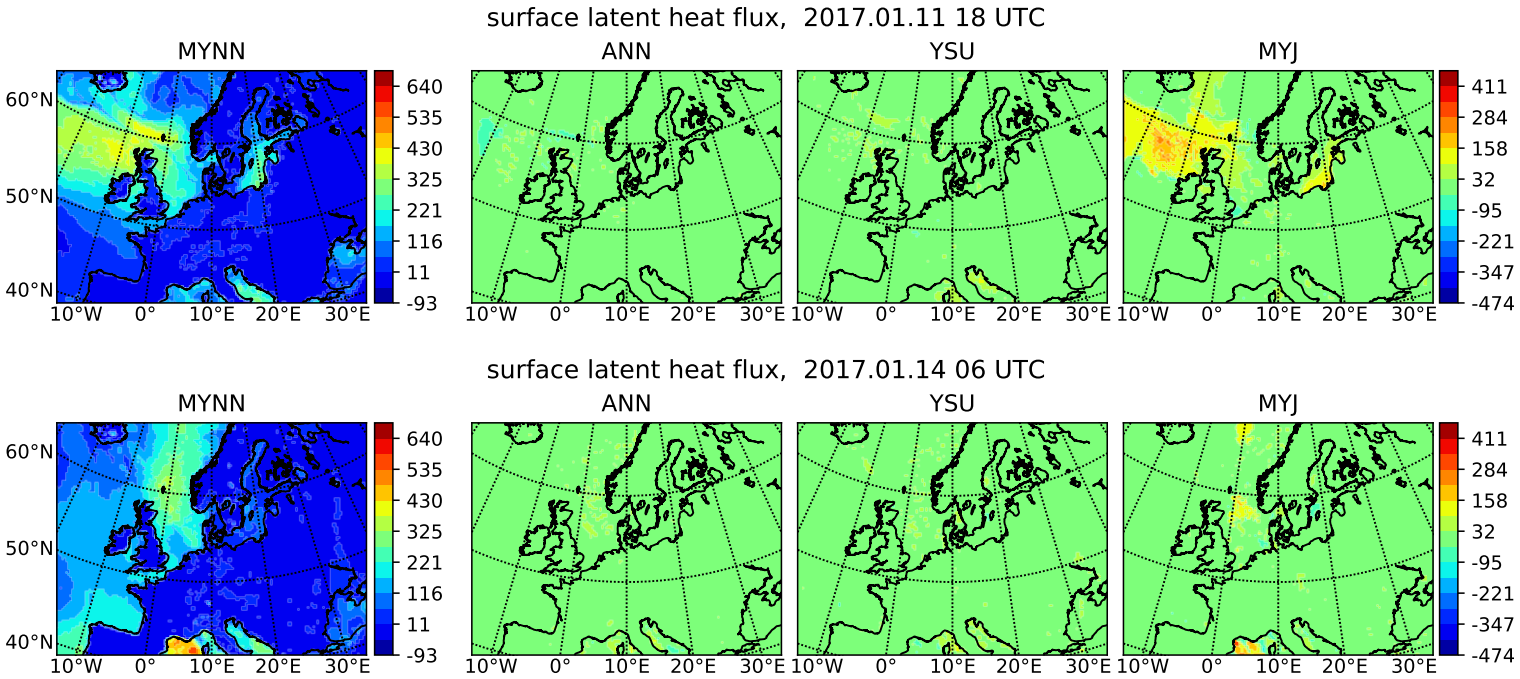


Figure 5.8: Predictions of surface sensible latent flux after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure 5.7.

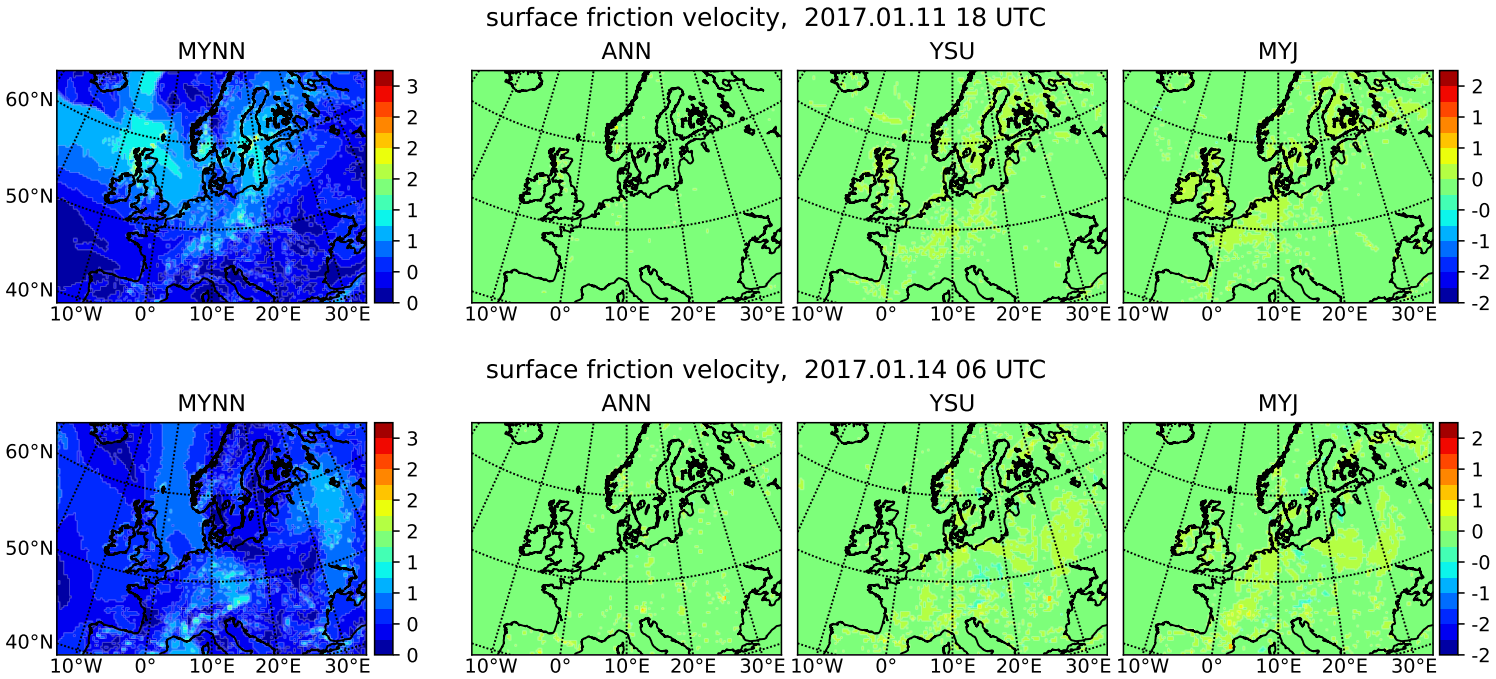


Figure 5.9: Predictions of surface friction velocity after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure 5.7.

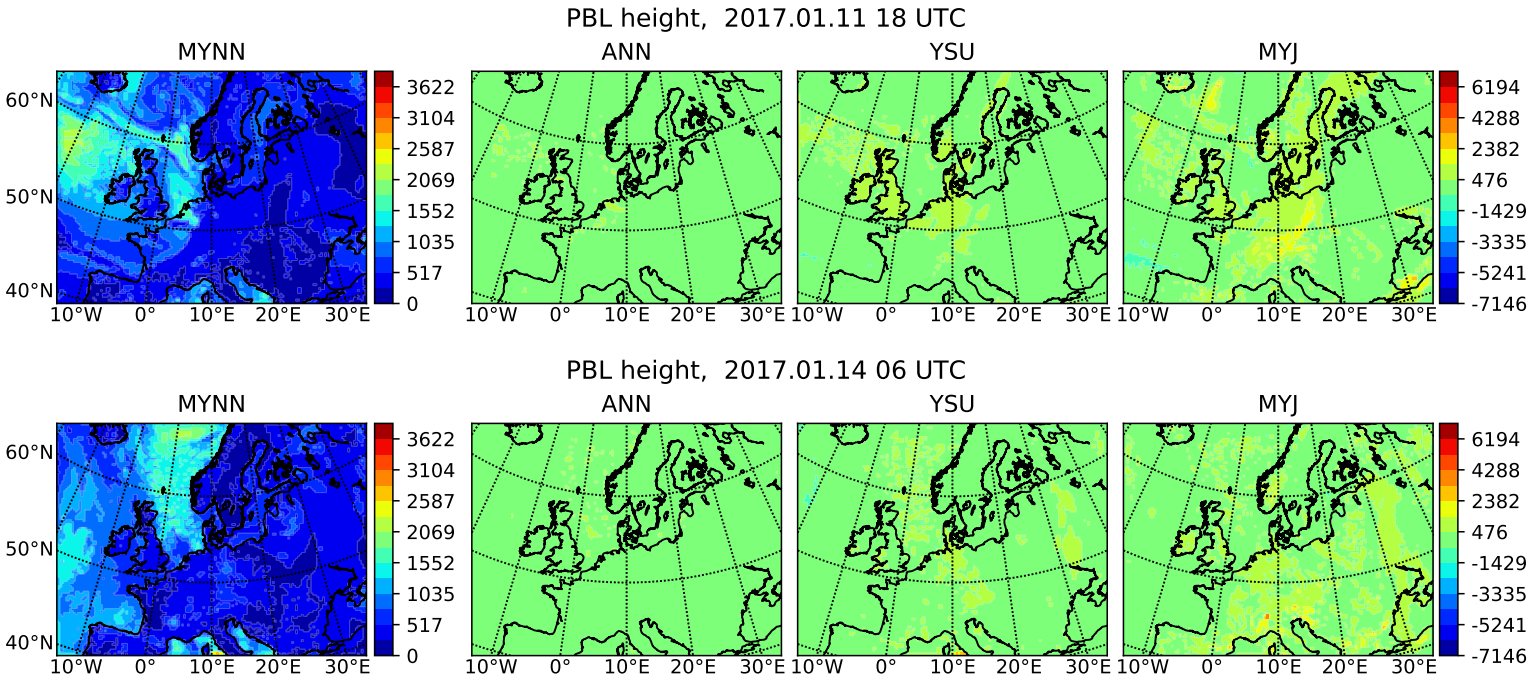


Figure 5.10: Predictions of *pblh* after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure 5.7.

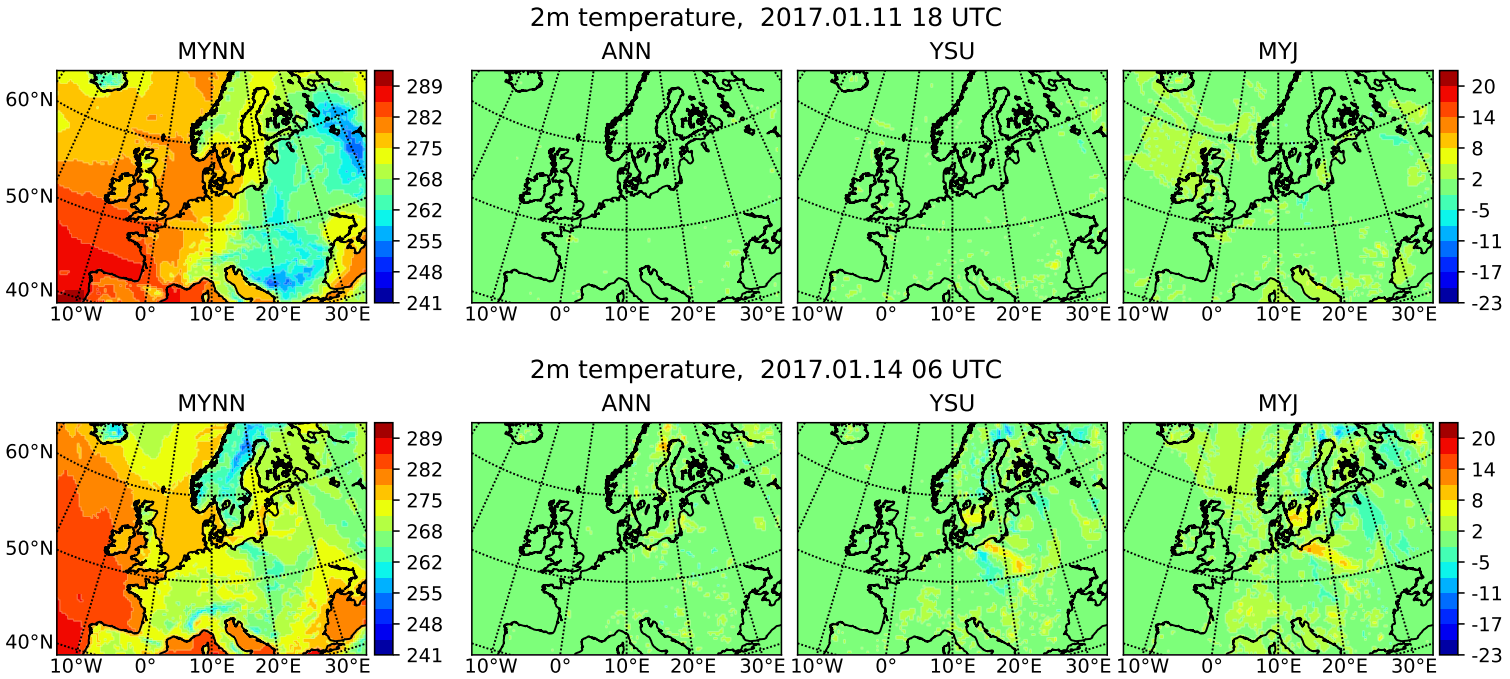


Figure 5.11: Predictions of 2m temperature after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure 5.7.

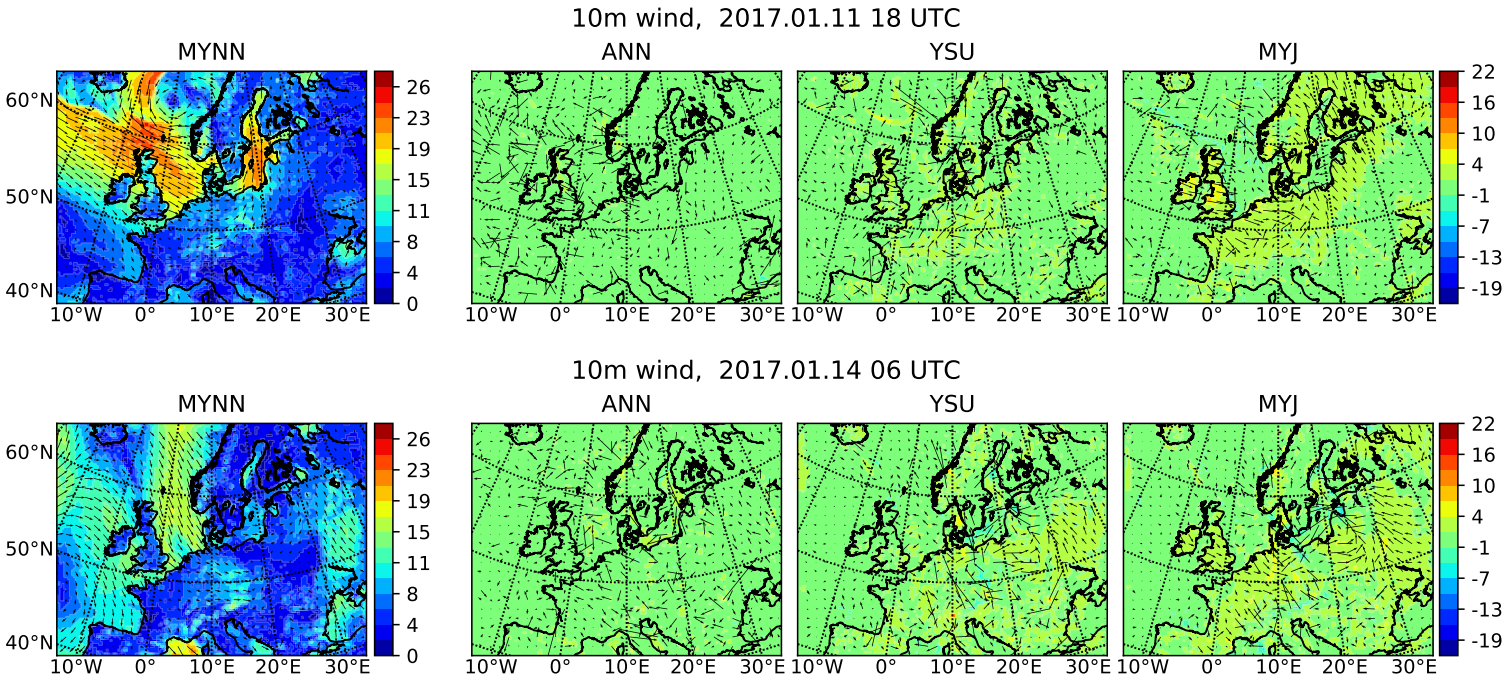


Figure 5.12: Predictions of 10m wind after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure 5.7.

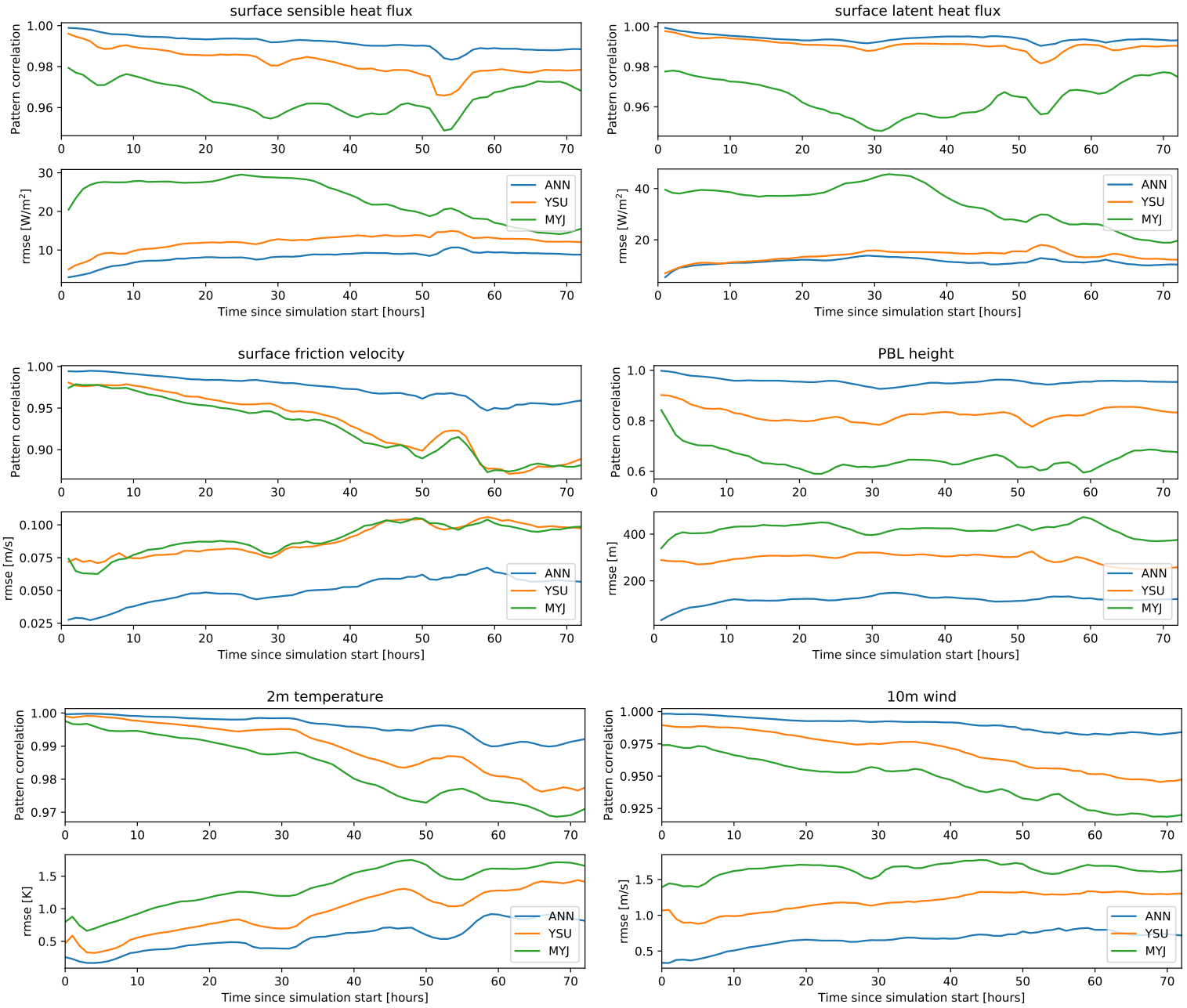


Figure 5.13: The plots in show the root mean square error, *rmse* and the pattern correlation, r as function of number of hours since the initial time. The titles indicate for which variable the measures are computed, and the legends show for which PBL scheme. Each title applies to the two plots below, showing r and *rmse*, respectively. The plots in this figure are based on the simulations initiated at 2017.01.12 06 UTC.

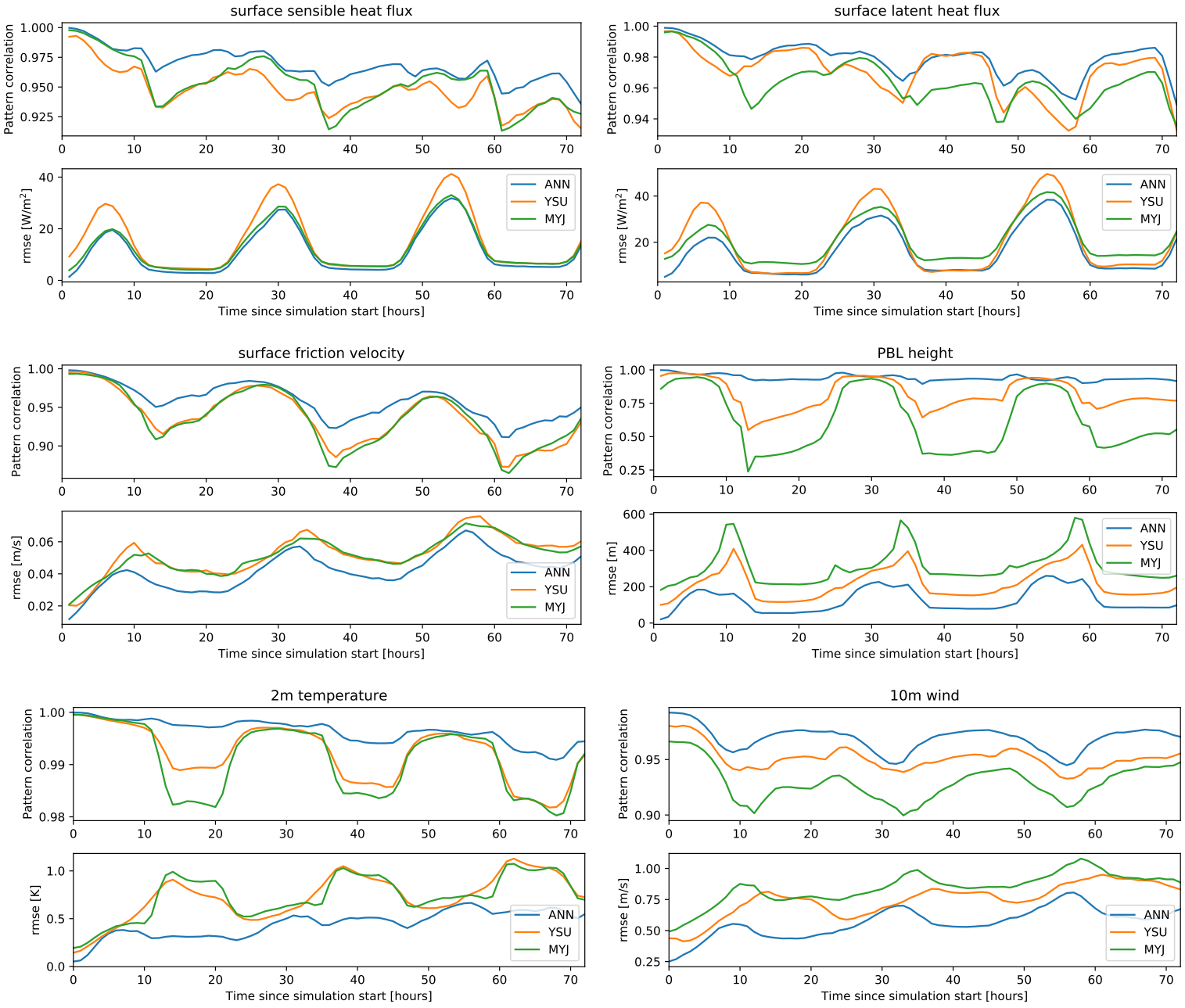


Figure 5.14: Similar to 5.13, except that the plots are based on the simulations initiated at 2018.08.02 06 UTC.

