Department of Physics and Astronomy
University of Heidelberg

Master Thesis in Physics
submitted by

Eric Niklas Willi Volkmann

born in Waldems Esch (Germany)

2023

BOLD observation model for the dynamical systems reconstruction of fMRI time series

This Master Thesis has been carried out by Eric Volkmann at the
Central Institute of Mental Health Mannheim
under the supervision of
Prof. Dr. Daniel Durstewitz

Abstract

The reconstruction of dynamical systems underlying observed time series is a common task across many scientific disciplines. One tool developed to accomplish this task is the piecewise linear recurrent neural network (PLRNN) with its universal approximation capabilities trained using backpropagation through time (BPTT) and teacher forcing. Because of it is mathematical tractable, a trained PLRNN allows for a simple DS analysis after it has been successfully trained on a system.

This thesis aims to apply this PLRNN training framework to functional magnetic resonance imaging (fMRI), which measures brain activity via the bold-oxygen-level dependent contrast, i.e. the BOLD signal. To this end, a BOLD observation model, that models the hemodynamic response in the BOLD signal, is developed in this thesis as well as the inversion algorithm for the BOLD observation needed for teacher forcing. The PLRNN with the BOLD observation model is then first benchmarked on an artificial Lorenz63 benchmark and then applied to the LEMON ("Leipzig Study for Mind-Body-Emotion Interactions") dataset. While the benchmark results appear very promising, the application to real fMRI time series remains challenging.

Zusammenfassung

Die Rekonstruktion dynamischer Systeme aus observierten Zeitreihen ist über viele wissenschaftliche Disziplinen hinweg eine häufige Aufgabe. Ein Werkzeug um dieses Problem zu lösen sind stückweise lineare rekurrente neuronale Netzwerke (PLRNN) mit ihren universalen Approximationsfähigkeiten. Sie werden mittels des Algorithmus "backpropagation through time (BPTT)" und "teacher forcing" (TF) trainiert. Aufgrund ihrer mathematischen Form erlauben PLRNNs eine einfache Analyse der rekonstruierten Systeme nach dem Training.

In dieser Arbeit wird das PLRNN Trainingsframework auf Zeitreihen aus der funktionelle Magnetresonanztomografie (fMRT), welche die Gehirnaktivität basierend auf der Blutoxygenierung, dem BOLD Signal, misst, angewendet. Dazu wird eine BOLD Beobachtungsmodell entwickelt, welches die neurovaskuläre Kopplung im BOLD Signal modelliert, und der dazugehörige Umkehralgorithmus, der für das Training mit TF notwendig ist. Das PLRNN mit dem BOLD Beobachtungsmodell wird zunächst auf einem künstlichen Datensatz basierend auf dem Lorenz63 getestet und dann auf den LEMON ("Leipzig Study for Mind-Body-Emotion Interactions") Datensatz angewandt. Während die Ergebnisse auf dem Lorenz System sehr vielversprechend sind, zeigen sich weiterhin viele Schwierigkeiten auf dem LEMON Datensatz. Diese hängen insbesondere damit zusammen, dass die fMRT Zeitreihen sehr kurz sind und Modelle daher schwierig zu evaluieren sind.

Contents

List of Figures

List of Tables

Chapter 1

Dynamical Systems

This chapter on dynamical systems is meant to provide a comprehensive introduction to the mathematical framework and definitions used to describe and characterize continuous and discrete dynamical systems and establish key analytical tools such as stability analysis and Lyapunov exponents. In this chapter, I adhere to the definitions and theorems outlined by [1] unless stated otherwise. For detailed proofs and additional references, please refer to their work.

1.1 Introduction to Dynamical Systems

Definition 1.1. Let $\mathscr{S}$ be a system (physical, mathematical, biological etc.) described by a finite number of state variables $x = (x_1, \cdots, x_n) \in \mathbb{R}^n$. At time t_0 the state of the system is specified by the values of the state variables $x(t_0) = (x_1(t_0), \cdots, x_n(t_0))$. If there is an operator $g_t : U \subseteq \mathbb{R}^n \to U$ such that:

1. $g_t(x(0)) = x(t)$
2. $(g_t \circ g_s)(x(0)) = (g_{t+s})(x(0)) = x(t+s)$
3. $g_0(x(t)) = x(t)$

then U and g_t define a *deterministic dynamical system* of finite dimension n.

The set $U \subseteq \mathbb{R}^n$ of possible state space values is the *phase space* or *state space* of the System $\mathscr{S}$. The phase space of a system with n state variables can be the entire space $\mathbb{R}^n$, a subset of $\mathbb{R}^n$ or a lower dimensional submanifold of dimension $m < n$.