Since the PBL scheme is used to compute the turbulent fluxes throughout the atmosphere, it is not enough to only consider the surface energy balance and the surface temperature and wind fields. The *pblh* depend on both the structure of the TKE-profile and the Θ_v-profile. For details, the reader is referred to the source code, available on the official WRF GitHub-page [22]. Thus, the plots of the *pblh* anomaly in Figure 5.10 indicate that the ANN scheme captures the overall features of the atmospheric structure within the boundary layer. This is also supported by Figure E.6 in Appendix E and the plots of $rmse$ and r as function of time in Figure 5.13 and 5.14.

In addition, we will examine two examples of profiles of potential temperature Θ and wind speed. The Figures 5.16 and 5.18 show the evolution of these profiles during the first 12 hours of the simulation. Figure 5.16 shows an example with statically stable surface conditions from the first simulation (winter), while Figure 5.18 shows an example with statically unstable surface conditions from the second simulation (summer). The examples are selected based on the average value of the surface sensible heat flux Q_0 (predicted by the MYNN scheme). The statically stable example is the one with the lowest average value of Q_0 during these first 12 hours, and opposite for the statically unstable example. Predictions of all four PBL schemes are shown.

As discussed at the end of the previous Chapter, no final conclusion can be based on two examples, especially when these two examples are the most extreme cases. However, we can still learn much from this qualitative comparison. Especially, because it can be difficult to quantify features such as structure of temperature and wind profiles. In addition, a few randomly selected examples are shown in Appendix D.

Since the two examples are selected based on the values of surface sensible heat flux Q_0 the Figures 5.15 and 5.17 show plots of Q_0 valid at the same time steps as the profiles (except at the initial time, since this is not an output from WRF).

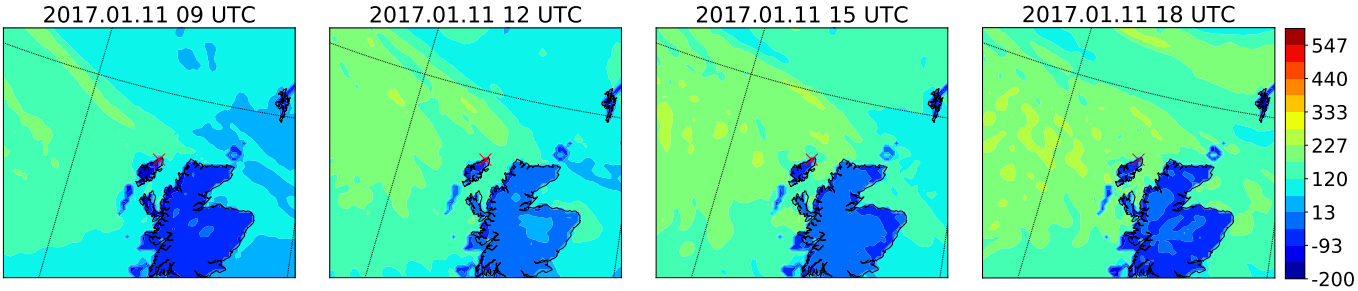


Figure 5.15: Surface sensible heat fluxes for the location of the profiles shown in Figure 5.16. The plots are based on the simulation initiated at 2017.01.11 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles, an island in north western Scotland, is marked with the red $\times$.

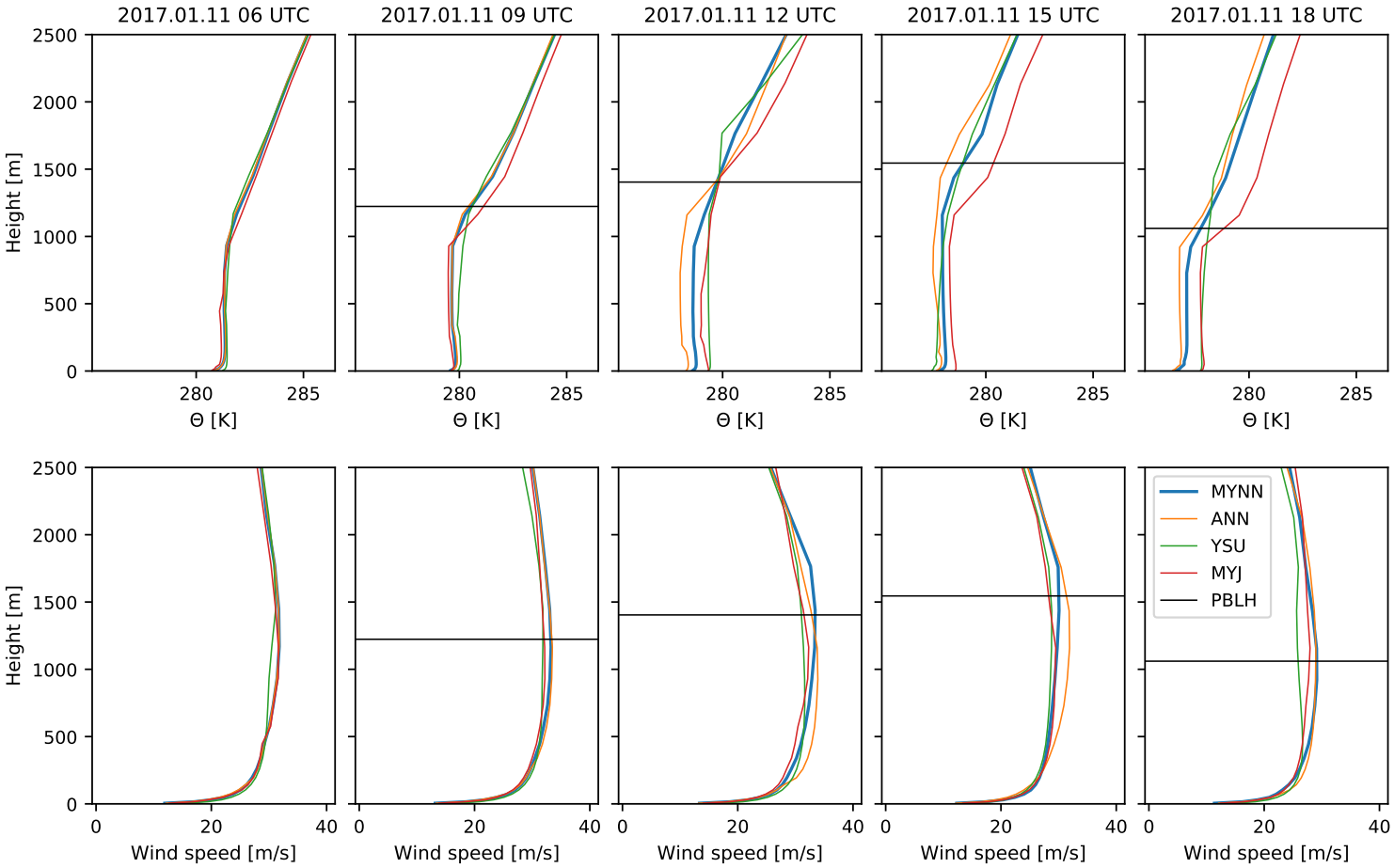


Figure 5.16: Profiles of Θ and wind speed for the location shown in Figure 5.15. This is the location with the lowest average value of surface sensible heat flux, Q_0, during the first 12 hours of simulation. The profiles are from the simulation initiated at 2017.01.11 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

From Figure 5.16, we see that the schemes do not predict identical values, but the structures of the profiles are similar for all four schemes. In this example, although the surface sensible heat flux is small, the Θ profile is almost neutral throughout a large part of the lower atmosphere. The boundary layer even grows until 15 UTC, suggesting that substantial turbulent mixing is present, probably induced by a combination the wind shear near the surface and rough surface topography.

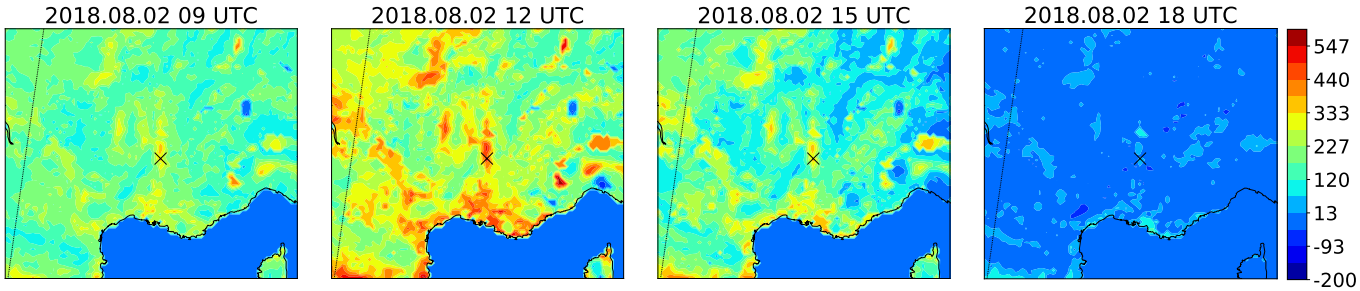


Figure 5.17: Surface sensible heat fluxes for the location of the profiles shown in Figure 5.18. The plots are based on the simulation initiated at 2018.08.02 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles, somewhere in southern France, is marked with the black ×.

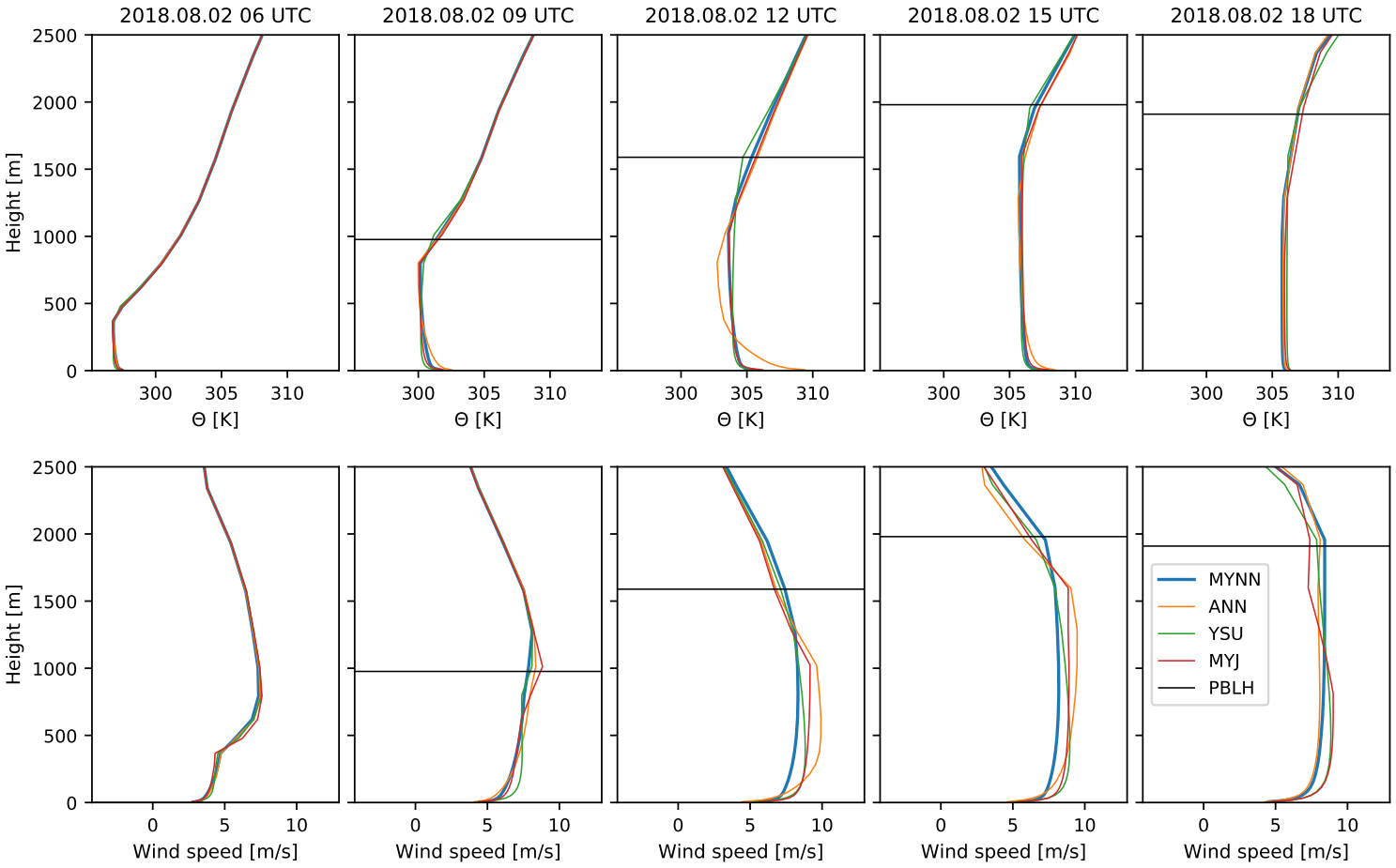


Figure 5.18: Profiles of Θ and wind speed for the location shown in Figure 5.17. This is the location with the highest average value of surface sensible heat flux, Q_0, during the first 12 hours of simulation. The profiles are from the simulation initiated at 2018.08.02 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

From Figure 5.18, we see that the Θ profile predicted by the ANN scheme behaves a somewhat differently compared the remaining schemes around 9-15 UTC. Whereas the other PBL schemes produce a well-mixed almost neutral profile, the ANN scheme produces a Θ profile with a quite different characteristic. With very high surface sensible heat fluxes, one would expect the boundary layer to be well-mixed with respect to Θ, so the behavior of the ANN scheme does not seem physically plausible. The wind speed profile produced by the ANN scheme, on the other hand, does not have a particularly different structure than the remaining wind speed profiles.

This example indicates that the ANN scheme might underestimate the turbulent thermal diffusivity for extremely statically unstable cases. This means that the turbulent mixing will be too small, which again means that the negative gradient will be enhanced as heating continues to occur at the surface. The negative Θ gradient above the surface has the largest magnitude at 12 UTC, after which the profile starts to "straighten out" and look more and more like the other profiles. This might be explained either by the decreasing surface sensible heat flux, or by the "too large" negative gradients that extend $3-400$m into the boundary layer, which will likely result in larger mixing, or possibly a combination of the two. Thus, this behavior of the ANN scheme seems to affect the Θ profiles temporarily (while large surface sensible heat flux occurs), while it does not seem to affect the overall surface energy balance.

Nothing certain can be concluded based on this one example, but one could suspect that the ANN scheme generally has difficulties predicting the *deep mixing* associated with large surface sensible heat flux. The extra examples shown in the Figures E.9-E.28 in Appendix D indicate that the ANN scheme generally produce Θ and wind speed profiles similar those of the other PBL schemes. This suggests that the behavior seen in Figure 5.18 could be a "rare" example. If this is the case, a plausible explanation is that the extremely high values of Q_0 are outside the variable distribution for Q_0 in the training dataset. We will return to this discussion in Chapter 6.

5.5 Comparing computational efficiency

In this section, we compare the efficiency of the four PBL schemes from the previous section. For this purpose, we use the profiling tool gprof [24], which measures the time spend inside each subroutine during execution of the code. For each PBL scheme, a 30-minute simulation is performed for the small model domain, i.e. the domain used for the generation of training data. Since the domain has $200 \times 150 = 3 \cdot 10^4$ horizontal grid points, the PBL scheme is called $9 \cdot 10^5$ times, which should be sufficient to obtain a realistic estimate of the computation time.

In Table 5.1, we show both the actual time spend on the PBL scheme, and what percentage of the total run time this corresponds to. As described in Section 5.2, two different versions of the ANN scheme are implemented: the first uses Fortran's intrinsic `matmul()` function, and the sec-

ond uses the `sgemm()` function. Table 5.1 show the computation time for both implementations of the ANN scheme and for the three other PBL schemes.

Table 5.1: Results of profiling 30-minute simulations with each of the PBL schemes.

Scheme	Time spend on the PBL scheme	% of total run time
MYNN	19.50 s	16.9%
ANN (matmul)	10.89 s	10.1%
ANN (sgemm)	9.49 s	9.1%
YSU	3.61 s	3.7%
MYJ	6.44 s	6.1%

From Table 5.1, we see that the ANN scheme is about twice as fast as the MYNN scheme, slightly faster with `sgemm()` than with `matmul()`. However, both the MYJ scheme and the YSU scheme are significantly faster.

Further, Table 5.2 show the computation times for selected subroutines of the MYNN scheme and the two implementations of the ANN scheme.

Table 5.2: Results of profiling 30-minute simulations with each of the PBL schemes. *Condensations scheme* denotes the partial-condensation scheme used to compute β_θ and β_q, and the category *Other* includes computation of *pblh* and assignment of 3D-grid variables to local 1D variables.

Scheme	Subroutine	Time spend on the PBL scheme	% of total run time
MYNN	Computing K_m, K_h and L	5.53 s	4.8%
	Condensation scheme	4.31 s	3.7%
	Computing tendencies	4.09 s	3.6%
	Solving TKE equation	0.70 s	0.6%
	Other	4.87 s	4.2%
ANN (matmul)	Neural network prediction	1.58 s	1.5%
	Computing tendencies	3.77 s	3.5%
	Solving TKE equation	0.55 s	0.5%
	Other	4.99 s	4.6%
ANN (sgemm)	Neural network prediction	0.85 s	0.8%
	Computing tendencies	3.72 s	3.6%
	Solving TKE equation	0.64 s	0.6%
	Other	4.28 s	4.1%

From Table 5.2, we see that the ANN scheme using `sgemm()` is almost twice as fast as the ANN scheme using `matmul()`. As described in Section 5.1, the neural networks substitute both the subroutine computing the diffusivities and the turbulent scale and the subroutine related to the partial-condensation scheme. Together, these two subroutines account for 9.84 s, about half of

the total computation time of the MYNN scheme. Thus, the computational cost is effectively reduced by more than a factor of 10 (when using `sgemm()`).

Notice that there are differences in the computation time of the remaining subroutines as well. The MYNN scheme is generally a bit slower, which may be partly explained by minor changes made to other parts of the code[2]. However, there is even a small difference in the computation time of identical subroutines of the two ANN schemes. This suggests that the estimates of the computation times have some uncertainty, which, however, has not been quantified.

Despite this uncertainty, there is no question that the subroutine computing the predictions of the neural networks only account for a small fraction of the computation time. Since, the ANN scheme is still three times slower than the YSU scheme, this cannot be explained by the neural network related subroutines alone. Instead, this may be due to overall different structuring of the code and possibly lack of optimization of the MYNN module. Consider the fact that computing the tendencies and the category "other" together account for 8.00-8.76 s of the computation time (for the two ANN schemes). The category "other" represent the prediction of the *pblh* (which is negligible) and assignment of variables from the 3D domain grid to local 1D variables. For comparison, the total computation time of the YSU and MYJ schemes are 3.61 s and 6.44 s, respectively. These schemes use the same 3D variables as input, and they also compute both the tendencies and the *pblh*. Thus, this strongly indicates that, at least parts of the MYNN code, could be optimized substantially. We speculate that if the neural network related subroutines were implemented in a different, more optimized code, the computational cost of the ANN scheme would be comparable to the cost of the fast PBL scheme options in WRF, such as the YSU scheme.

[2]These minor changes mostly consist of removing un-used variables and `if` statements.

Chapter 6

Discussion and conclusion

This Chapter will first briefly summarize some of the key points of the first few chapters. Then, in the Sections 6.1 and 6.2, we go through the findings of the Chapters 4 and 5 and follow up on the important discussions we encountered along the way. The Chapter will be wrapped up in Section 6.3 with an overall conclusion and suggestions for further work in this field.

The aim of the thesis was to examine the possibilities of using neural networks for PBL turbulence parameterization. In Chapter 1, we saw how some traditional turbulence parameterization models, exemplified by the Mellor-Yamanda and the MYNN models, make use of additional prognostic equations to parameterize the turbulent fluxes to second order accuracy. The MYNN2.5 and MYNN3 models introduce 1 and 4 extra prognostic equations, respectively. Going to second order accuracy, however, introduces third order terms that need to be parameterized, which means that even more assumptions are needed. Thus, the quality of the model depends both on the closure assumptions and the quality of the data used to tune the closure constants. This motivates the idea of using a more general machine learning based regression method such as neural networks, i.e. general in the sense that it can emulate any functional form. For this type of model, only the input and output variables need to be specified, which effectively eliminates the need for many of the closure assumptions applied in traditional turbulence modeling.

One challenge when using neural networks is that they may require large amounts of data, and it is essential that the dataset covers all scenarios that can occur in the physical system. However, data from high resolution models such as large-eddy simulations is computationally expensive to acquire, and therefore, as a first attempt, it was natural to use a computationally cheaper alternative. Thus, the training dataset was generated from six weather simulations performed with WRF using the MYNN2.5 scheme. This allows us to gain experience on the subject without the need of expensive data generation.

6.1 Development of the ANN scheme

In Chapter 4, we discussed different approaches to constructing a neural network based model. We motivated why the diffusivities are appropriate output variables due to considerations regarding numerical stability. This of course imposes the assumption that the effects of subgrid-scale dynamics reduce to a diffusion-like phenomenon. Although this assumption might not be entirely correct, it is a very well-established parameterization method and has an intuitive physical interpretation, see Section 1.3.1. Further, the assumption implies that the turbulent mixing of all physical quantities depends on the same two diffusion coefficients, which are related mainly to the static stability, the gradients of the wind field, and the potential presence of turbulence. On the other hand, the turbulent flux for each physical quantity depends on the specific vertical profile of that variable. Thus, if the fluxes were the direct output of the neural network, it would require a much more diverse dataset.

Several different variables were considered as potential inputs of the neural networks, and we found that, in addition to q (square root of twice the TKE) and the variables B and S (related to buoyance and shear), following variables contributed positively to the performance of the neural networks: the height above the surface z, the surface sensible heat flux Q_0, the friction velocity u_*, the potential temperature Θ, and the mixing ratios for water vapor and liquid water, Q_v and Q_c. Except for the surface fluxes, these are all defined in the same model level as the output variables. We saw that the neural networks tended to overfit, especially when the latter three variables were used as input variables. As we discussed briefly in Section 4.3, the primary justification for adding these variables is to be able to account for the buoyancy effects related to condensation and evaporation. Thus, these variables are only relevant as inputs when phase changes occur. The hypothesis is, therefore, that the training dataset is too small to correctly represent all possible weather situations[1], and therefore the neural networks instead falsely correlate certain boundary layer dynamics with certain values of Θ, Q_v and Q_c.

To avoid overfitting, a simpler model was developed as well, where these three input variables were omitted. Although some of the remaining variables may contain redundant information as well, e.g. the height above the surface and the surface fluxes of heat and momentum are only relevant close to the surface, these variables are more essential for the boundary layer dynamics and are therefore not omitted.

In addition, Chapter 4 presented several important findings: the most important feature was the logarithmic scaling, which solves the problem of variable distributions covering several orders of magnitude. We saw that using different models for statically stable/unstable samples also

[1] It should be redundant to justify this statement. Obviously more than six 24-hour weather simulations are necessary to fully describe the variability of this system.

improved the results quite significantly, and of course this was necessary for being able to apply the logarithmic scaling to the buoyancy parameter B. Further, the logarithmic scaling gave a new choice regarding the loss function: either to apply the loss function directly to the outputs of the neural networks or first compute the physical values and then apply the loss function. We saw that applying the *mae* loss function to the physical values gave the best results. Applying the *mse* loss function directly to the log-scaled values also gave reasonable results, while applying the *mse* loss function the physical values gave significantly worse results.

Further, we showed that the choices of the hyperparameters batch size and activation function seem to have little or no impact for the model performance. The ReLU and leakyReLU did give slightly better results than the sigmoid and tanh activations and therefore seem to be preferable. However, the differences in performance were small compared to the effects of the features described above.

Regarding the model size, it is difficult to conclude anything based this study. From the Figures 4.12 and 4.16, we got the impression that the model size was quite important for the model performance, since the loss values decreased significantly when increasing the number of layer and nodes in the networks. However, when the models were implemented in WRF, we found that the best performing model, was the one using the smallest networks, which had performed worst when evaluated off-line on the "independent" test dataset. As already discussed, this of course indicates that the independent test dataset was not really independent after all (and the same is true for the validation dataset). Recall that the "best" model from every training was selected as the model that performed best on the validation dataset. If this dataset was truly independent this should ensure that the selected model is the one that generalizes best, but we cannot assume that in our case. In other words, we cannot be certain that model, which was implemented and evaluated, was the optimal model. On the contrary, it is quite likely that a better model exists.

In hindsight, the solution to this issue seems embarrassingly simple: the training, validation and test datasets should be sampled from different simulations using different initial times, and maybe even different model domains. Although having pointed out this weakness of the method used, it must be stressed that the other findings should still be generally applicable. The logarithmic scaling of the input and output variables had a clear theoretical motivation and solved a problem unrelated to whether the training and validation datasets are mutually independent. The choice of loss function determines which metric is used when comparing the predictions to the target values, and therefore the optimal loss function should depend solely on the problem the networks are solving.

6.2 Evaluation of the ANN scheme

We have already discussed the overfitting of the models and, therefore, here the focus will be on the evaluation of the best ANN scheme. We saw in Section 5.4 that the ANN scheme gave results very similar to the existing PBL scheme options in WRF. For the surface fluxes, 2m temperature, 10m wind and $pblh$, it gave results significantly closer to the MYNN scheme than the MYJ and the YSU schemes. This can be seen from the Figures 5.7-5.14 as well as the Figures E.3-E.8 in Appendix E. Especially interesting are the Figures 5.13 and 5.14 showing the $rmse$ and r values for all 6 variables as function of time for the ANN, YSU and MYJ schemes. In both simulations, throughout the 72 hours, the ANN scheme has both the lowest $rmse$ values and highest r values for all variables.

When we examined the profiles of Θ and wind speed produced by the four PBL schemes, we saw indications that the ANN scheme is not capable of predicting the deep mixing associated with very high surface sensible heat flux Q_0. However, recall that the domain used for evaluating the ANN scheme is significantly larger than the domain used for generation of the training dataset. The idea behind this was to test, how well the ANN scheme could be generalized to weather conditions in different climate zones. The specific profile in this example was located in Southern France, far south of the boundary of the domain used for generating the training dataset. This may very well explain why the ANN scheme underestimates the thermal diffusivity when encountering extreme Q_0 values: the value is simply outside the probability distribution of Q_0 in the training dataset. In the training dataset, the largest value of Q_0 does not exceed ~ 260 W/m^2, whereas Q_0 in this example exceeds at least ~ 400 W/m^2 when at its maximum, see Figure 5.17. Thus, the neural networks extrapolate, and we cannot expect the result to be reliable. This, however, also suggests that including examples with higher Q_0 values in the training dataset will solve the problem.

The true physical equations have "embedded" negative feedbacks, which prevent unphysical "run-away" effects, but this is not the case for a neural network based model. Since neural networks are highly nonlinear functions, the consequences of extrapolation are unpredictable, and one cannot expect physically realistic behavior. Therefore, it is actually surprising that the predictions of the ANN scheme are still quite accurate, and it indicates that the ANN scheme is robust and do not have problems with positive feedback loops. The opposite seems to be the case with the ANN scheme using Θ, Q_v and Q_c as additional inputs, as we saw from the comparison of the three ANN schemes, e.g. Figure 5.5.

Finally, we profiled two different ANN scheme implementations, using the `matmul()` and `sgemm()` functions for the matrix products, respectively. The fastest of these resulted in a PBL scheme

that was about twice as fast as the MYNN scheme. However, comparing only the neural network related subroutines to the subroutines they have substituted, the computational cost is reduced by more than a factor of 10. Further, in Section 5.5, we argued that if the neural networks were implemented in a more optimized code, the ANN scheme might be comparable in speed with the faster PBL scheme options in WRF, such as the YSU scheme.

6.3 Conclusion and outlook

This thesis has demonstrated that it is possible to create an accurate and robust turbulence closure model using neural networks to compute the turbulent diffusivities. Further, it was shown that this type of model has the potential of being an efficient alternative to expensive second order PBL schemes such as the MYNN scheme while retaining the second order accuracy.

Further, we saw that if one wishes to exploit the information in the variables T, Θ, Q_c and Q_v, the results indicate that a larger dataset is needed. However, it seems that this introduces a risk of a positive feedback loops, if the neural networks start extrapolating. Thus, it might be difficult to create a robust neural network based model when including these variables as inputs.

Having established a method for constructing neural network based turbulence parameterization models, this thesis has paved the way for future works in this field of research. As an obvious and relatively easy next step, the final part of the model optimization could be repeated using more data, and sampling the training, validation and test datasets from different simulations to ensure a higher degree of mutual independence between the datasets. In addition, to avoid unphysical behavior related to extrapolation, the dataset should include more extreme values for the surface fluxes.

Finally, a natural next step would be to train the neural networks on a training dataset generated from more accurate high resolution data from, e.g., large-eddy simulations. Since fewer closure assumptions are needed when using this type of model, this could potentially improve the quality of turbulence parameterization in the future.

Bibliography

[1] Wyngaard, John C., *Turbulence in the Atmosphere.* Cambridge University Press, 2010.

[2] Lautrup, B., *Physics of Continuous Matter, Second Edition.* CRC Press, 2011.

[3] Holton, J. R. and Hakim, G. J.: *Dynamic Meteorology, 5th edition*, Elsevier, 2013.

[4] Wallace, John M., and Peter Victor Hobbs. *Atmospheric Science: An Introductory Survey.* Amsterdam: Elsevier Academic Press, 2006.

[5] Tølløse, Kasper S.: *Parameterization of boundary layer turbulence in NWP*, Report for project outside course scope. Unpublished, contact author for details: qrm173@alumni.ku.dk

[6] Crank, J., Nicolson, P., *A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type*, Proc. Camb. Phil. Soc. 43 (1): 50–67, 1947. https://doi:10.1017/S0305004100023197

[7] Mellor, George L., *Analytic prediction of the properties of stratified planetary surface layers.* Journal of Atmospheric Sciences, 30, 1061-1069, 1973. https://doi.org/10.1175/1520-0469(1973)030<1061:APOTPO>2.0.CO;2

[8] Mellor, G. L. and T. Yamada, *A hierarchy of turbulence closure models for planetary boundary layers.* Journal of Atmospheric Sciences, 31, 1791–1806, 1974. https://doi.org/10.1175/1520-0469(1974)031<1791:AHOTCM>2.0.CO;2

[9] Mellor, G. L. and T. Yamada, *Development of a turbulence closure model for geophysical fluid problems.* Reviews of Geophysics and Spacs Physics, 20, 851–875, 1982. https://doi.org/10.1029/RG020i004p00851

[10] Nakanishi, M., *Improvement of the Mellor–Yamada turbulence closure model based on large eddy simulation data.* Bound. Layer Meteor., 99, 349–378, 2001. https://doi.org/10.1023/A:1018915827400

[11] Nakanishi, M. and Niino, H., *An improved Mellor–Yamada level-3 model with condensation physics: Its design and verification.* Bound. Layer Meteor., 112, 1–31, 2004.
https://doi.org/10.1023/B:BOUN.0000020164.04146.98

[12] Nakanishi, M. and Niino, H., *An improved Mellor–Yamada level-3 model: Its numerical stability and application to a regional prediction of advection fog.* Bound. Layer Meteor., 119, 397–407, 2006.
https://doi.org/10.1007/s10546-005-9030-8

[13] Nakanishi, M. and Niino, H., *Development of an Improved Turbulence Closure Model for the Atmospheric Boundary Layer.* Journal of the Meteorological Society of Japan, Vol. 87, No. 5, 895-912, 2009.
https://doi.org/10.2151/jmsj.87.895

[14] Sommeria, G. and Deardorff, J.W., *Subgrid-Scale Condensation in Models of Nonprecipitating Clouds.* Journal of the Atmospheric Sciences, vol. 34, 344-355, 1976.
https://doi.org/10.1175/1520-0469(1977)034<0344:SSCIMO>2.0.CO;2

[15] Hong, S.-Y., and Noh, Y., *A New Vertical Diffusion Package with an Explicit Treatment of Entrainment Processes.* Monthly Weather Review, vol. 134, 2318-2341, 2006.
https://doi.org/10.1175/MWR3199.1

[16] Janjić, Z.I., *Nonsingular implementation of the Mellor-Yamada level 2.5 scheme in the NCEP Meso model.* NCEP Office Note. No.437 (61 pp), 2002.

[17] Banks, R.F., Tiana-Alsina, J., Baldasano, J.M., Rocadenbosch, F., Papayannis, A., Solomos, S. and Tzanis, C.G., *Sensitivity of boundary-layer variables to PBL schemes in the WRF model based on surface meteorological observations, lidar, and radiosondes during the HygrA-CD campaign.* Atmospheric Research vol. 176–177 (2016) 185–201, 2016.
https://doi.org/10.1016/j.atmosres.2016.02.024

[18] Fekih, A. and Mohamed, A., *Evaluation of the WRF model on simulating the vertical structure and diurnal cycle of the atmospheric boundary layer over Bordj Badji Mokhtar (southwestern Algeria).* Journal of King Saud University – Science, vol. 31, iss. 4, 602-611, 2019.
https://doi.org/10.1016/j.jksus.2017.12.004

[19] Tyagi, B., Magliulo, V., Finardi, S., Gasbarra, D., Carlucci, P., Toscano, P., Zaldei, A., Riccio, A., Calori, G., D'Allura, A. and Gioli, B., *Performance Analysis of Planetary Boundary Layer Parameterization Schemes in WRF Modeling Set Up over Southern Italy.* Atmosphere,

vol. 9, iss. 7, 2018.
https://doi.org/10.3390/atmos9070272

[20] *Weather Research and Forecasting Model - ARW Version 4 Modeling System User's Guide*, January 2019.
https://www2.mmm.ucar.edu/wrf/users/downloads.html

[21] Skamarock, W.C., Klemp, J.B., Dudhia, J., Gill, D.O., Liu, Z., Berner, J., Wang, W., Powers, J.G., Duda, M.G., Barker, D.M. and Huang, X.-Y., *A Description of the Advanced Research WRF Model Version 4.* NCAR Technical Note No.556, March 2019.
https://doi:10.5065/1dfh-6p97

[22] Gill, Dave et al., *The official repository for the Weather Research and Forecasting (WRF) model.*
https://github.com/wrf-model/WRF

[23] Xianyi, Z., Kroeker, M., *OpenBLAS - An optimized BLAS library.*
https://www.openblas.net

[24] Fenlason, Jay, *GNU gprof.*
https://sourceware.org/binutils/docs/gprof

[25] National Centers for Environmental Prediction/National Weather Service/NOAA/U.S. Department of Commerce, *NCEP GDAS/FNL 0.25 Degree Global Tropospheric Analyses and Forecast Grids*, Research Data Archive at the National Center for Atmospheric Research, Computational and Information Systems Laboratory, Boulder, CO, 2015.
https://doi.org/10.5065/D65Q4T4Z

[26] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning.*, MIT Press, 2016.
http://www.deeplearningbook.org

[27] Bishop, Christopher M., *Pattern Recognition and Machine Learning.* Springer, First ed., 2006.

[28] Mehta, P., Wang, C.-H., Day, A.G.R. and Richardson, C., *A high-bias, low-variance introduction to Machine Learning for physicists.*
https://arXiv:1803.08823

[29] Kingma, D.P. and Lei Ba, J., *Adam: a Method for Stochastic Optimization.*, International Conference on Learning Representations, 2015.
https://arxiv.org/abs/1412.6980

[30] Smith, L.N., *Cyclical Learning Rates for Training Neural Networks.* IEEE Winter Conference on Applications of Computer Vision, WACV 2017.
https://arxiv.org/abs/1506.01186

[31] Snoek, J., Larochelle, H. and Adams, R.P., *Practical Bayesian Optimization of Machine Learning Algorithms.* 2012. https://arxiv.org/abs/1206.2944.

[32] Abiodun, O.I., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, N.A. and Arshad, H., *State-of-the-art in artificial neural network applications: A survey*
https://doi.org/10.1016/j.heliyon.2018.e00938

[33] Python Core Team. *Python: A dynamic, open source programming language. Python Software Foundation*, 2019.
https://www.python.org/

[34] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* 2015.
https://www.tensorflow.org/

[35] Chollet, F. and Others, *Keras.* 2015.
https://keras.io

[36] Ling, J., Kurzawski, A. and Templeton, J., *Reynolds Averaged Turbulence Modeling using Deep Neural Networks with Embedded Invariance.* Journal of Fluid Mechanichs, vol. 807, 155-166, 2016.
https://doi.org/10.1017/jfm.2016.615

[37] Li, Wei, *Stable Boundary Layer Height Parameterization: Learning from Artificial Neural Networks.* Atmospheric and Climate Sciences, vol. 3, 523-531, 2013.
http://dx.doi.org/10.4236/acs.2013.34055

[38] O'Gorman, Paul A. and Dwyer, John G., *Using Machine Learning to Parameterize Moist Convection: Potential for Modeling of Climate, Climate Change, and Extreme Events.* Journal of Advances in Modeling Earth Systems, vl. 10, 2548–2563, 2018.
https://doi.org/10.1029/2018MS001351

[39] Rasp, Stephan, Pritchard, Michael S. and Gentine, Pierre, *Deep learning to represent subgrid processes in climate models.* Proceedings of the National Academy of Sciences, vol. 115 no. 39, 9684–9689, 2018.
https://doi.org/10.1073/pnas.1810286115

[40] Ukkonen, Peter and Makela, Antti, *Evaluation of Machine Learning Classifiers for Predicting Deep Convection.* Journal of Advances in Modeling Earth Systems, vol. 11, 2019. https://doi.org/10.1029/2018MS001561

Appendix A

Namelist examples

Examples of the files `Namelist.wps` and `Namelist.input`. The examples show the configurations for a 30-hour simulation, starting at 2017.01.06 06 UTC. Clarifying comments have been added where appropriate.

`Namelist.wps`:

```
&share
 wrf_core = 'ARW',
 max_dom = 1,
 start_date = '2017-01-06_06:00:00',   ! specifying the initial time
 end_date   = '2017-01-07_12:00:00',   ! specifying the end time
 interval_seconds = 21600              ! specifying the interval between boundary condition files
 io_form_geogrid = 2,
/

&geogrid
 parent_id         =   1,
 parent_grid_ratio =   1,
 i_parent_start    =   1,              ! specifying the first index for staggered dimension in x
 j_parent_start    =   1,              ! specifying the first index for staggered dimension in y
 e_we              = 201,              ! specifying the last index for staggered dimension in x
 e_sn              = 151,              ! specifying the last index for staggered dimension in y

 geog_data_res = 'maxsnowalb_ncep+albedo_ncep+default',
 dx = 10000,
 dy = 10000,
 map_proj = 'lambert',                 ! specifying the type of map projection
 ref_lat   = 57,                       ! reference latitude
 ref_lon   = 9,                        ! reference longitude
 truelat1  = 57,                       ! true latitude 1
 truelat2  = 57,                       ! true latitude 2
 stand_lon = 9,                        ! standard longitude
 geog_data_path = 'path_to_files'
/

&ungrib
 out_format = 'WPS',
 prefix = 'FILE',
/

&metgrid
 fg_name = 'FILE'
 io_form_metgrid = 2,
/
```

Namelist.input:

```
! first, the duration of the run and the initial time and end time are specified (must correspond to wps)
&time_control
run_days                            = 1,
run_hours                           = 6,
run_minutes                         = 0,
run_seconds                         = 0,
start_year                          = 2017,
start_month                         = 01,
start_day                           = 06,
start_hour                          = 06,
end_year                            = 2018,
end_month                           = 01,
end_day                             = 07,
end_hour                            = 12,
interval_seconds                    = 21600
input_from_file                     = .true.,
history_interval                    = 180,
frames_per_outfile                  = 1000,
restart                             = .false.,
restart_interval                    = 7200,
io_form_history                     = 2
io_form_restart                     = 2
io_form_input                       = 2
io_form_boundary                    = 2
auxhist24_outname                   = "trainingdata_d<domain>_<date>"
io_form_auxhist24                   = 2
auxhist24_interval                  = 60
iofields_filename                   = "iofields_d01.txt"
/

&domains
time_step                           = 60,       ! setting time step
time_step_fract_num                 = 0,
time_step_fract_den                 = 1,
max_dom                             = 1,
e_we                                = 201,      ! number of grid points in x (must correspond to wps)
e_sn                                = 151,      ! number of grid points in y (must correspond to wps)
e_vert                              = 41,
auto_levels_opt                     = 2         ! related to the vertical coordinate eta
max_dz                              = 1000.     ! related to the vertical coordinate eta
dzbot                               = 10.       ! related to the vertical coordinate eta
dzstretch_s                         = 1.3       ! related to the vertical coordinate eta
dzstretch_u                         = 1.1       ! related to the vertical coordinate eta
p_top_requested                     = 5000,     ! related to the vertical coordinate eta
num_metgrid_levels                  = 32,
num_metgrid_soil_levels             = 4,
dx                                  = 10000,    ! spatial resolution in horizontal direction x
dy                                  = 10000,    ! spatial resolution in horizontal direction y
grid_id                             = 1,
parent_id                           = 0,
i_parent_start                      = 1,
j_parent_start                      = 1,
parent_grid_ratio                   = 1,
parent_time_step_ratio              = 1,
feedback                            = 1,
smooth_option                       = 0
/

&physics
physics_suite                       = 'CONUS'   ! selecting the CONUS physics suite
mp_physics                          = -1,
cu_physics                          = -1,
ra_lw_physics                       = -1,
ra_sw_physics                       = -1,
sf_surface_physics                  = -1,
radt                                = 30,
bldt                                = 0,
cudt                                = 5,
icloud                              = 1,
num_land_cat                        = 21,
```

```
 sf_urban_physics                    = 0,
 sf_sfclay_physics                   = 5,        ! overwriting the surface flux scheme with MYNN
 bl_pbl_physics                      = 5,        ! overwriting the surface flux scheme with MYNN(2.5)
 bl_ann_option                       = .false.   ! option related to ANN scheme (not generally available)
 /

 &fdda
 /

 &dynamics
 hybrid_opt                          = 2,
 w_damping                           = 0,
 diff_opt                            = 1,
 km_opt                              = 4,
 diff_6th_opt                        = 0,
 diff_6th_factor                     = 0.12,
 base_temp                           = 290.
 damp_opt                            = 3,
 zdamp                               = 5000.,
 dampcoef                            = 0.2,
 khdif                               = 0,
 kvdif                               = 0,
 non_hydrostatic                     = .true.,
 moist_adv_opt                       = 1,
 scalar_adv_opt                      = 1,
 gwd_opt                             = 1,
 /

 &bdy_control
 spec_bdy_width                      = 5,
 specified                           = .true.
 /

 &grib2
 /

 &namelist_quilt
 nio_tasks_per_group = 0,
 nio_groups = 1,
 /

 ! configurations related to the DFI, must correspond to the initial time (1hour before, 1/2hour after)
 &dfi_control
 dfi_opt                             = 3
 dfi_nfilter                         = 7
 dfi_write_filtered_input            = .true.
 dfi_write_dfi_history               = .false.
 dfi_cutoff_seconds                  = 3600
 dfi_time_dim                        = 1000
 dfi_bckstop_year                    = 2017
 dfi_bckstop_month                   = 01
 dfi_bckstop_day                     = 06
 dfi_bckstop_hour                    = 05
 dfi_bckstop_minute                  = 00
 dfi_bckstop_second                  = 00
 dfi_fwdstop_year                    = 2017
 dfi_fwdstop_month                   = 01
 dfi_fwdstop_day                     = 06
 dfi_fwdstop_hour                    = 06
 dfi_fwdstop_minute                  = 30
 dfi_fwdstop_second                  = 00
 /
```

Appendix B

Simulations used for training data

This appendix shows plots of relevant plots for the simulations used for the training dataset. All six simulation run for 30 hours, where the first 6 hours is the spin-un phase, while data is sampled for the training dataset (and the validation and test datasets) from the last 24 hours. The initial times of the six simulations are: 2017.01.06 06 UTC, 2017.02.13 06 UTC, 2017.08.03 06 UTC, 2018.07.25 06 UTC, 2019.05.20 06 UTC and 2019.09.15 06 UTC.

Figure B.1 shows the surface elevation height and surface roughness to illustrate the variations in the surface conditions in the domain. The figures B.2-B.7 show plots of 2m temperature, surface sensible heat flux and surface friction velocity. The plots are pedictions after 6 hours, i.e. the first time step, where we start sampling data. Especially the surface sensible heat flux is important for the overall structure of the boundary layer.

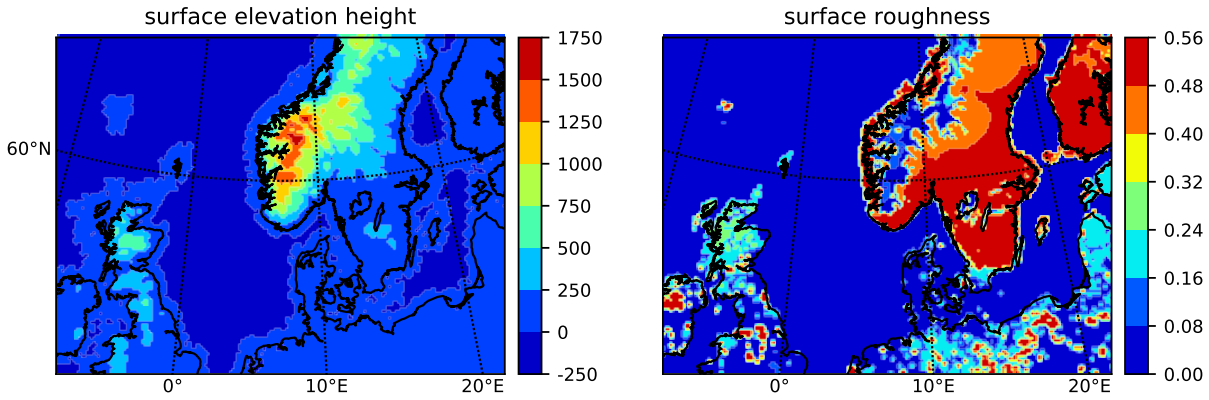


Figure B.1: Surface elevation height (left) and surface roughness (right) of the domain used for generation of training data.

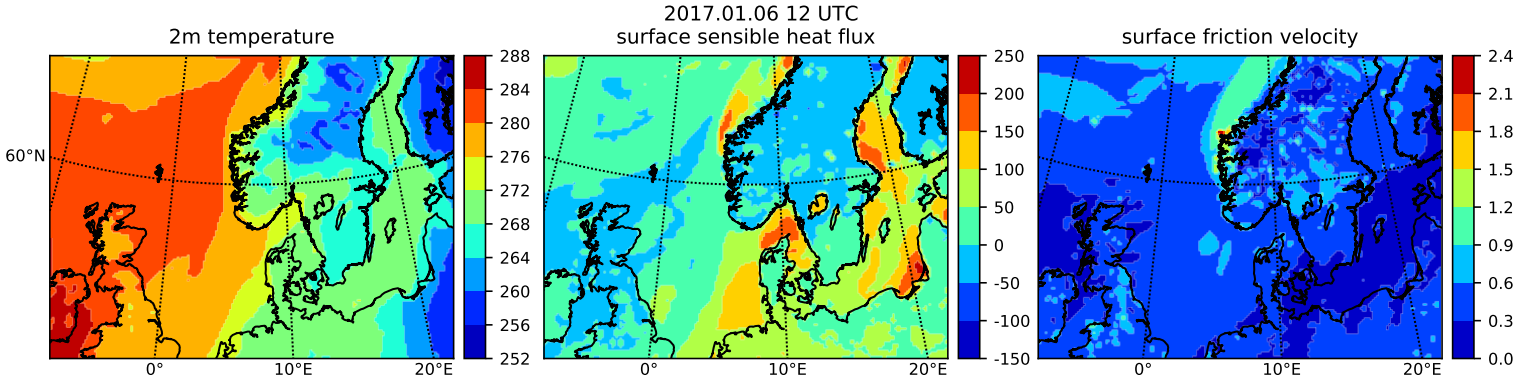


Figure B.2: 2m temperature (left), surface sensible heat flux (middle) and surface friction velocity (right). The prediction is valid 6 hours after initial time, see title on the figure.

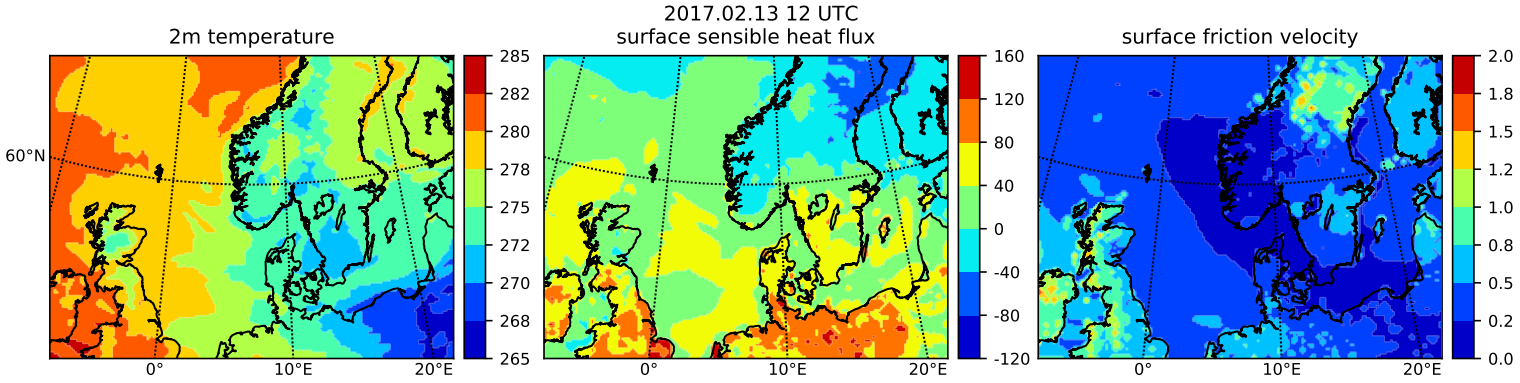


Figure B.3: 2m temperature (left), surface sensible heat flux (middle) and surface friction velocity (right). The prediction is valid 6 hours after initial time, see title on the figure.