The study of a dynamical system involves the definition of a phase space U and an evolution equation to obtain the operator g_t and analyze its properties.

1.1.1 Continuous Dynamical Systems

Let $f(x,t)$ be a continuous function defined on an open set $U \times I \in \mathbb{R}^n \times \mathbb{R}$. Define the system of ordinary differential equations of order n

$$\frac{dx}{dt} = f(x,t) \tag{1.1}$$

where $x = (x_1, \cdots, x_n) \in U$, $f(x,t) = (f_1(x,t), \cdots, f_n(x,t)) : U \times I \to U$ and $t \in I \subseteq \mathbb{R}$. The dependent variables are the components of the vector $x \in U$ and t is the independent variable. The set U is the phase space of the system, the set $U \times I$ is referred to as the *extended phase space*.

The solution of the differential equation 1.1 is the map $\phi(t) : I \to U$ such that

$$\frac{d\phi}{dt} = f(\phi(t), t) \tag{1.2}$$

The solution of the initial value problem with $x_0 \in U$

$$\dot{x} = f(x) \tag{1.3}$$

$$x(t_0) = x_0 \tag{1.4}$$

is also often written as $\phi(t; t_0)$, $\phi(t; x_0)$ or $\phi_t(x_0)$ to highlight the dependence of the solutions on the initial conditions.

If the function f fulfills $f(x+y,t) = f(x,t) + f(y,t)$ for every $x, y \in U$ and $t \in I$, and if $\phi_1(t)$ and $\phi_2(t)$ are both solutions with $\phi_1(t) \neq \phi_2(t)$, then $\psi(t) = \phi_1(t) + \phi_2(t)$ is also a solution of 1.1. In this case, the system of equations is linear.

If the function f does not explicitly depend on time, the system of differential equations $\dot{x} = f(x)$ is *autonomous*. Non-autonomous differential equations are straightforwardly reduced to autonomous ones by increasing the phase space dimension by 1 and introducing the new independent variable $\tau = t$ and the equation $\frac{d\tau}{dt} = 1$ (c.f. [2, p. 9]).

A system of autonomous differential equations defines a *vector field* X in phase space, also called a *phase flow*: the components of a vector field X, associated with the coordinates $x_1, \cdots x_n$, are the functions $f_1(x), \cdots, f_n(X)$. The vector $f(x) = (f_1(x), \cdots, f_n(x))$ is a tangent vector to the image of $\phi(t)$ in phase space.

In the following I give some core definitions for (continuous) dynamical systems. For the upcoming definitions, let $\phi_t(x)$ be the solution to the equation $\dot{x} = f(x)$ with $x \in \mathbb{R}^n$

Definition 1.2 (Orbits)**.** The *orbit* of a point $x_0 \in U$ is the set $\mathscr{O}_{x_0} = \{x \in U : x = \phi_t(x_0), t \in I\}$. A function $\phi(t)$ is a periodic solution if there is a constant $T > 0$ such that

$$\phi(T) = \phi(t + T) \tag{1.5}$$

for all $t \in \mathbb{R}$. A periodic solution of a differential equation is a *periodic orbit.*

Definition 1.3 (Fixed points)**.** A point x_0 is called a *fixed point* if $\phi_t(x_0) = x_0 \ \forall t$, i.e. the orbit of x_0 is only itself $\mathscr{O} = \{x_0\}$. This implies that the derivative of f vanishes at x_0: $\dot{f}(x_0) = 0$.

Definition 1.4 (Nullclines)**.** For the components f_i of f, the equations $f_i(x) = 0$ define curves called *nullclines.* On the nullcline $f_i(x) = 0$, the vector field component along the direction of the coordinate x_i vanishes.

Definition 1.5 (ω-and α-limiting sets)**.** A point $y \in \mathbb{R}^n$ is in the *ω-limit* set of x, $\omega(x)$, if there is a sequence $\{t_n\}_{n\in\mathbb{N}}$, with $t_n \to +\infty$, such that

$$\lim_{n\to\infty} \phi_{t_n}(x) = y. \tag{1.6}$$

Similarly, if there is a sequence $\{t_n\}_{n\in\mathbb{N}}$ that instead has $t_n \to -\infty$ such that $\lim_{n\to\infty} \phi_{t_n}(x) = y$, then y is in the *α-limit set* of x, $\alpha(x)$.