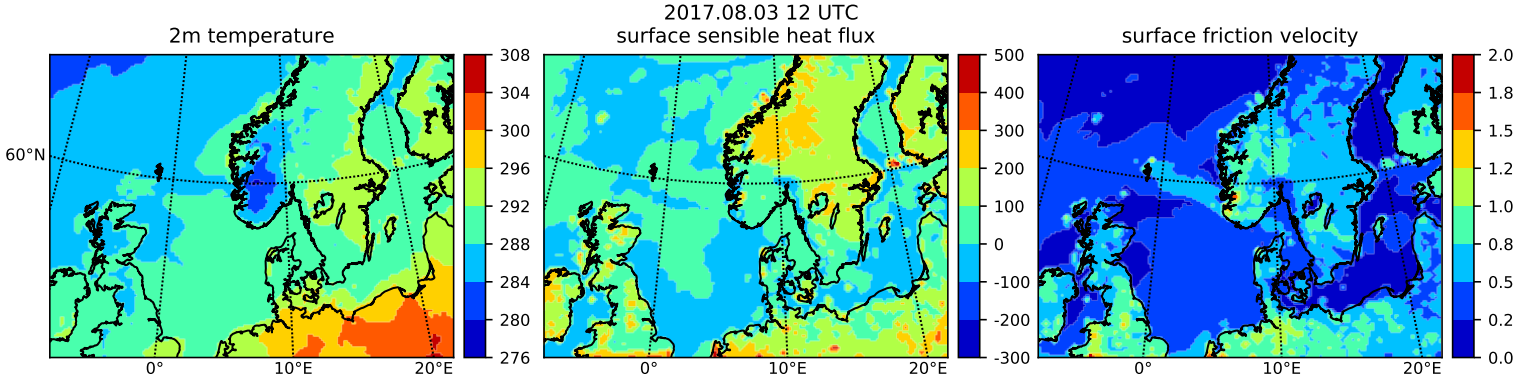


Figure B.4: 2m temperature (left), surface sensible heat flux (middle) and surface friction velocity (right). The prediction is valid 6 hours after initial time, see title on the figure.

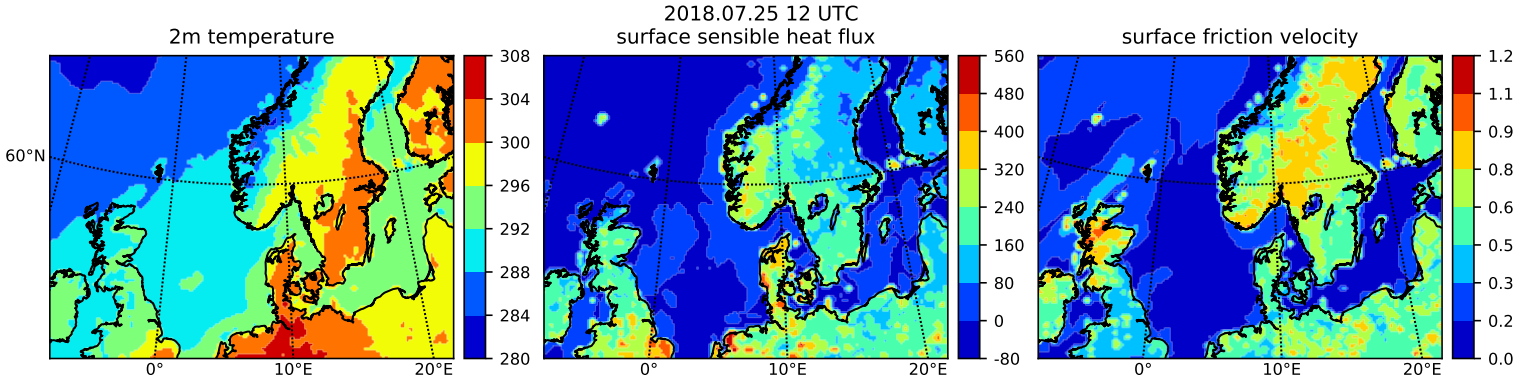


Figure B.5: 2m temperature (left), surface sensible heat flux (middle) and surface friction velocity (right). The prediction is valid 6 hours after initial time, see title on the figure.

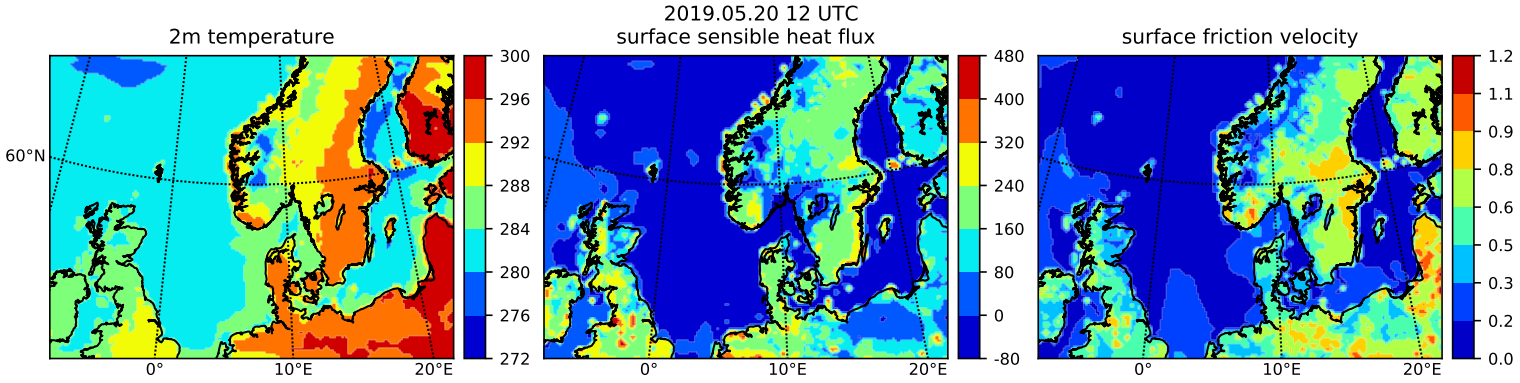


Figure B.6: 2m temperature (left), surface sensible heat flux (middle) and surface friction velocity (right). The prediction is valid 6 hours after initial time, see title on the figure.

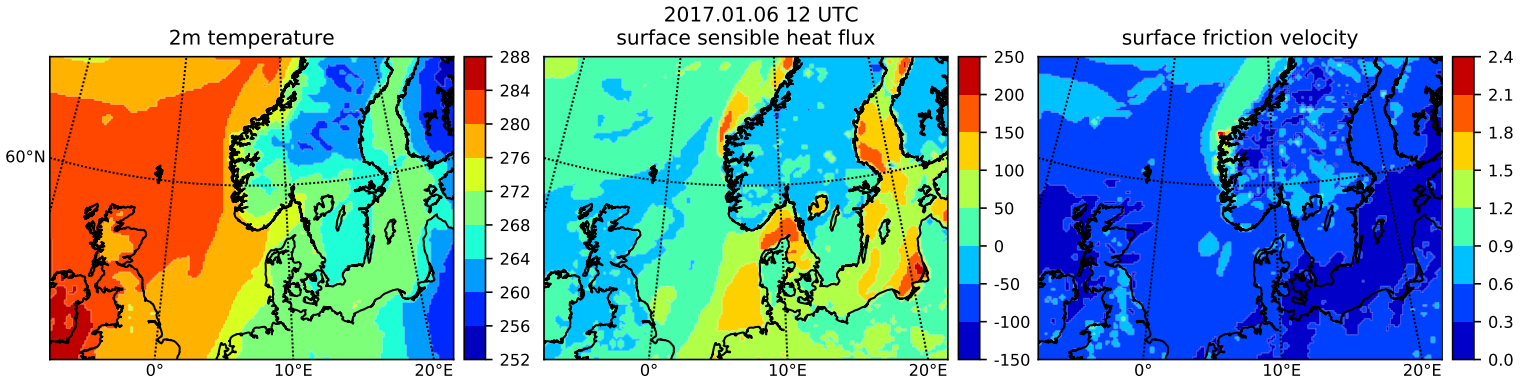


Figure B.7: 2m temperature (left), surface sensible heat flux (middle) and surface friction velocity (right). The predictions are valid 6 hours after initial time, see title on the figure.

Appendix C

Additional variable distributions

This appendix shows the variable distributions for the statistically stable samples using logarithmic scaling. The distributions including all stable samples are shown in Figures C.1 and C.3, while the distributions including only stable samples with non-zero TKE are shown in Figures C.2 and C.4.

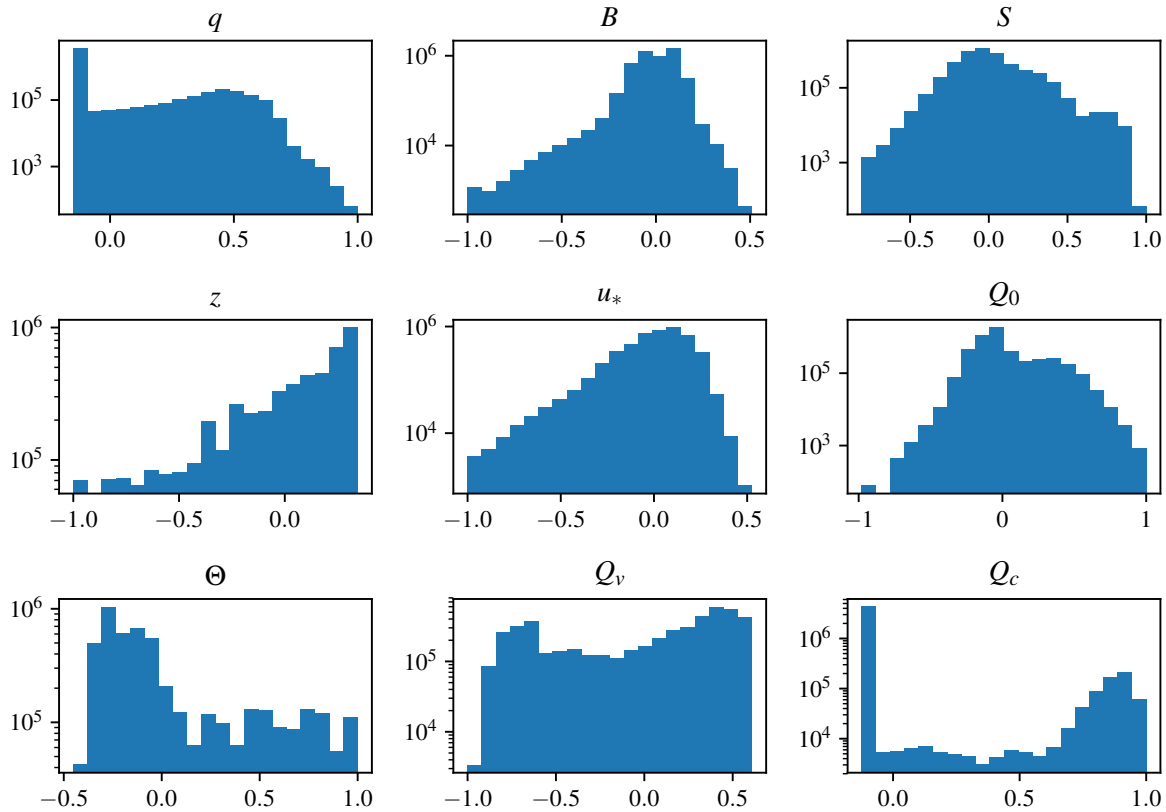


Figure C.1: Distributions of all the input variables. First the logarithmic scaling is applied and then the linear scaling from Equation (4.3). The histograms are based on all stable samples from the training dataset.

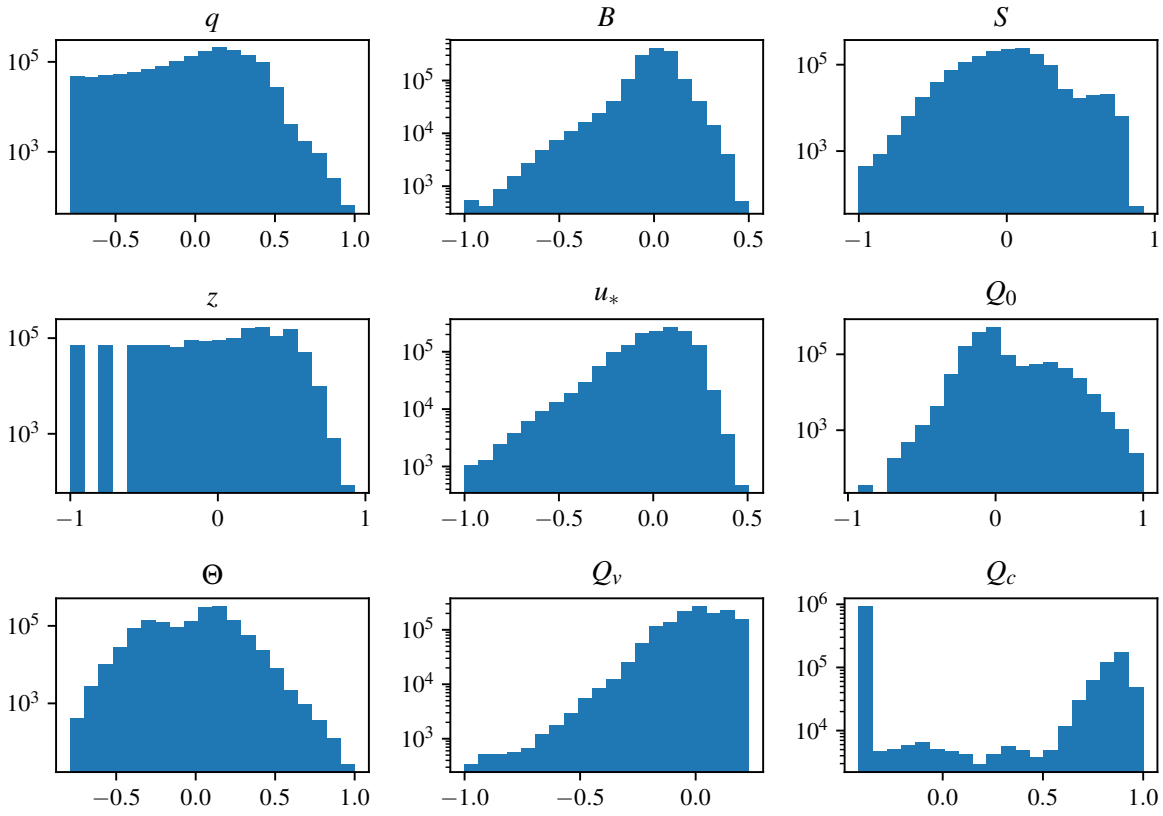


Figure C.2: Distributions of all the input variables. First the logarithmic scaling is applied and then the linear scaling from Equation (4.3). The histograms are based on stable samples with non-zero TKE from the training dataset.

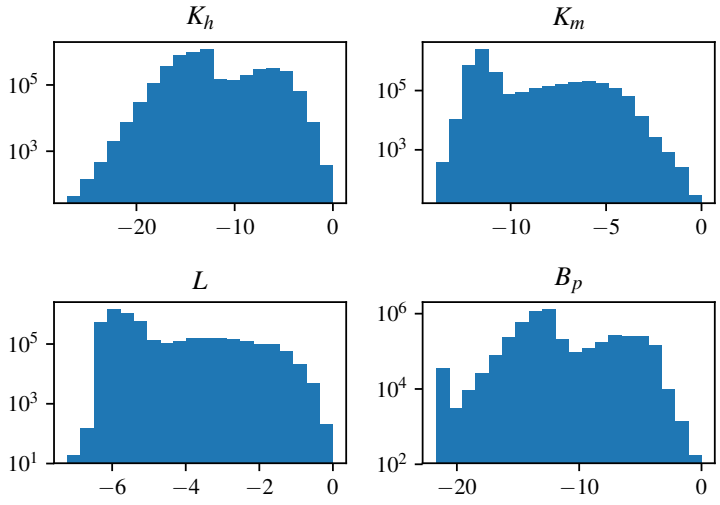


Figure C.3: Distributions of all the output variables. First the linear scaling from Equation (4.4) is applied, and then the logarithmic scaling. The histograms are based on all stable samples from the training dataset.

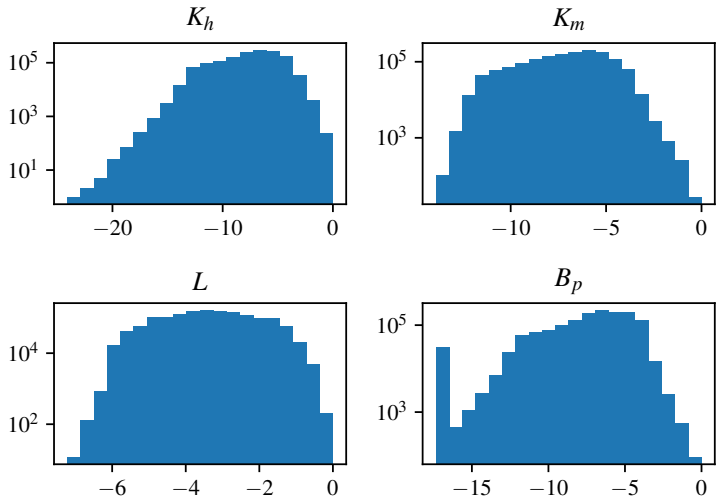


Figure C.4: Distributions of all the output variables. First the linear scaling from Equation (4.4) is applied, and then the logarithmic scaling. The histograms are based on stable samples with non-zero TKE from the training dataset.

Appendix D

Additional examples of predictions by the neural networks

This appendix shows more examples of predictions by the three models compared in Section 4.4.7. Whereas the two figures shown in Section 4.4.7 were choses as the two models with most extreme values of surface sensible heat flux, the ten examples shown here are randomly selected.

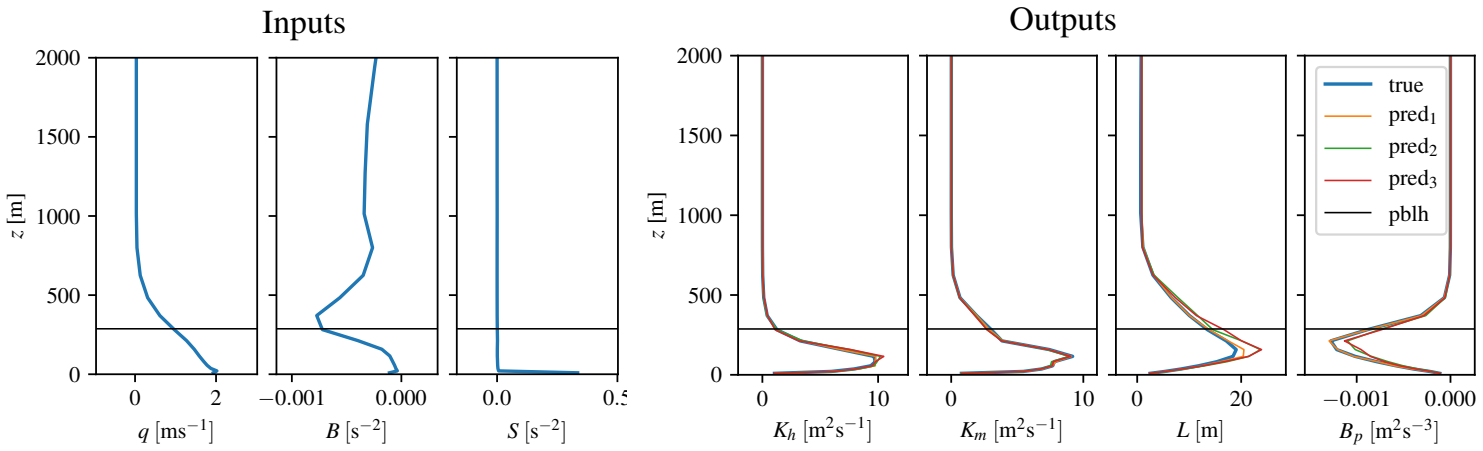


Figure D.1: Example of predictions by the three neural networks. The three plots to the left show the profiles of the three inputs, q, B and S, while the four plots to the right show the output variables K_h, K_m, L and B_p. Both the "true" values and the predictions by the three networks are shown. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx -2.2\text{W/m}^2$, and the value of the friction velocity for this case is $u_* \approx 0.7\text{m/s}$. The planetary boundary layer height, *pblh*, is showed as well.

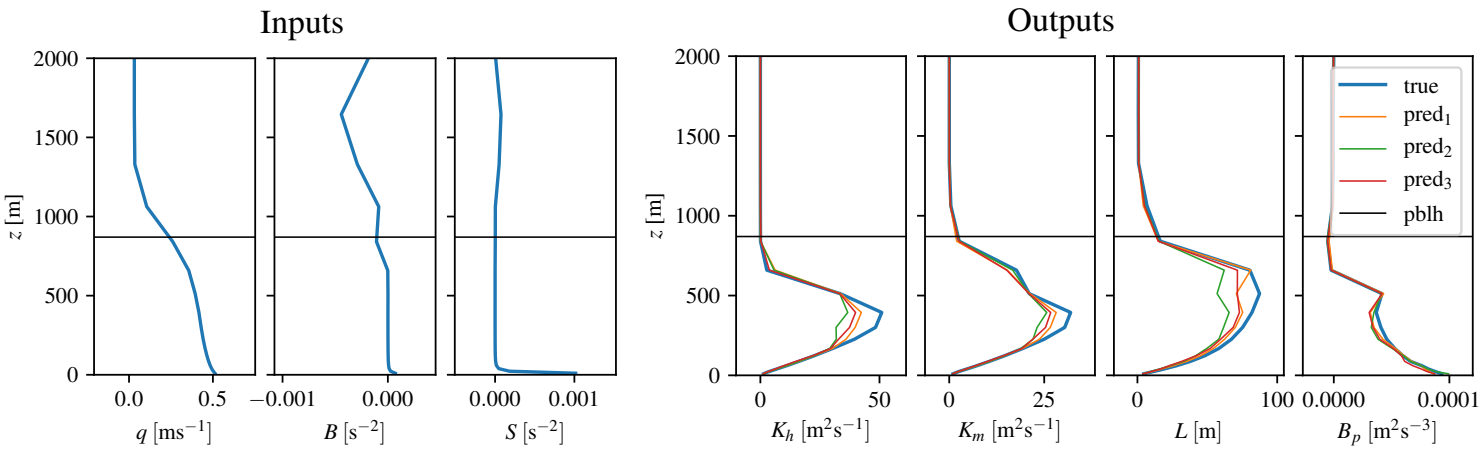


Figure D.2: Example of predictions by the three neural networks. Similar to Figure 5.16. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx 0.5\mathrm{W/m^2}$, and the value of the friction velocity for this case is $u_* \approx 0.2\mathrm{m/s}$.

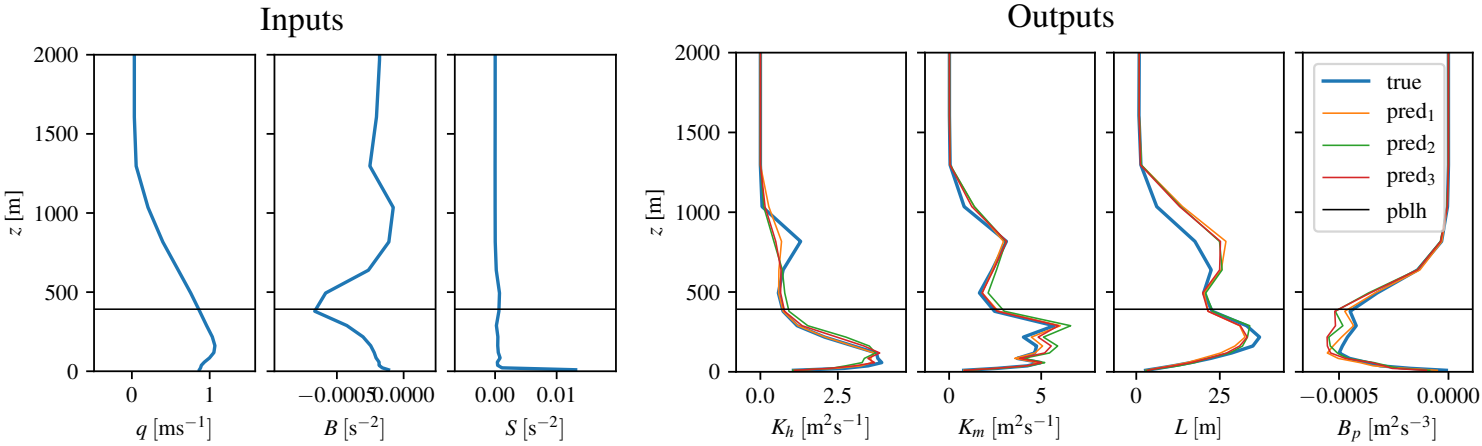


Figure D.3: Example of predictions by the three neural networks. Similar to Figure 5.16. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx 6.4\mathrm{W/m^2}$, and the value of the friction velocity for this case is $u_* \approx 0.3\mathrm{m/s}$.

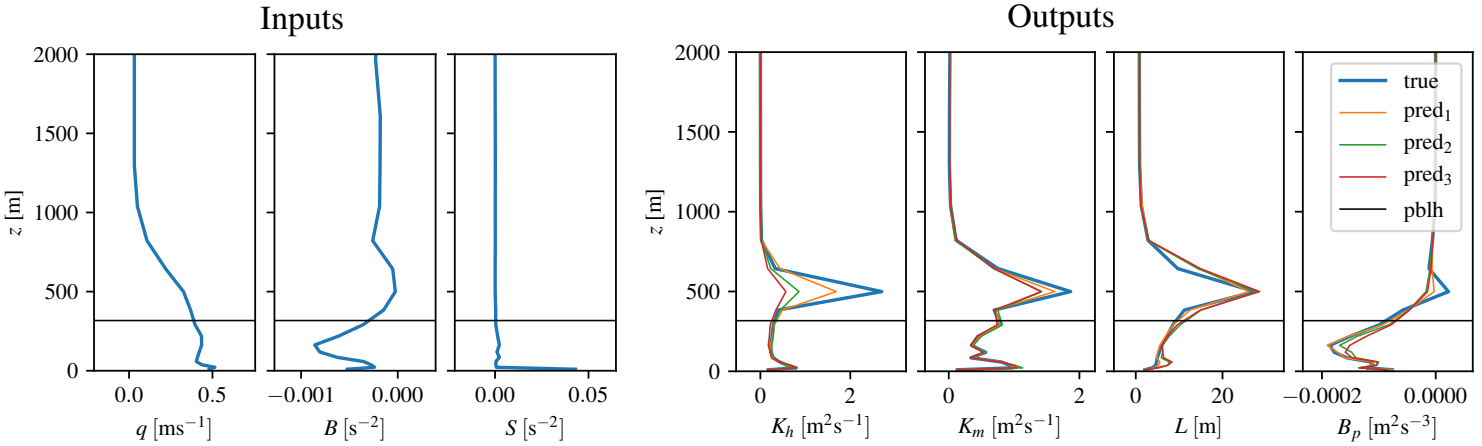


Figure D.4: Example of predictions by the three neural networks. Similar to Figure 5.16. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx -2.2\mathrm{W/m^2}$, and the value of the friction velocity for this case is $u_* \approx 0.2\mathrm{m/s}$.

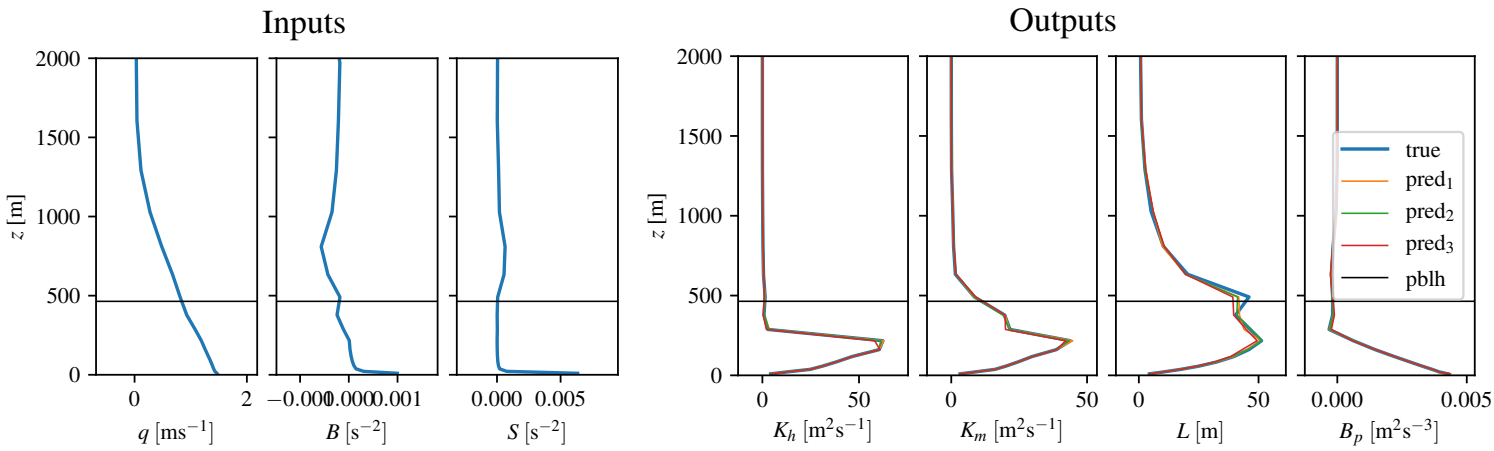


Figure D.5: Example of predictions by the three neural networks. Similar to Figure 5.16. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx 153\mathrm{W/m^2}$, and the value of the friction velocity for this case is $u_* \approx 0.5\mathrm{m/s}$.

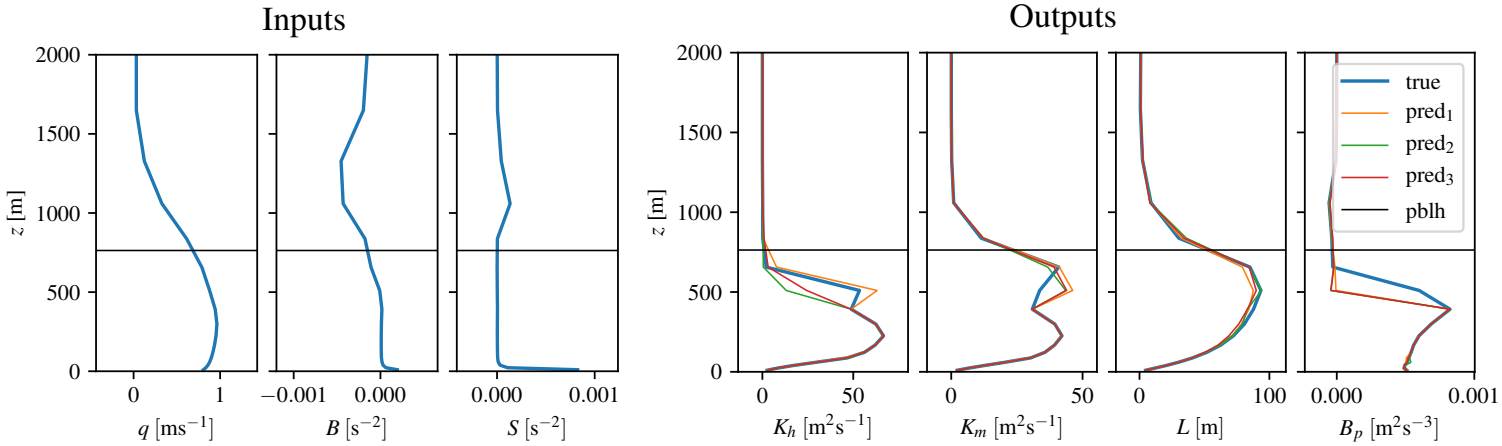


Figure D.6: Example of predictions by the three neural networks. Similar to Figure 5.16. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx 134\mathrm{W/m^2}$, and the value of the friction velocity for this case is $u_* \approx 0.2\mathrm{m/s}$.

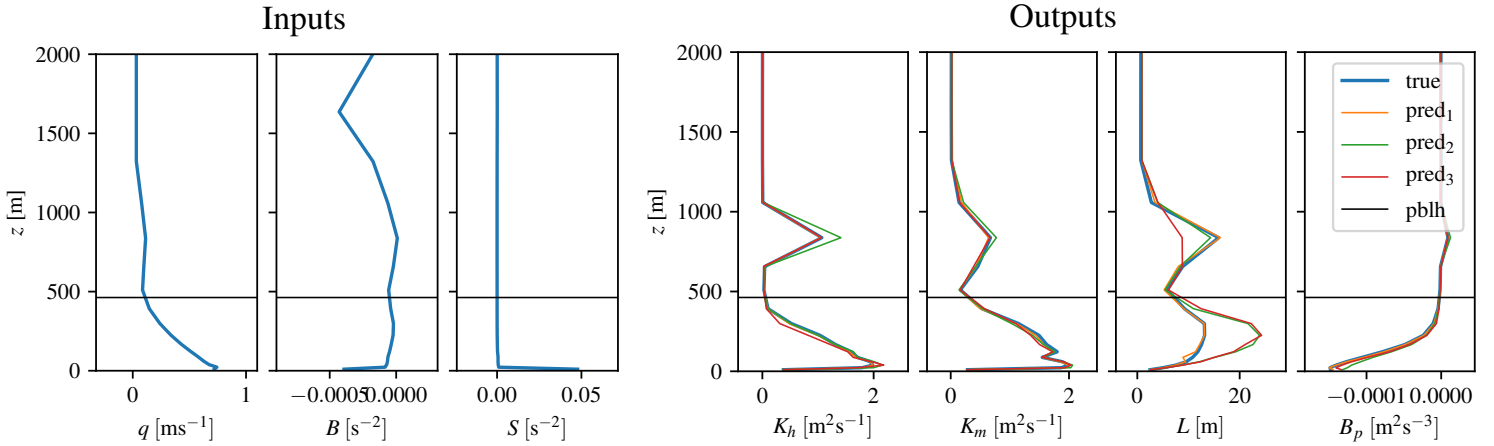


Figure D.7: Example of predictions by the three neural networks. Similar to Figure 5.16. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx -8.1\mathrm{W/m^2}$, and the value of the friction velocity for this case is $u_* \approx 0.3\mathrm{m/s}$.

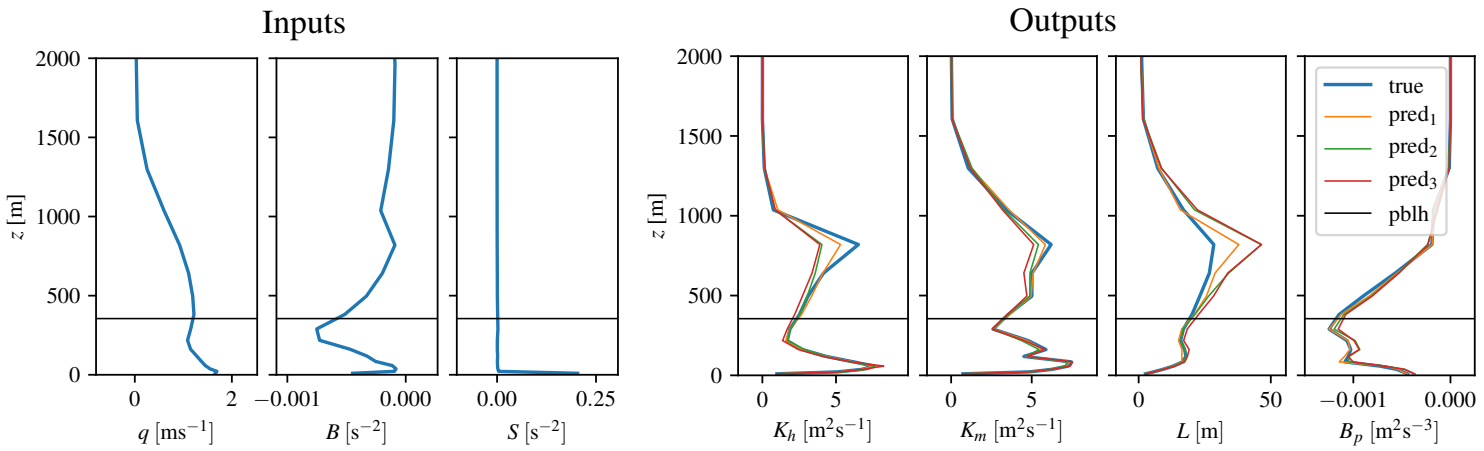


Figure D.8: Example of predictions by the three neural networks. Similar to Figure 5.16. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx -15.9\text{W/m}^2$, and the value of the friction velocity for this case is $u_* \approx 0.6\text{m/s}$.

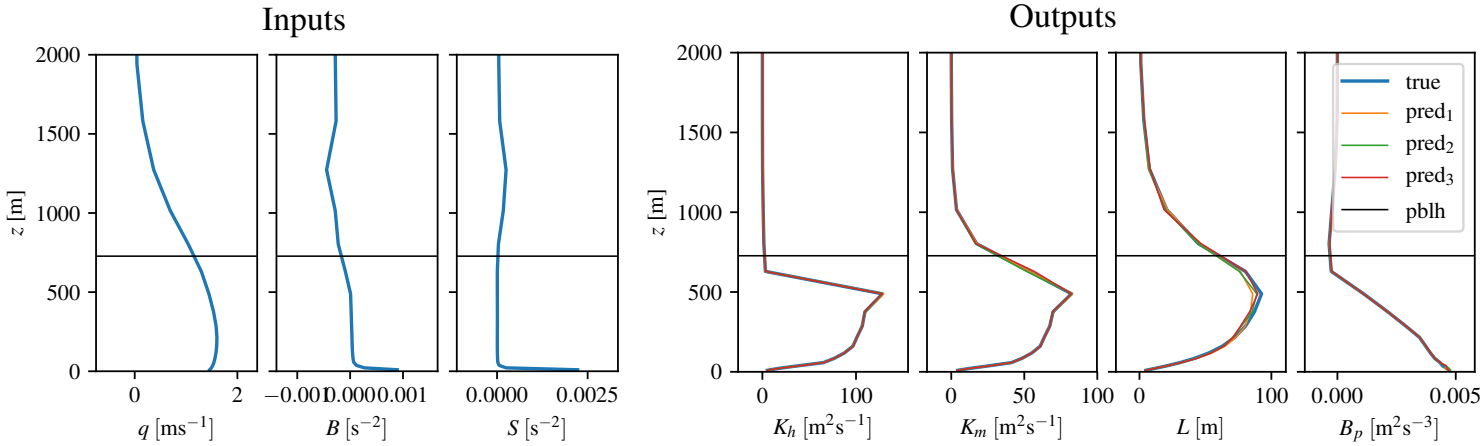


Figure D.9: Example of predictions by the three neural networks. Similar to Figure 5.16. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx 164\text{W/m}^2$, and the value of the friction velocity for this case is $u_* \approx 0.5\text{m/s}$.

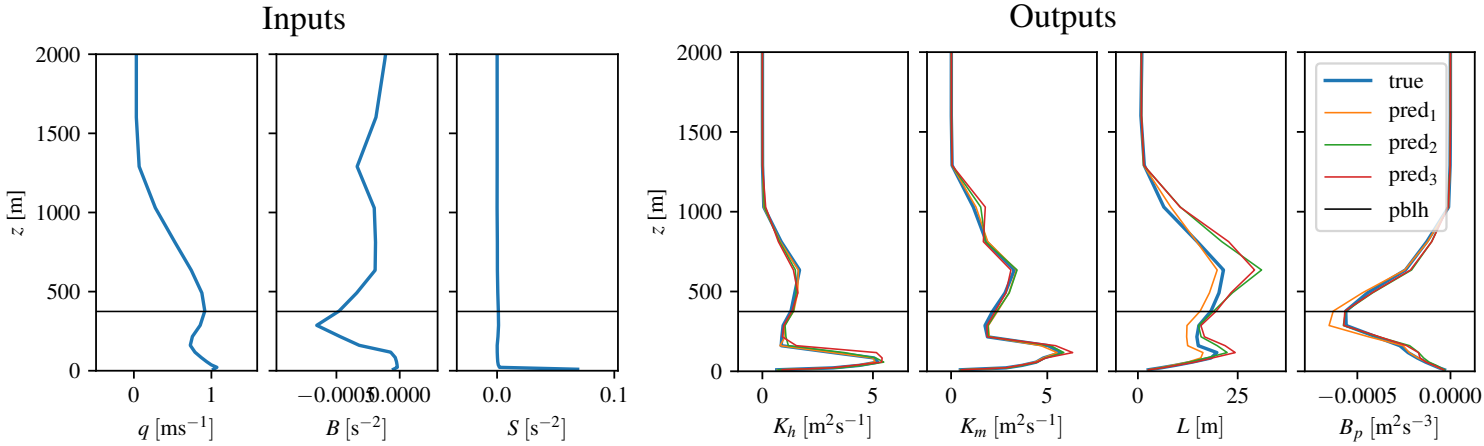


Figure D.10: Example of predictions by the three neural networks. Similar to Figure 5.16. The example is randomly selected. The value of the surface sensible heat flux is $Q_0 \approx -0.5\text{W/m}^2$, and the value of the friction velocity for this case is $u_* \approx 0.4\text{m/s}$.

Appendix E

Additional results

This appendix contain all the extra figures to Chapter 5. Section E.1 shows plots of the the anomaly fields for the three ANN schemes for the second simulation (initiated 2018.08.02 06 UTC). Section E.2 shows plots of the the anomaly fields for the best ANN scheme, the YSU scheme and the MYJ scheme, also for the second simulation. Section E.3 shows 10 additional examples of Θ and wind speed profiles. These examples are from 5 randomly selected locations and for both simulations.

E.1 Extra plots comparing the three ANN schemes

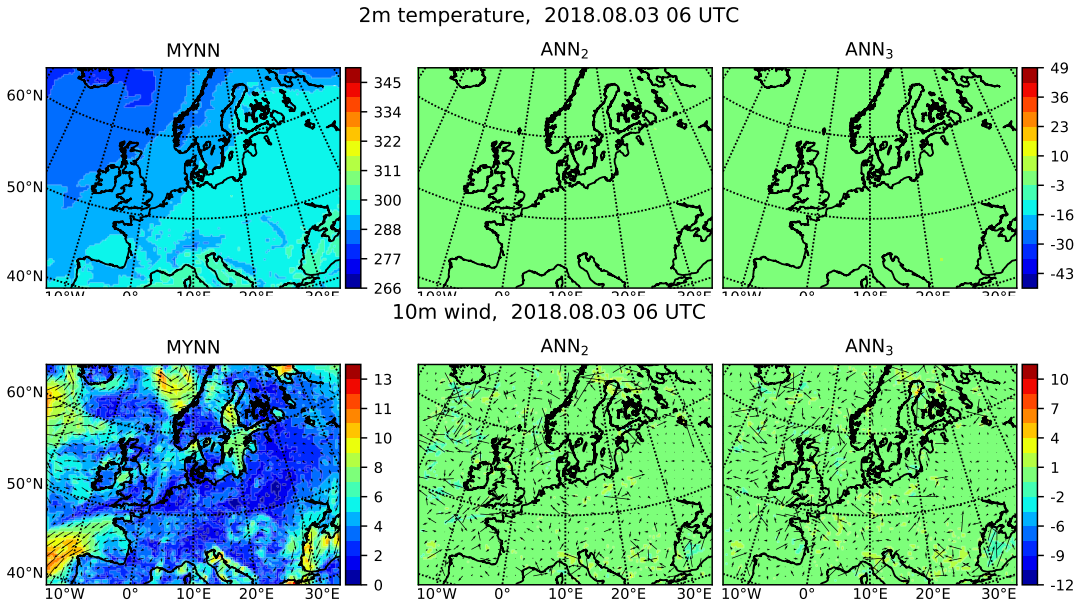


Figure E.1: Predictions of 2m temperature (first row) and 10m wind (second row). First column shows plots of the predictions of the MYNN scheme, while the three next columns show the anomaly fields for the three ANN schemes, computed as $pred_{\text{ANN}} - pred_{\text{MYNN}}$. All predictions are valid at 2018.08.03 06 UTC, i.e. 24 hours after the initial time. The names ANN_1, ANN_2, ANN_3 correspond to the model numbers in Section 4.4.7.

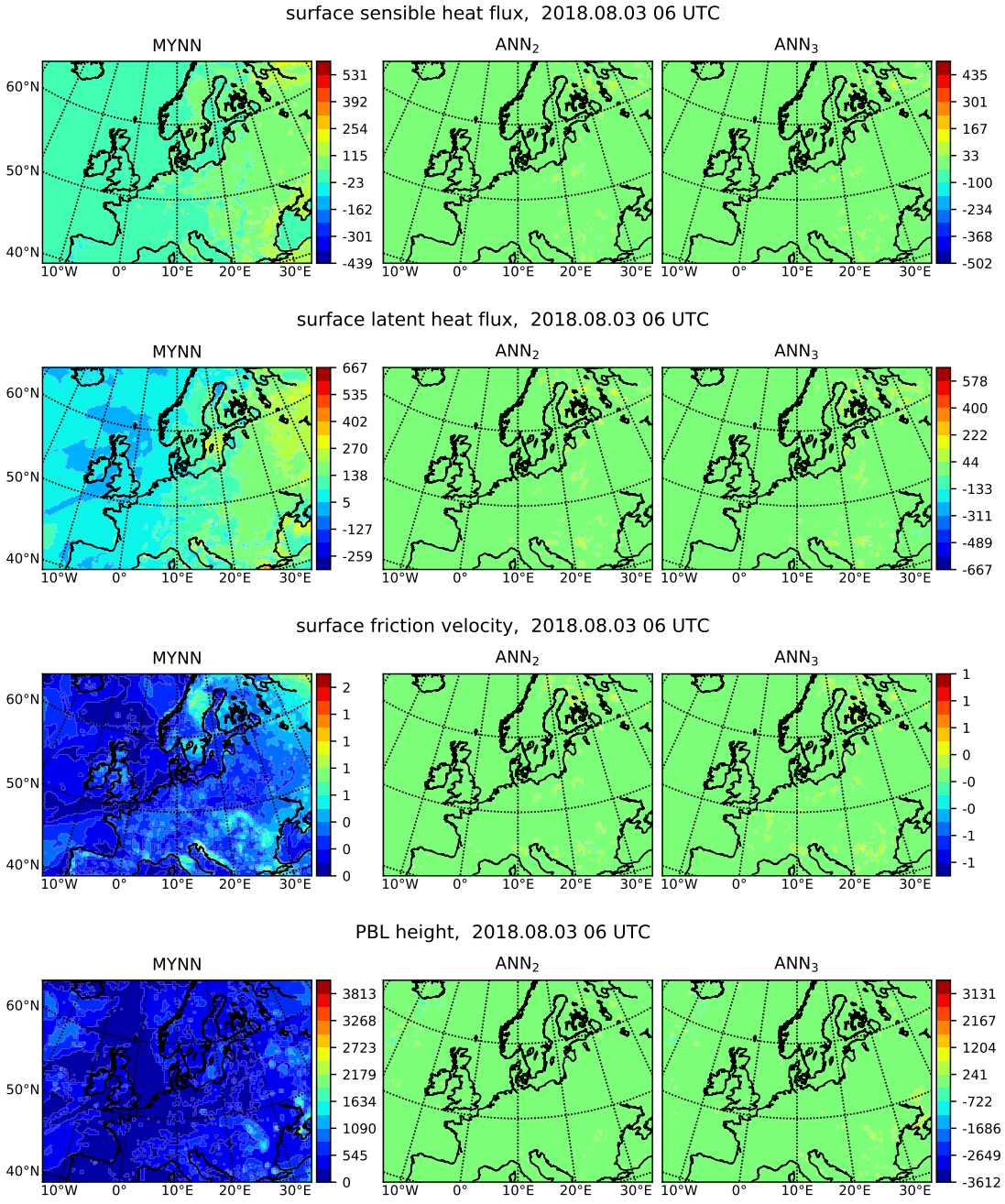


Figure E.2: Similar to Figure 5.2, except for different variables. This figure shows surface sensible heat flux (first row), surface latent heat flux (second row), surface friction velocity (third row) and pblh (fourth row).