Fixed points and periodic orbits are simultaneous α-and ω-limit sets. α-and ω-limit sets are closed invariant subsets with respect to the time evolution ϕ_t. Based on these definitions we can define an attractor of a dynamical system.

Definition 1.6 (Attractor)**.** A closed invariant subset $\mathcal{A}$ is called an *attracting set* of a dynamical system if there is a neighborhood $\mathcal{U}$ of $\mathcal{A}$ such that for all $x \in \mathcal{U}$

$$\phi_t(x) \in \mathcal{U} \ \forall t \qquad\qquad \lim_{t\to\infty} \phi_t(x) \in \mathcal{A} \tag{1.7}$$

An attractor $\mathcal{A}$ is an attracting set which has a *basin of attraction*

$$\mathcal{B}(\mathcal{A}) = \{x \in U | \omega(x) \subset \mathcal{A}\} \tag{1.8}$$

with non-zero (Lebesgue) measure.

Often the qualifier "strange" is added to the term attractor in literature. There is however no strict definition of this qualifier. It usually refers to attractors with properties such as: (i) unusual geometry (ii) dimension is fractal, not an integer (iii) chaotic dynamics, trajectories separate exponentially fast (at least initially).

So far we have not given any thought to the existence of a solution to Eq. 1.1. The theory of ordinary differential equations (ODE) provides us the following theorems about existence and uniqueness of solutions.

Theorem 1.7 (Local existance and uniqueness of solutions of ordinary differential equations)**.** *Consider the differential equation 1.1, where $f(x,t)$ is a Lipschitz function in an open set $U \times I \subseteq \mathbb{R}^n \times \mathbb{R}$, that is, $|f(x,t) - f(y,t)| \leq k|x-y|$ $\forall x, y \in U, t \in I$ and $k \in \mathbb{R}$ is finite. Then, there are constants $\tau_1, \tau_2 > 0$ and a function $\phi(t) : I_1 \to \mathbb{R}^n$ solution of the differential equation 1.1 where $\phi(t_0) = x_0$, $x_0 \in U, t_0 \in I_1 \subset I$, and $I_1 = [t_0 - \tau_1, t_0 + \tau_1]$.*

If there is another solution $\phi_2(t) : I_2 \to \mathbb{R}^n$, with $\phi_2(t_0) = x_0$, and $I_2 = [t_0 - \tau_2, t_0 + \tau_2]$, then $\phi(t) = \phi_2(t)$ in the interval $[t_0 - \tau_3, t_0 + \tau_3]$ where $\tau_3 = \min(\tau_1, \tau_2)$.

This existence and uniqueness theorem is only valid locally in time. In the case where the phase space is compact and the vector field is continuously differentiable, there is a stronger theorem that guarantees solutions are defined for all times.

Theorem 1.8 (Global existance and uniqueness of solutions of ordinary differential equations)**.** *If there exists a bounded set D in phase space, positively invariant under the flow ϕ_t, meaning $\phi_t(D) \subseteq D$, then solutions with initial conditions in D are globally defined in time.*

From the existence and uniqueness theorem 1.7 for autonomous equations, it follows that the solutions of differential equations define a one-parameter group, the time t-(local) group of diffeomorphisms. That is

1. $\phi_t(x_0)$ is of class C^r with $r \geq 1$
2. $\phi_0(x_0) = x_0$
3. $\phi_{t+s}(x_0) = \phi_t(\phi_s(x_0))$, $\forall t, s \in I$, where $I = [t_0 - \tau, t_0 + \tau]$.

This means that the flow ϕ_t satisfies the conditions required in 1.1 to define a dynamical system on U.

1.1.2 Stability of Fixed points

Definition 1.9 (Lyapunov Stability)**.** Let x_0 be a fixed point of the DE $\dot{x} = f(x)$, with $x_0 \in \mathbb{R}^n$, and let $\mathcal{U}(x_0)$, $\mathcal{V}_0(x_0)$ and $\mathcal{V}_1(x_0)$ be neighborhoods of x_0. The fixed point x_0 is Lyapunov stable if:

1. There is a neighborhood $\mathcal{U}(x_0)$ of x_0 such that the solutions $\phi_t(x)$ with $x \in \mathcal{U}(x_0)$ are defined $\forall t \geq 0$
2. For every sufficiently small neighborhood $\mathcal{V}_0(x_0) \subset \mathcal{U}(x_0)$ there is another neighborhood $\mathcal{V}_1(x_0) \subset \mathcal{V}_0(x_0)$ such that every solution $\phi_t(x)$ with $x \in \mathcal{V}_1(x_0)$ is contained in $\mathcal{V}_0(x)$.