E.2 Extra plots for evaluation of the best ANN scheme

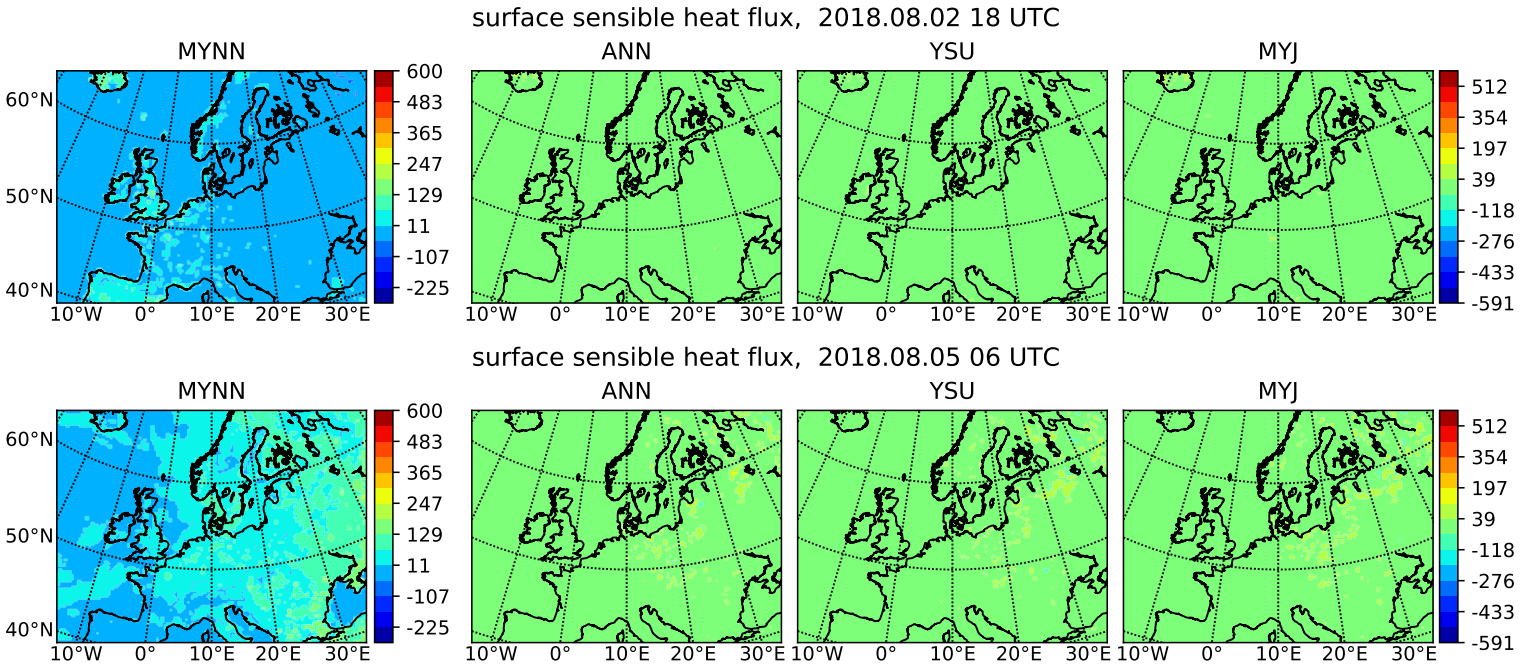


Figure E.3: Predictions of surface sensible heat flux after 12 hours (first row), and after 72 hours (second row). First column shows plots of the predictions of the MYNN scheme, while the three next columns show the anomaly fields for the ANN scheme, YSU scheme and the MYJ scheme.

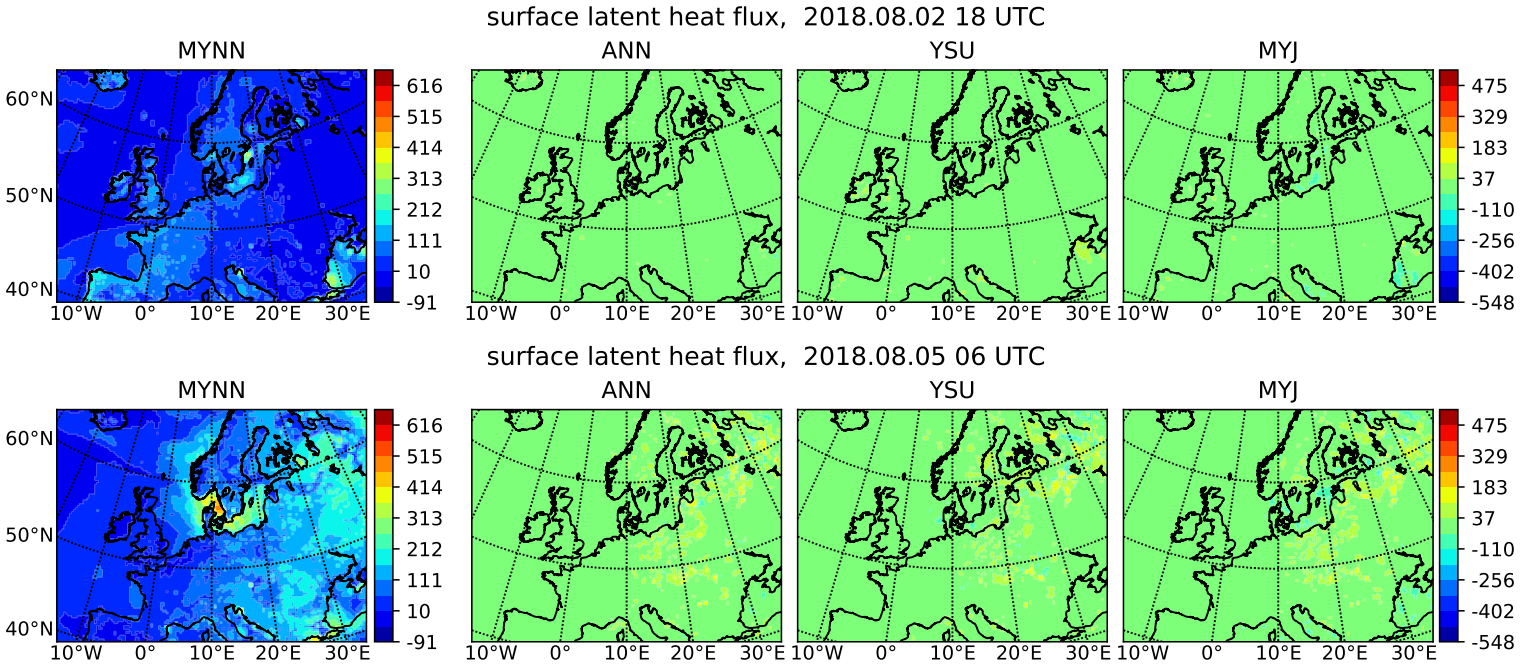


Figure E.4: Predictions of surface sensible latent flux after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure E.3.

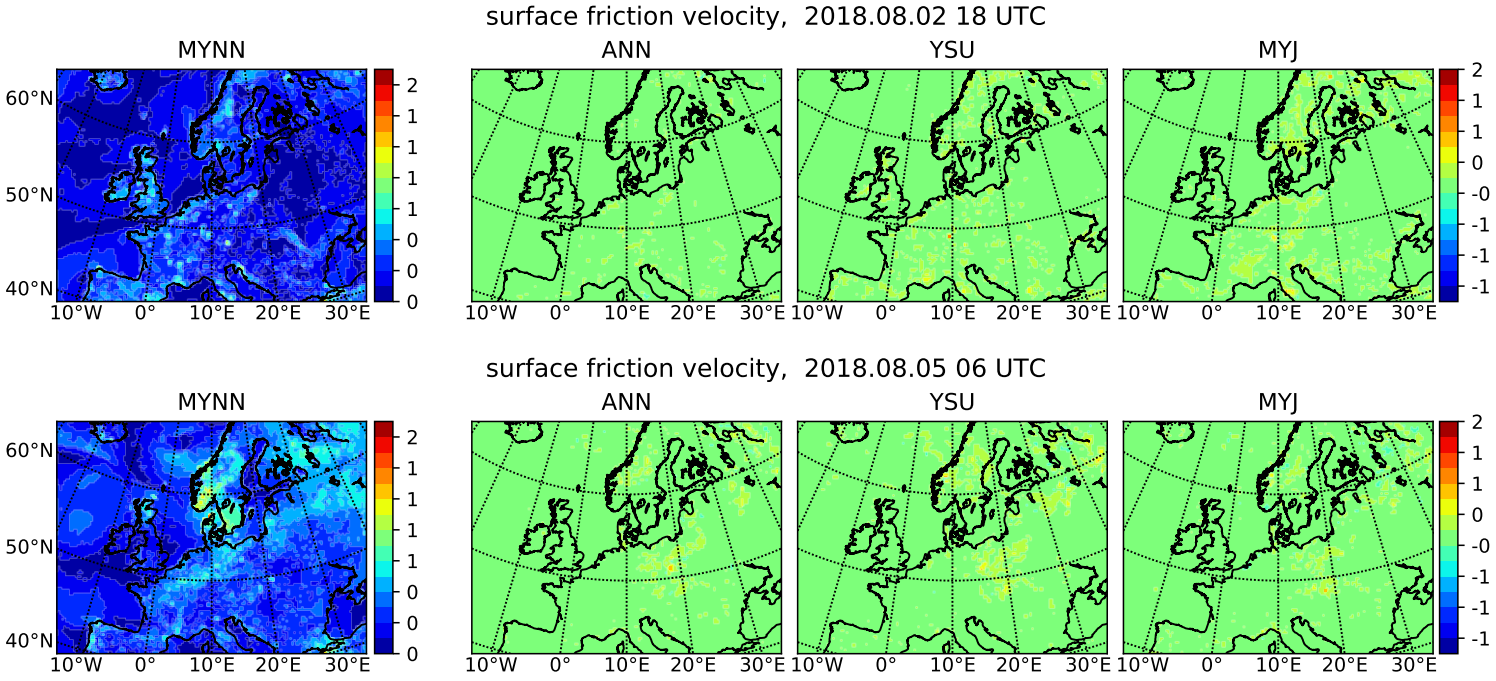


Figure E.5: Predictions of surface friction velocity after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure E.3.

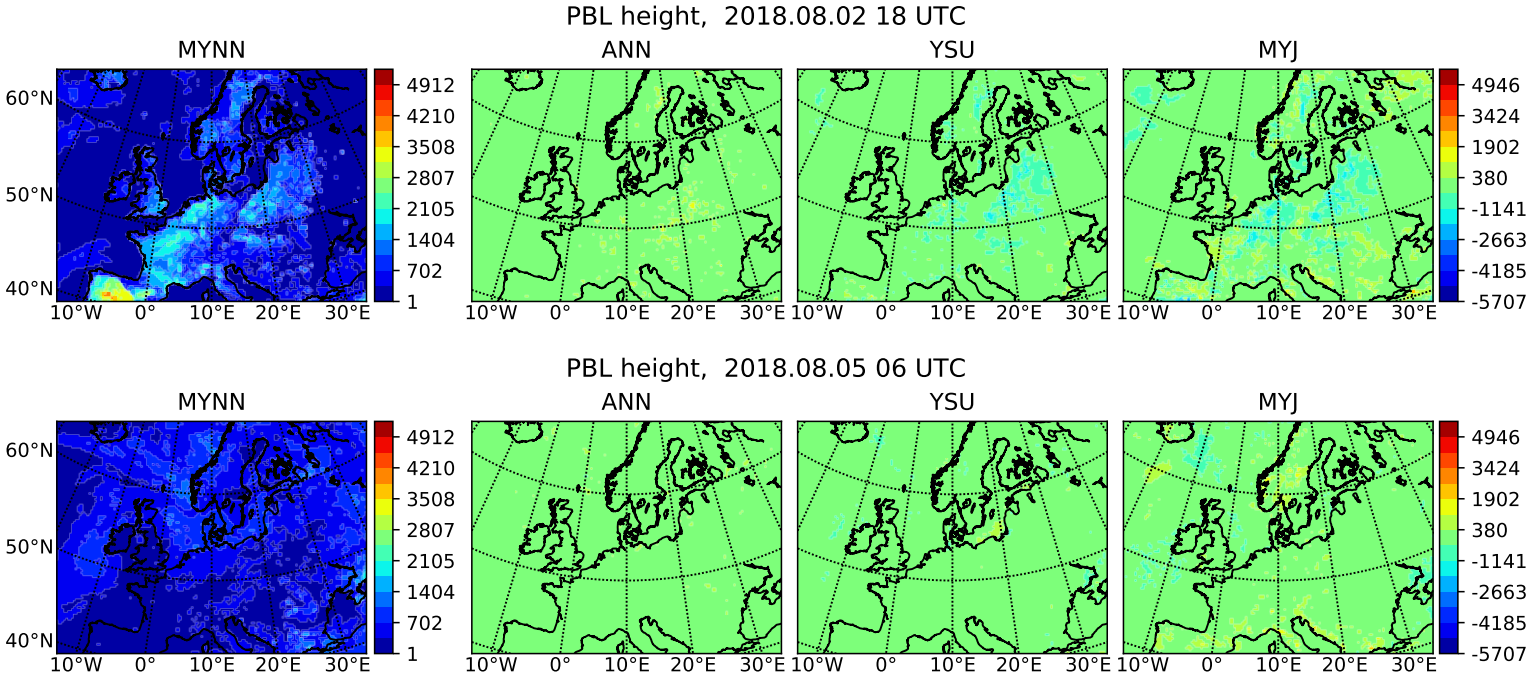


Figure E.6: Predictions of *pblh* after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure E.3.

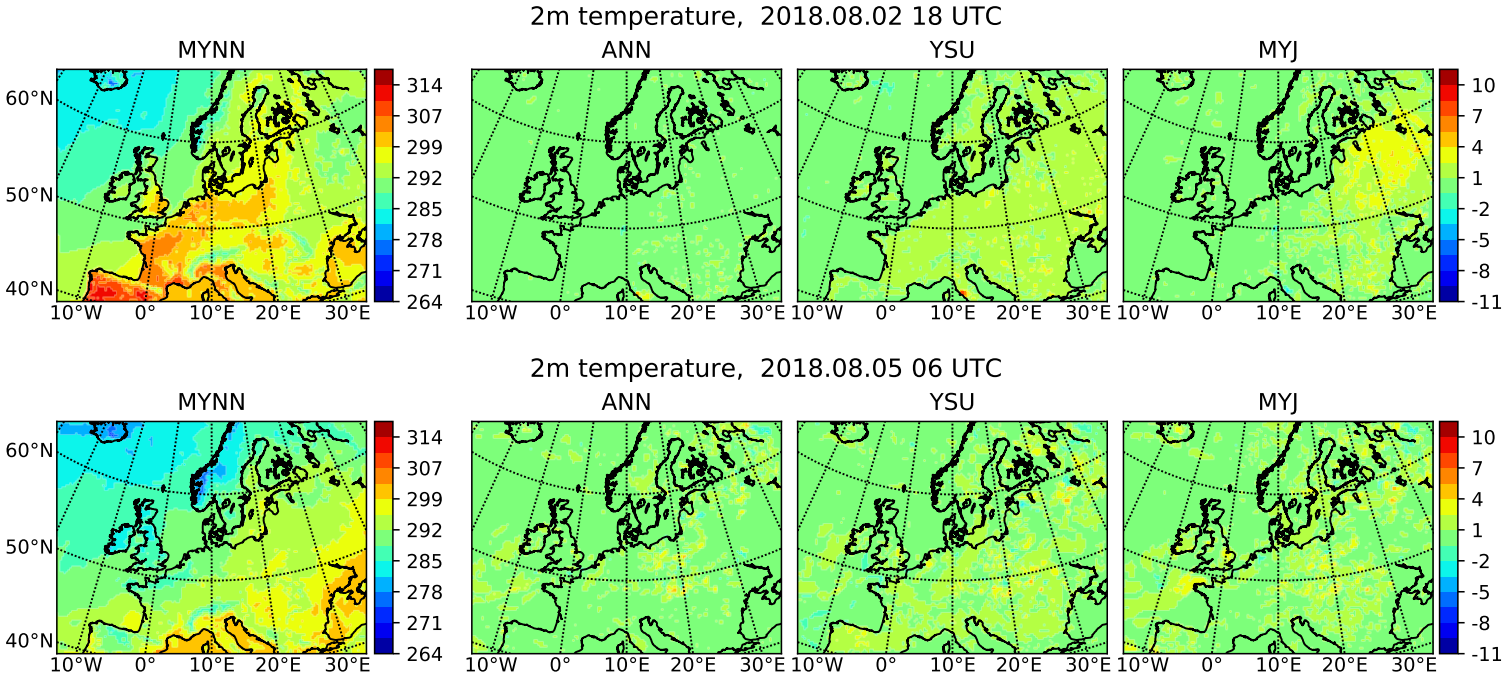


Figure E.7: Predictions of 2m temperature after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure E.3.

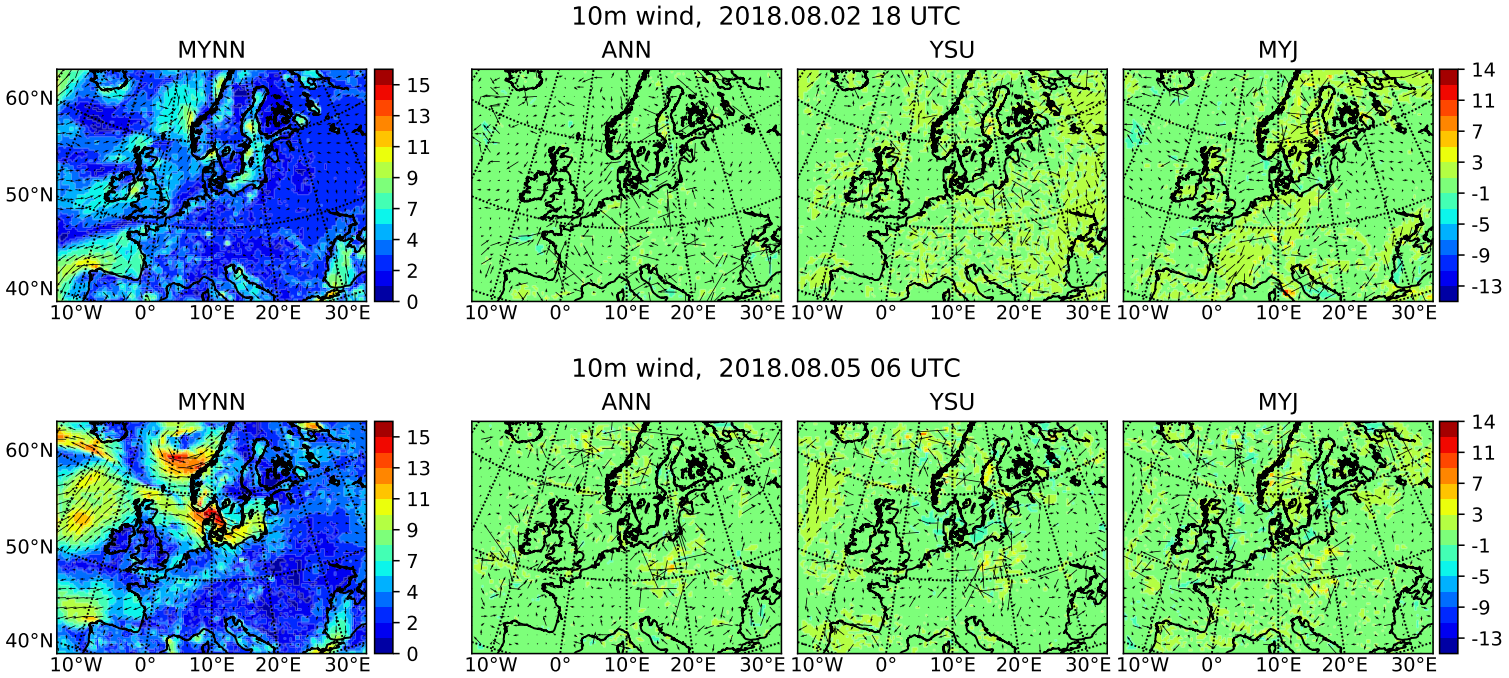


Figure E.8: Predictions of 10m wind after 12 hours (first row), and after 72 hours (second row). Otherwise similar to Figure E.3.

E.3 Extra examples of Θ and wind speed profiles

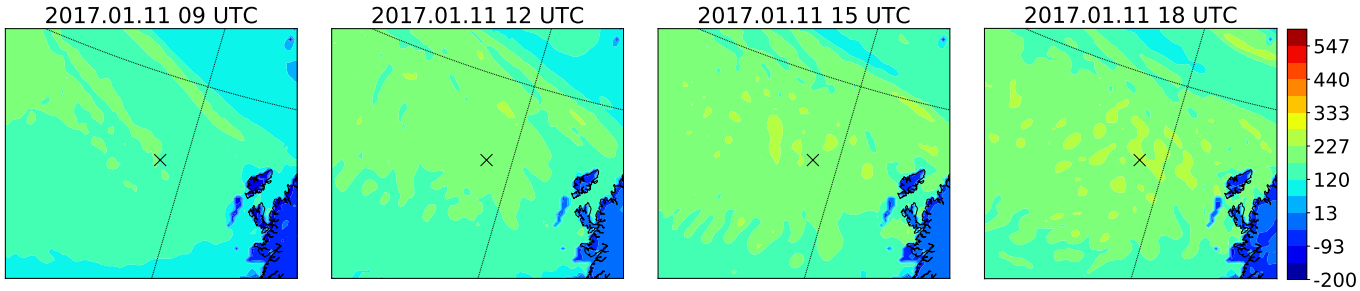


Figure E.9: Surface sensible heat fluxes for the location of the profiles shown in Figure E.10. The plots are based on the simulation initiated at 2017.01.11 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

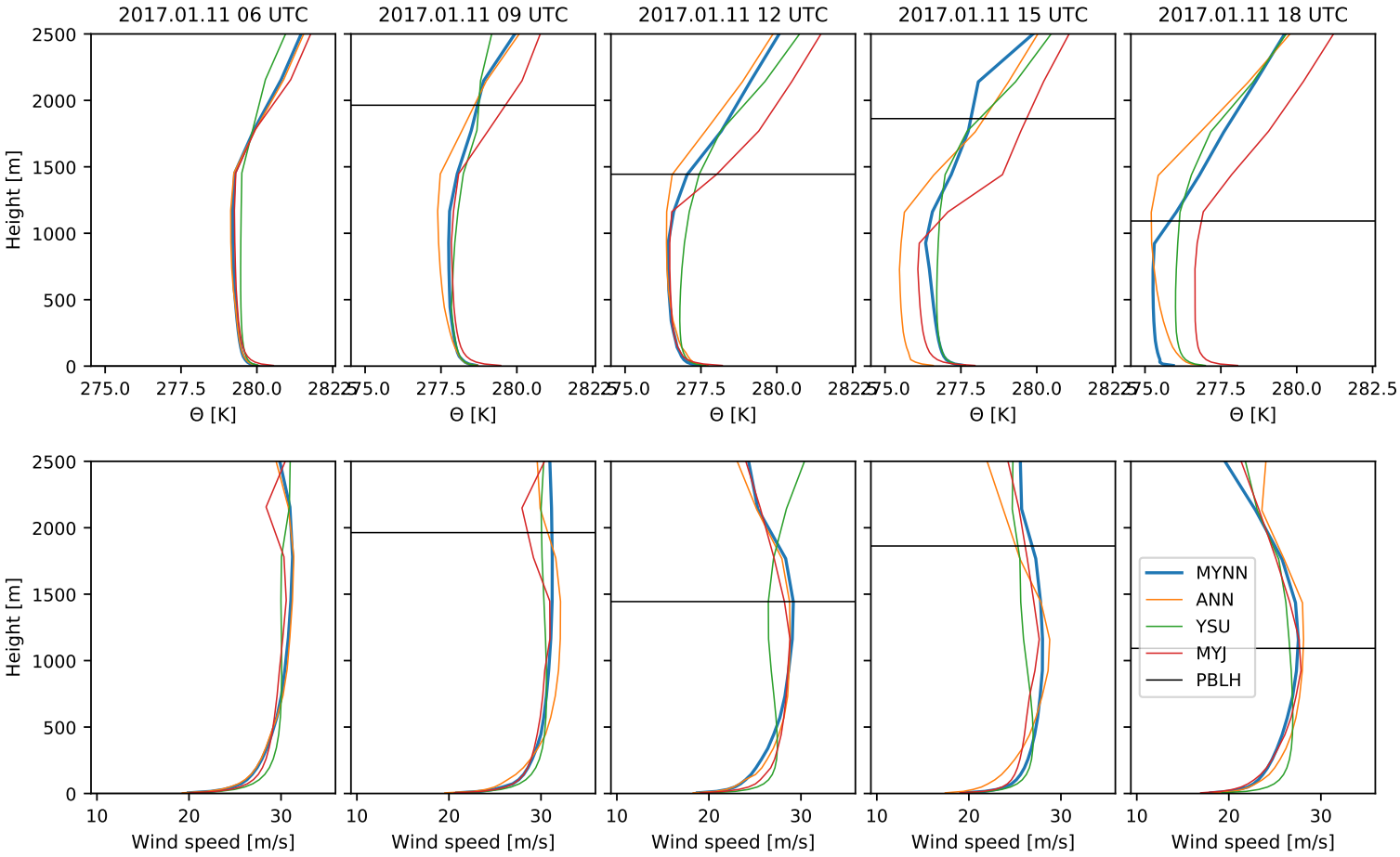


Figure E.10: Profiles of Θ and wind speed for the location shown in Figure E.9. The location is randomly selected. The profiles are from the simulation initiated at 2017.01.11 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

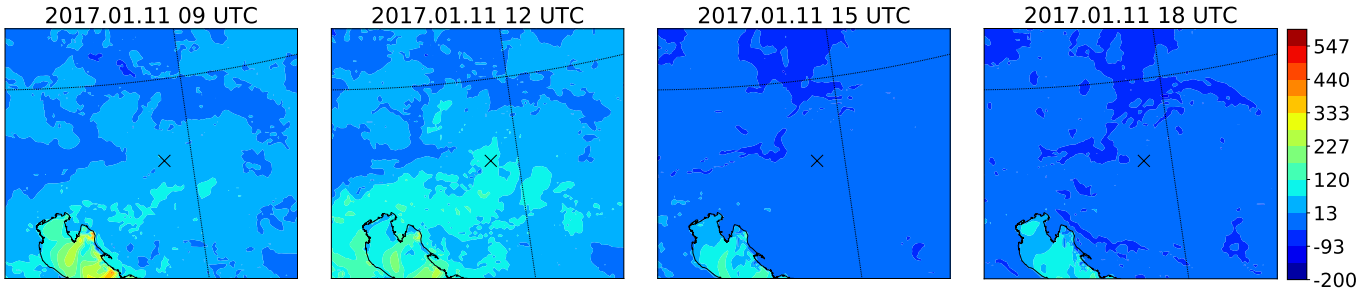


Figure E.11: Surface sensible heat fluxes for the location of the profiles shown in Figure E.12. The plots are based on the simulation initiated at 2017.01.11 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

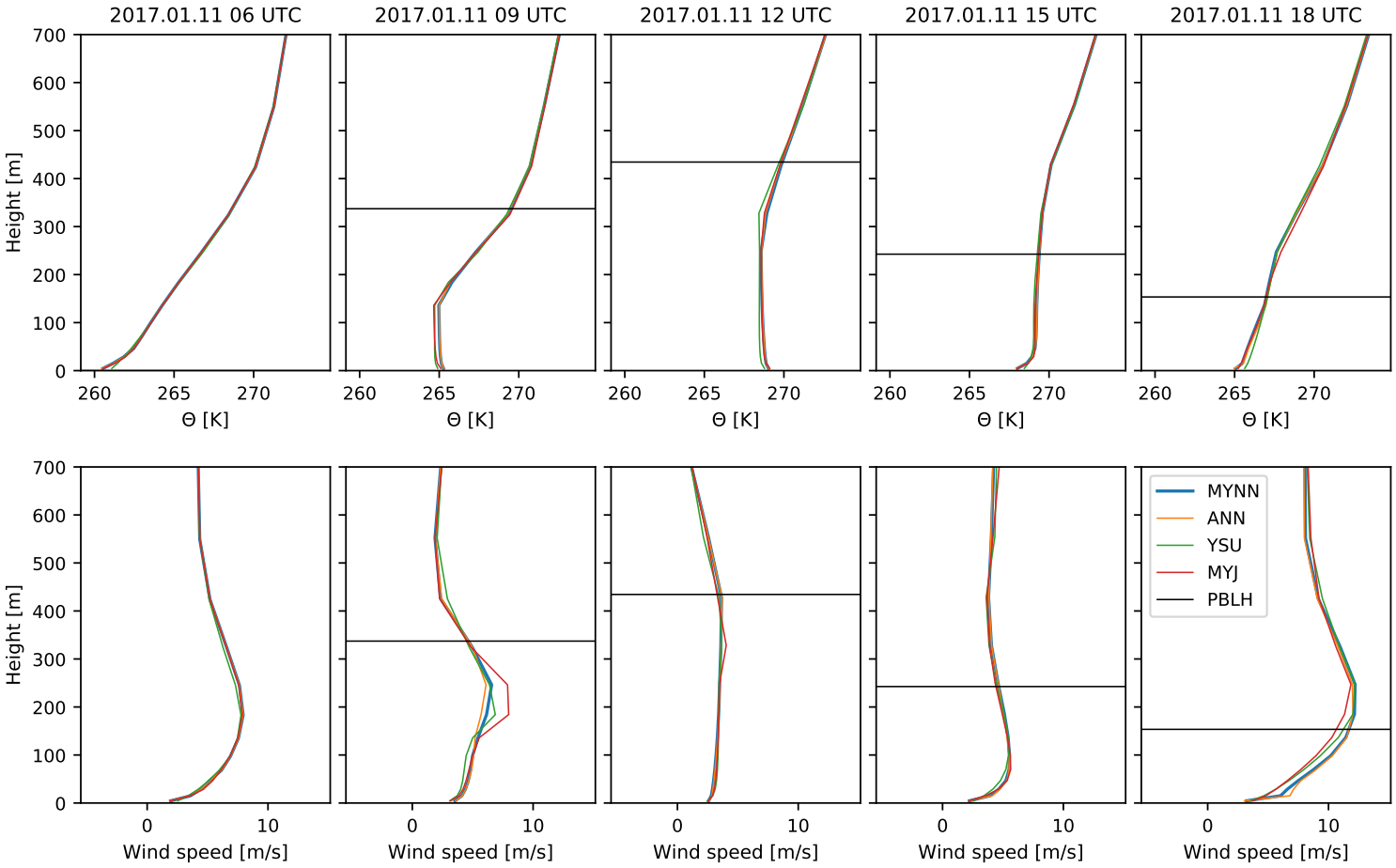


Figure E.12: Profiles of Θ and wind speed for the location shown in Figure E.11. The location is randomly selected. The profiles are from the simulation initiated at 2017.01.11 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

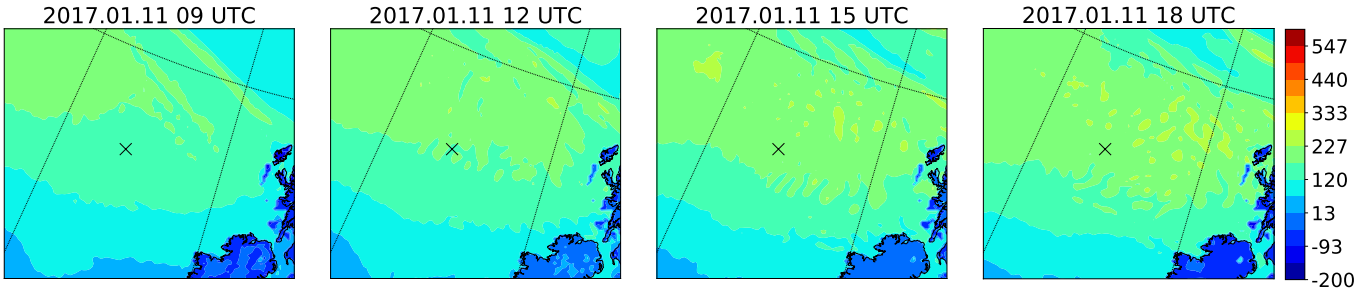


Figure E.13: Surface sensible heat fluxes for the location of the profiles shown in Figure E.14. The plots are based on the simulation initiated at 2017.01.11 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

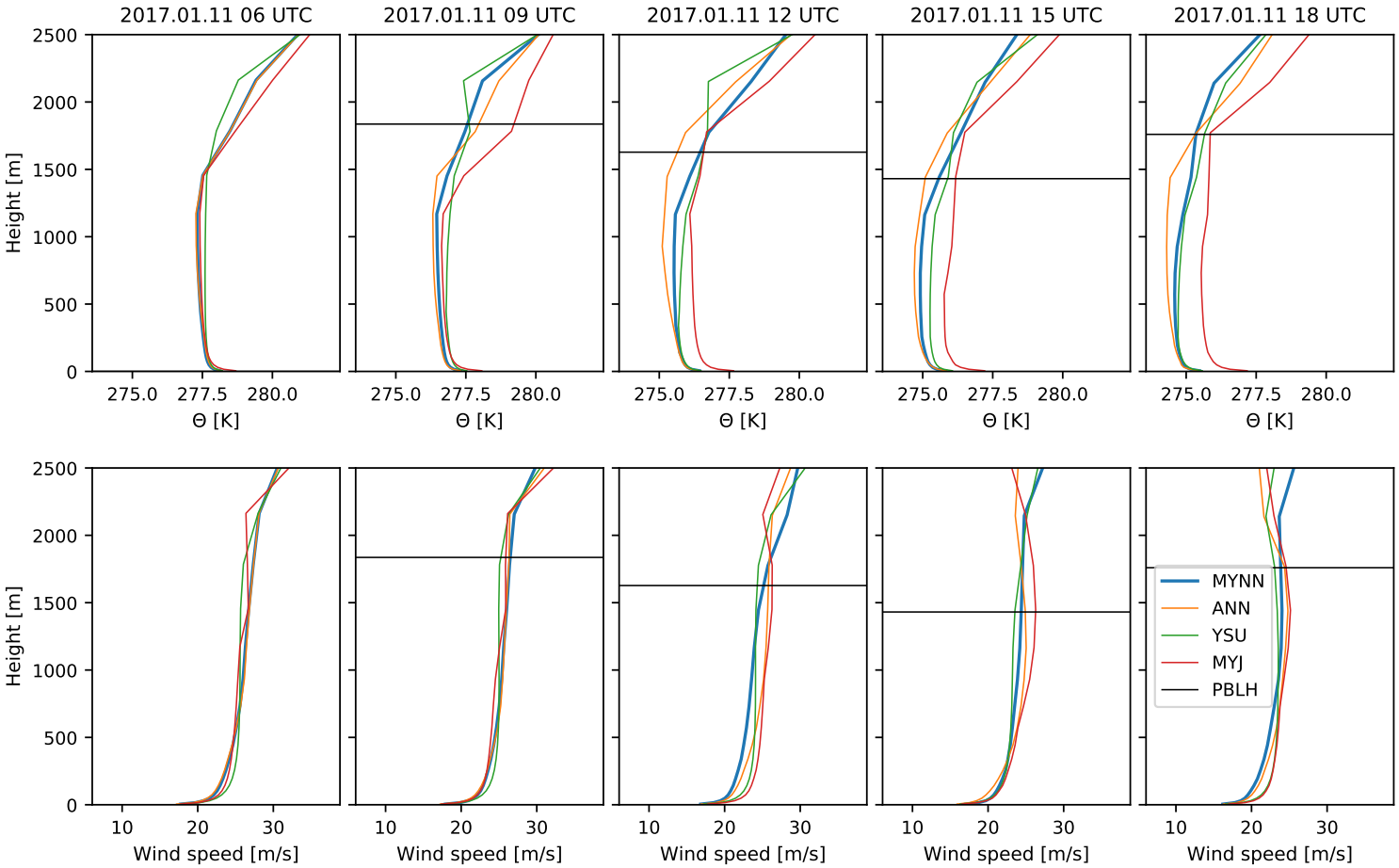


Figure E.14: Profiles of Θ and wind speed for the location shown in Figure E.13. The location is randomly selected. The profiles are from the simulation initiated at 2017.01.11 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

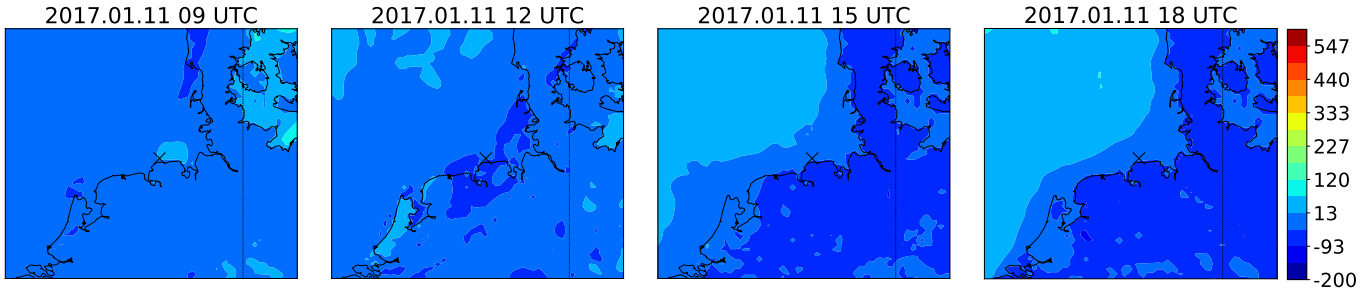


Figure E.15: Surface sensible heat fluxes for the location of the profiles shown in Figure E.16. The plots are based on the simulation initiated at 2017.01.11 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

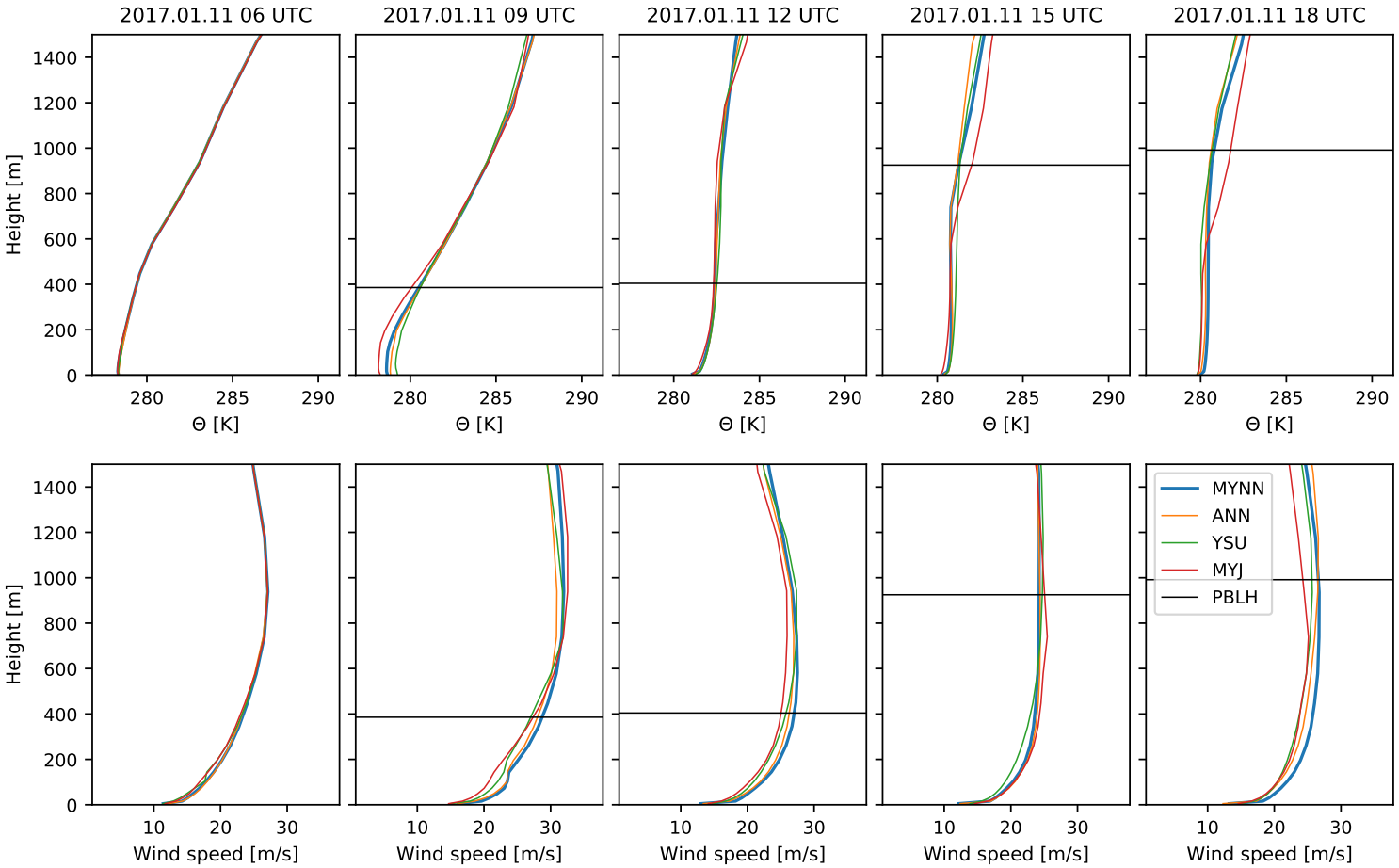


Figure E.16: Profiles of Θ and wind speed for the location shown in Figure E.15. The location is randomly selected. The profiles are from the simulation initiated at 2017.01.11 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

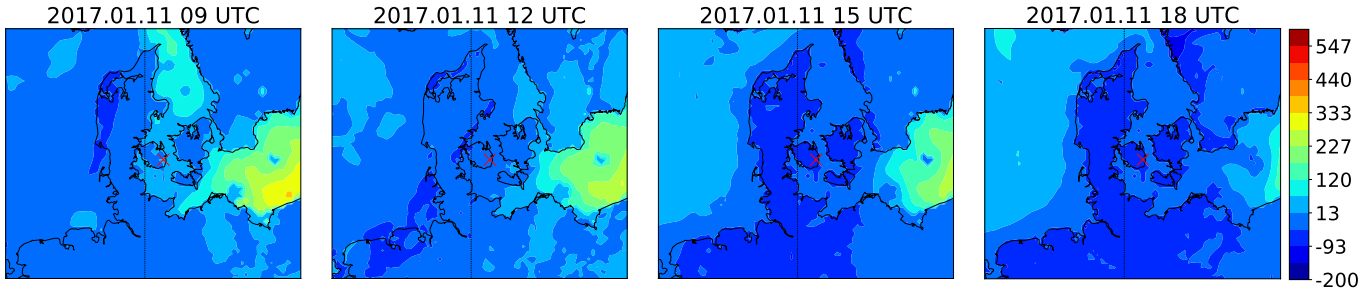


Figure E.17: Surface sensible heat fluxes for the location of the profiles shown in Figure E.18. The plots are based on the simulation initiated at 2017.01.11 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

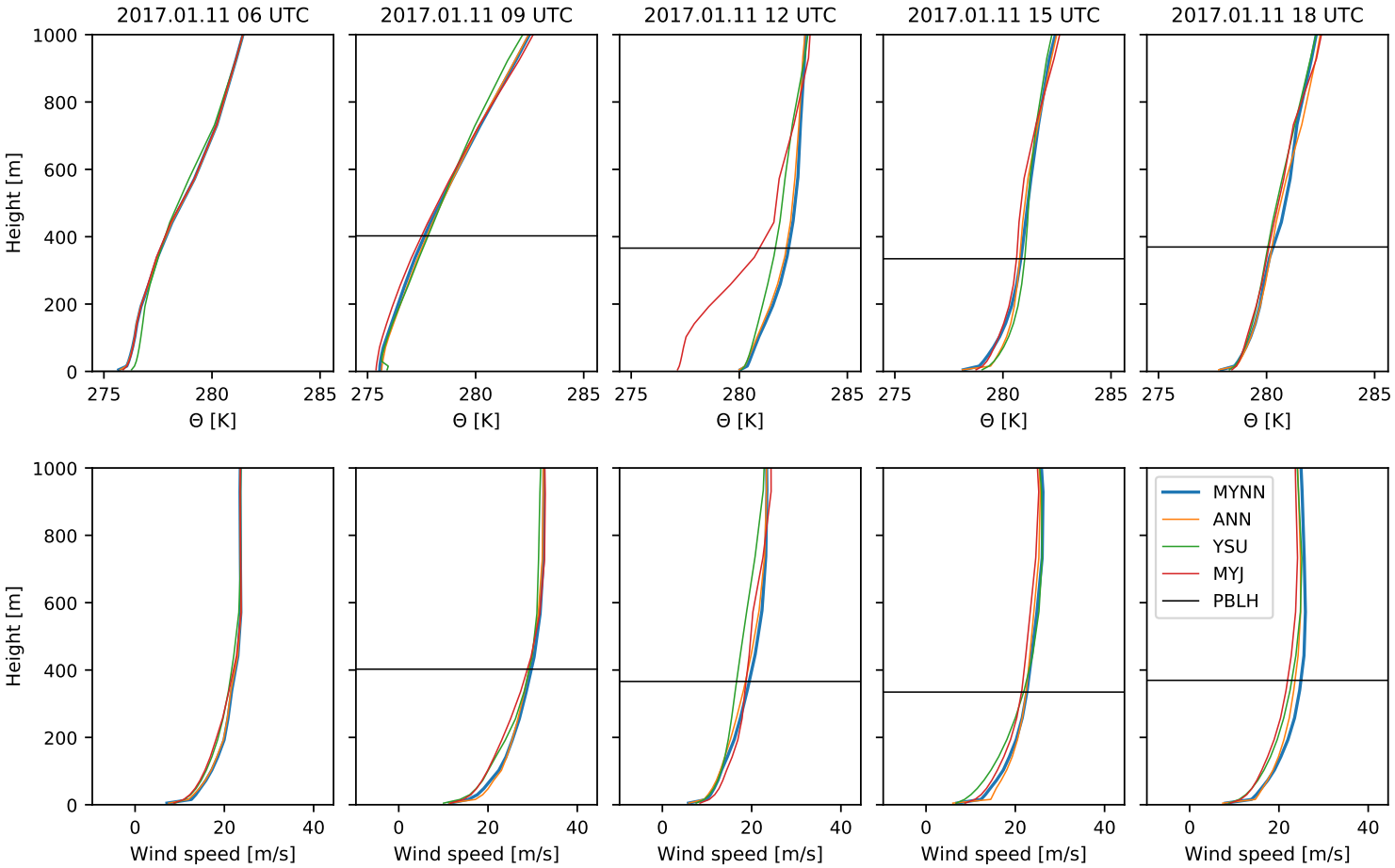


Figure E.18: Profiles of Θ and wind speed for the location shown in Figure E.17. The location is randomly selected. The profiles are from the simulation initiated at 2017.01.11 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

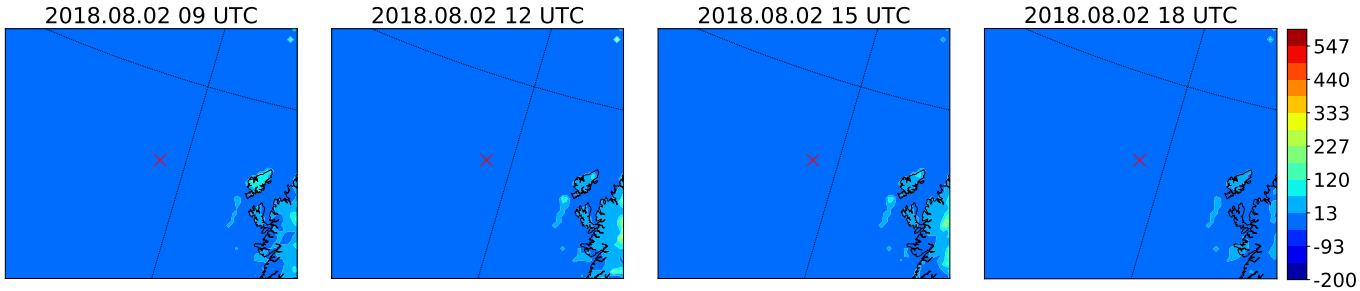


Figure E.19: Surface sensible heat fluxes for the location of the profiles shown in Figure E.20. The plots are based on the simulation initiated at 2018.08.02 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

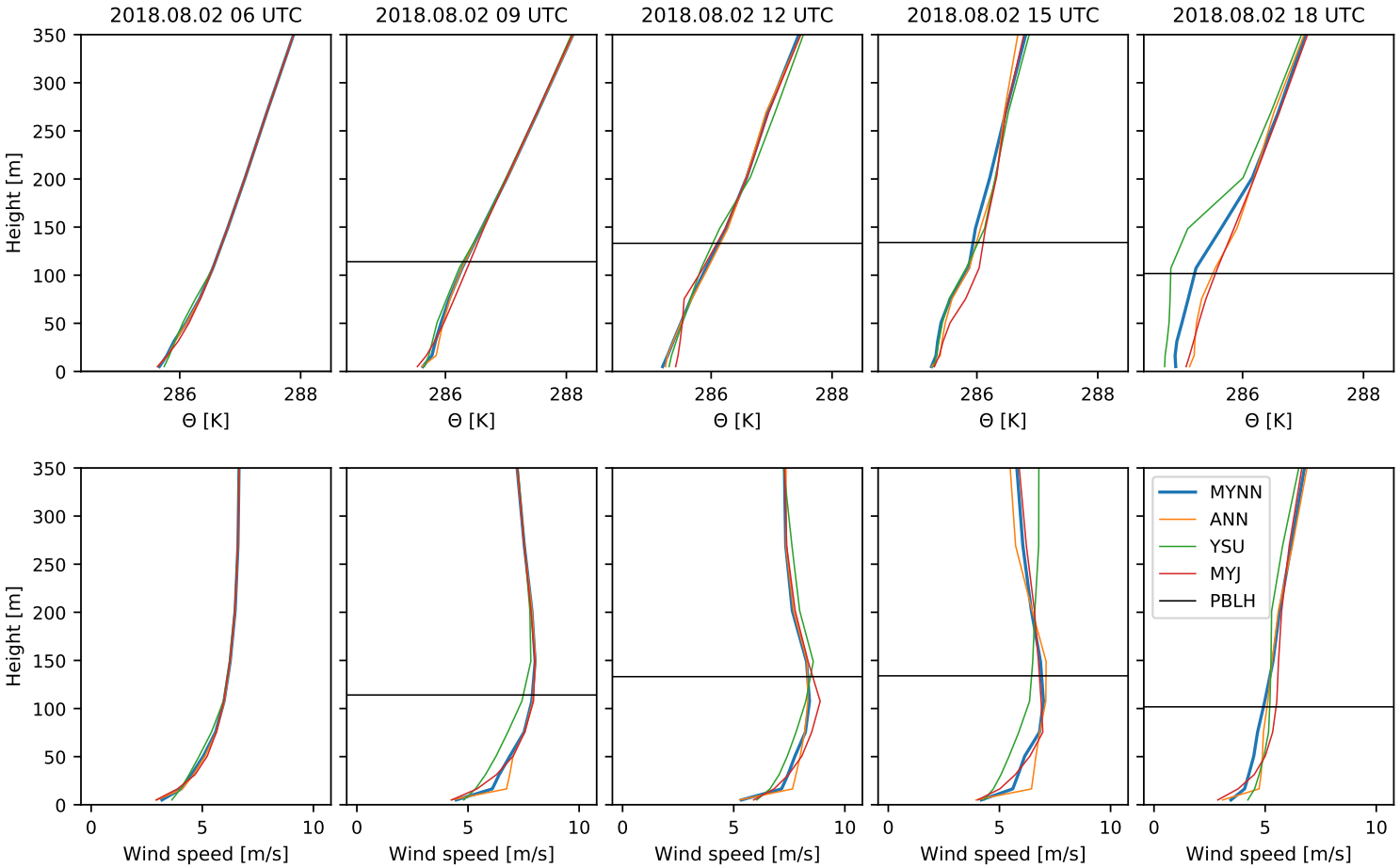


Figure E.20: Profiles of Θ and wind speed for the location shown in Figure E.19. The location is randomly selected. The profiles are from the simulation initiated at 2018.08.02 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

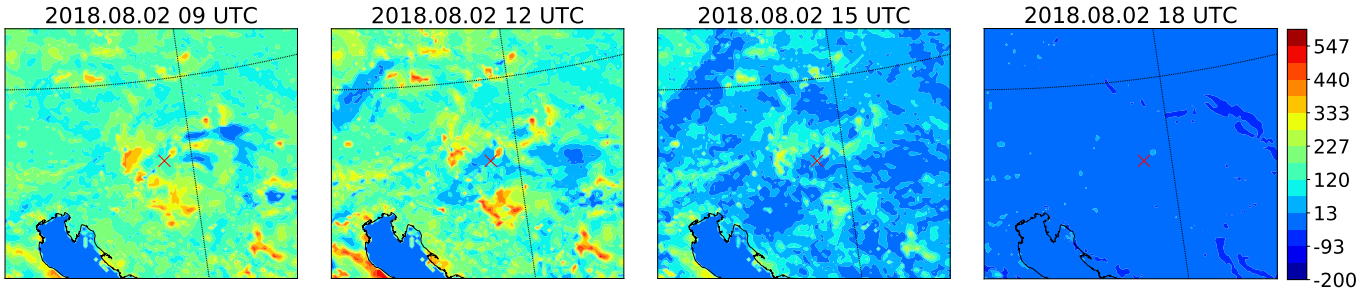


Figure E.21: Surface sensible heat fluxes for the location of the profiles shown in Figure E.22. The plots are based on the simulation initiated at 2018.08.02 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

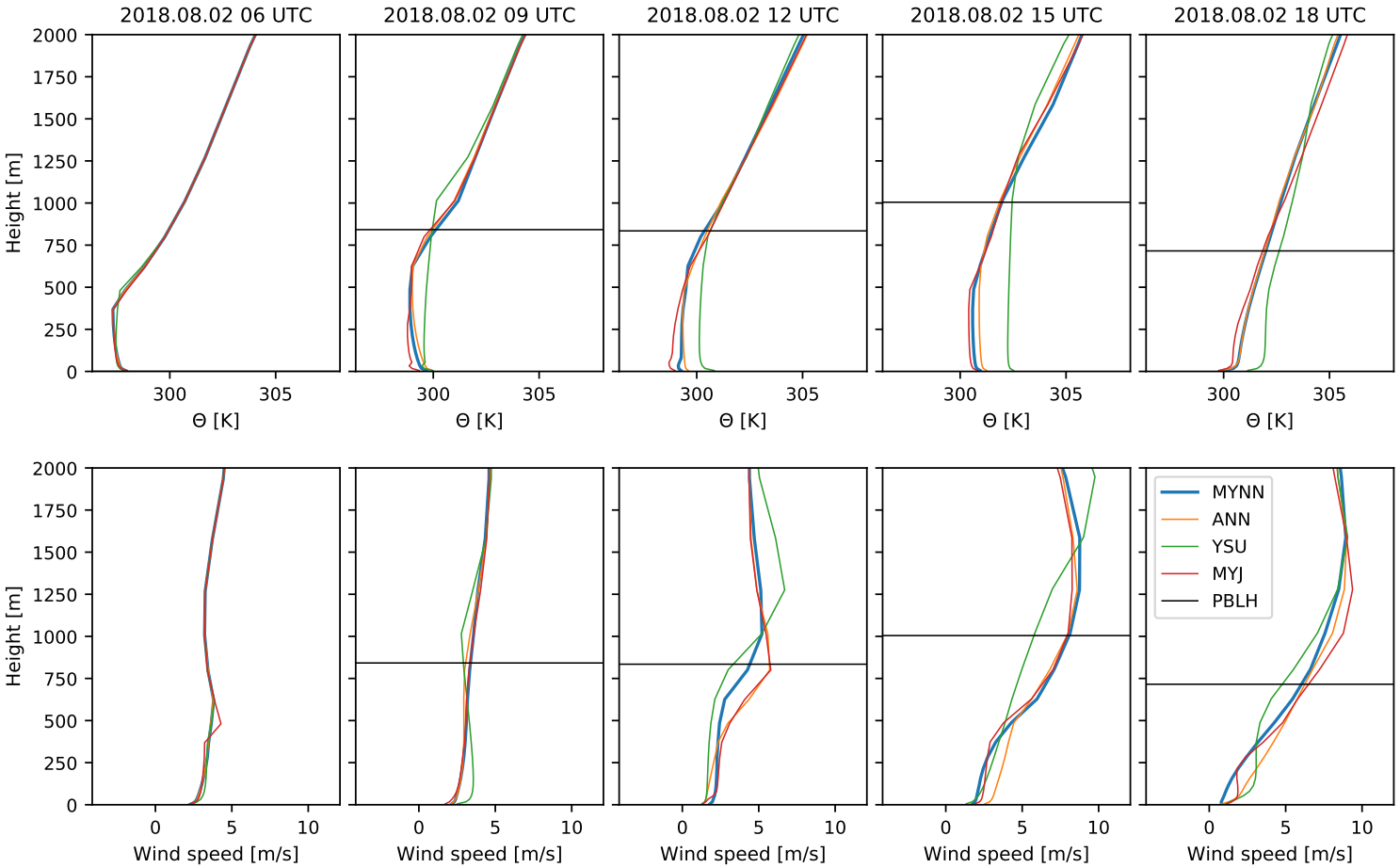


Figure E.22: Profiles of Θ and wind speed for the location shown in Figure E.21. The location is randomly selected. The profiles are from the simulation initiated at 2018.08.02 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

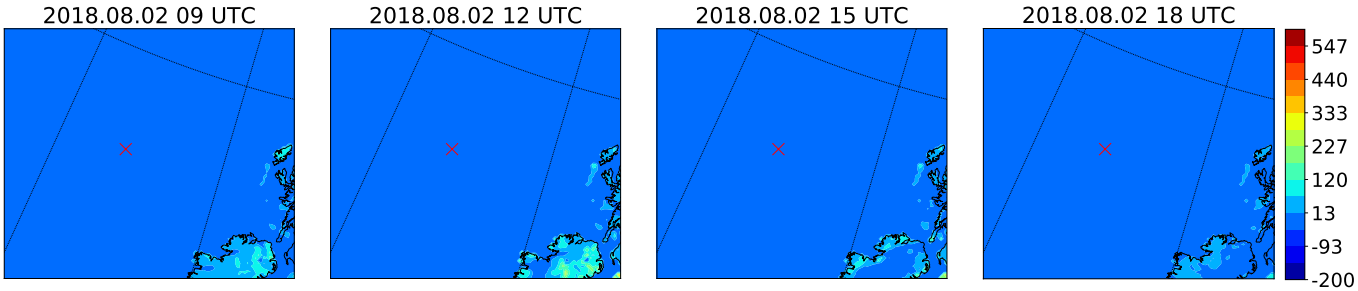


Figure E.23: Surface sensible heat fluxes for the location of the profiles shown in Figure E.24. The plots are based on the simulation initiated at 2018.08.02 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

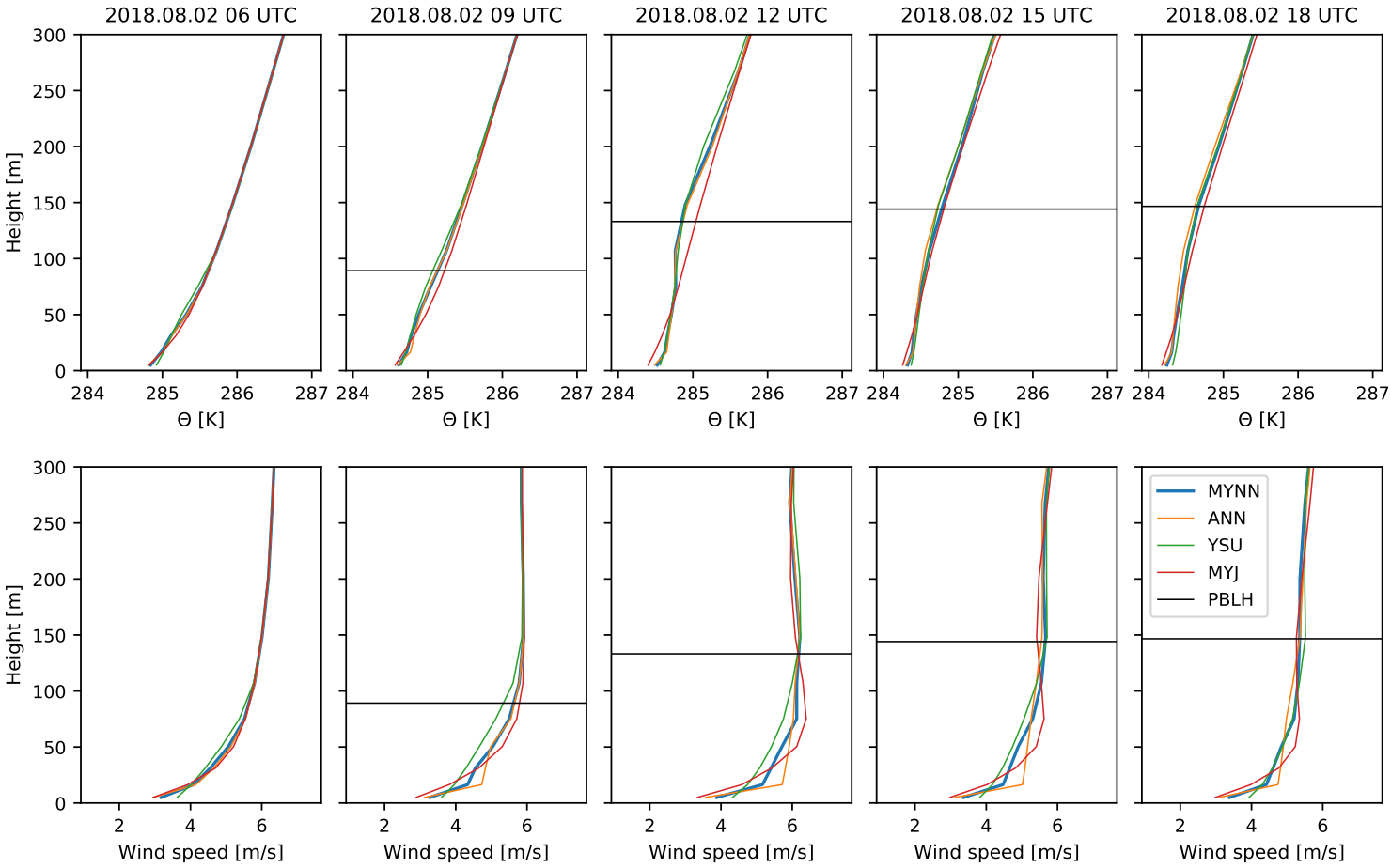


Figure E.24: Profiles of Θ and wind speed for the location shown in Figure E.23. The location is randomly selected. The profiles are from the simulation initiated at 2018.08.02 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

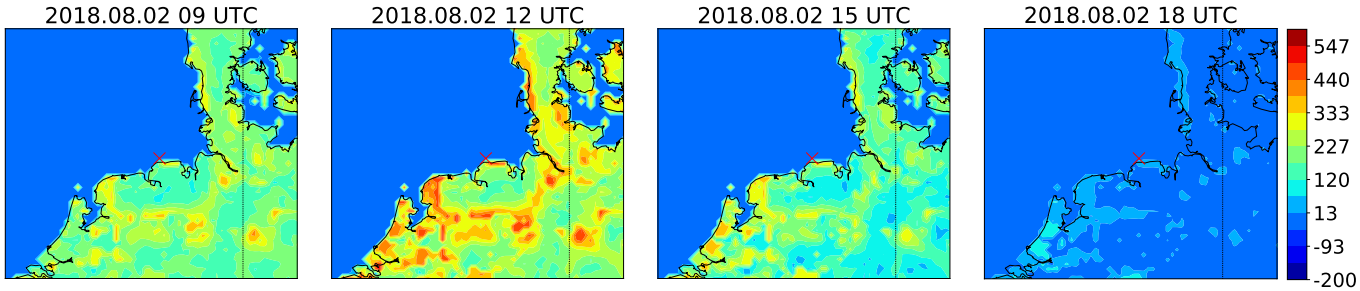


Figure E.25: Surface sensible heat fluxes for the location of the profiles shown in Figure E.26. The plots are based on the simulation initiated at 2018.08.02 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

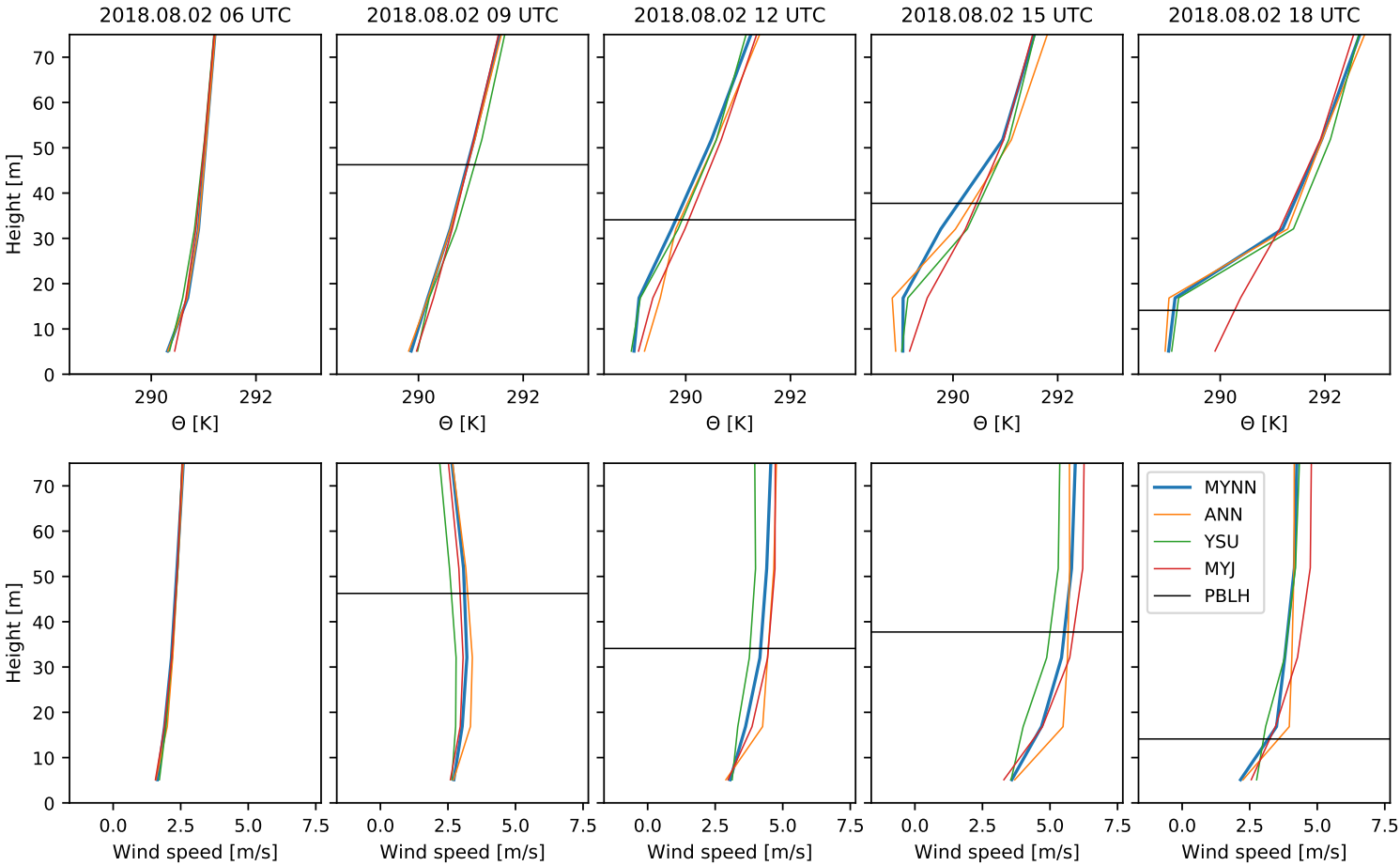


Figure E.26: Profiles of Θ and wind speed for the location shown in Figure E.25. The location is randomly selected. The profiles are from the simulation initiated at 2018.08.02 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

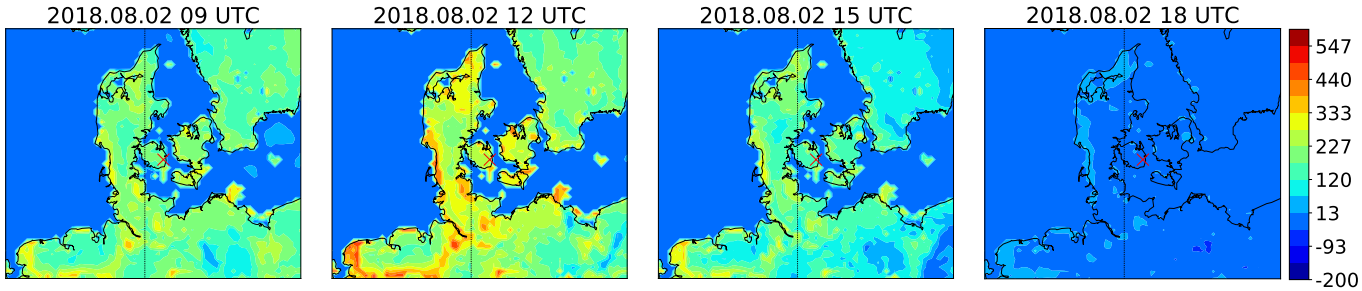


Figure E.27: Surface sensible heat fluxes for the location of the profiles shown in Figure E.28. The plots are based on the simulation initiated at 2018.08.02 06 UTC. The times thus correspond to the fluxes after 3, 6, 9 and 12 hours (the fluxes at initial time is not available in the output file from WRF). The location for the profiles is marked with the black ×.

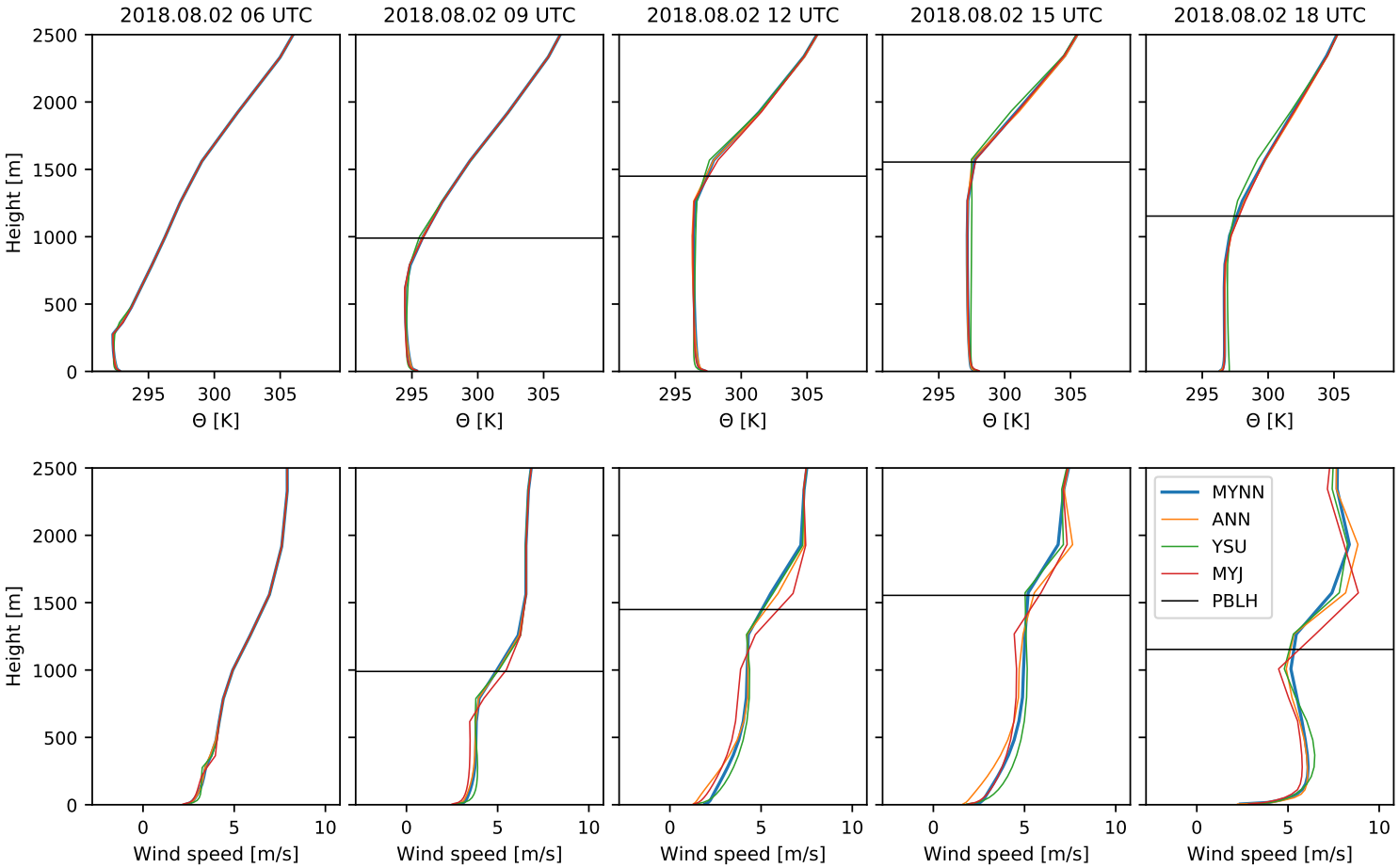


Figure E.28: Profiles of Θ and wind speed for the location shown in Figure E.27. The location is randomly selected. The profiles are from the simulation initiated at 2018.08.02 06 UTC and are valid after 0, 3, 6, 9, and 12 hours, respectively. The planetary boundary layer height, *pblh* is the one predicted by the MYNN scheme (not available for the initial time step).

List of Figures

List of Tables