In addition, if $\phi_t(x) \to x_0$ for $t \to \infty$, then x_0 is *asymptotically stable.* If a fixed point is not Lyapunov stable, then it is *unstable.*

In the definition of Lyapunov stability, the condition $\phi_t(x) \to x_0$ for $t \to \infty$ is insufficient to guarantee the asymptotic stability of x_0 because $\phi_t(x)$ can have large values and leave the neighborhood of x_0 before converging.

In general, for nonlinear DEs, analyzing the stability of a fixed point can be difficult. At times, it is possible to find the *Lyapunov function* and then use the Lyapunov theorem to get information about the fixed point stability.

Definition 1.10. Let $\mathcal{V}(x_0)$ be a neighborhood of the fixed point x_0 of a DE and $V(x) : \mathcal{V}(x_0) \to \mathbb{R}$ a continuous, differentiable function on $\mathcal{V}(x_0)$. If $V(x_0) = 0$ and $V(x) > 0$ for $x \in \mathcal{V}(x_0) \setminus \{x_0\}$, then $V(x)$ is a *Lyapunov function.*

The intuition of Lyapunov functions comes from the 2d case. In the neighborhood of the fixed point x_0, the projections of the level curves of the Lyapunov function $z = V(x)$ on the phase space are closed curves. On these curves, the vector field can only be tangent, so point inside or outside these curves. The following theorem generalizes this to higher dimensions.

Theorem 1.11 (Lyapunov theorem)**.** *Let $x_0 \in \mathbb{R}^n$ be an isolated fixed point of the differential equation $\dot{x} = f(x)$. Let $\mathcal{V}(x_0)$ be a neighborhood of x_0 with the Lyapunov function $V : \mathcal{V}(x_0) \to \mathbb{R}$.*

If $V(x)$ obeys the condition $\frac{dV}{dt} \leq 0$ in $\mathcal{V}(x_0) \setminus \{x_0\}$, then the fixed point x_0 is asymptotically stable.

There is however no general method to construct a Lyapunov function. A different approach is to analyze the stability of a fixed point is through the linearization of the vector field around the fixed point. For that define the Jacobian matrix at the point x_0

$$\boldsymbol{J}^t(x_0) = \frac{df^t(x)}{dx}(x_0). \tag{1.9}$$

Then we can perturb the system around the fixed point by δx and linearize the equation. This linear system is then much easier to analyze. It turns out that the behavior of the system depends on the eigenvalues of the Jacobian. This will be further discussed in section 2.4.

1.1.3 Difference Equations as Dynamical Systems

Now we will shift our view towards the discrete case of dynamical systems. Consider the following equation

$$x_{n+1} = f(x_n) \tag{1.10}$$

where f is some continuous function and $n \in \mathbb{N}$. The evolution of the system is given by feeding the output at step n, x_n, back to the function f to obtain the next state of the system. Unlike systems defined by differential equations 1.1, which define a continuous group of phase space transformations, the parameter of systems defined by difference equations 1.10 is discrete. The solutions define a discrete transformation group.

For difference equations, the notion of a vector field is no longer valid and the definitions of fixed point and stability must be adapted. The definitions of limit sets and attractors extend very naturally to discrete system. Note it is still common to speak of phase flows in discrete dynamical systems, although it is technically abuse of language.

Definition 1.12. Consider a difference equation $x_{n+1} = f(x_n)$, where $f : I \to I$ is a continuous function and $I \subseteq \mathbb{R}^n$. Let $f^k(x) = f(f^{k-1})(x)$ be the iterate of order k. A point $x^* \in \mathbb{R}^n$ in the phase space of the difference equation is called *fixed point* of period k if

$$f^k(x^*) = x^* \tag{1.11}$$

The orbit of a point $x_0 \in I$ is the set $\mathscr{O}_{x_0} = \{x_0, f(x_0), f^2(x_0), \cdots\}$. An important result for fixed points in this context is given by the following theorem

Theorem 1.13 (Brouwer's fixed point theorem)**.** *If $f : I \to I$ is continuous and surjective on $I \subset \mathbb{R}^n$ and I is compact and convex, then f has at least one fixed point $x^* \in I$.*

Next we need a notion of stability for difference equations:

Definition 1.14. A period k fixed point $x^* \in I$ of the discrete DS is *asymptotically stable*, if there is a neighborhood $\mathcal{V}(x^*) \subset I$ such that, for every $x \in \mathcal{V}(x^*)$, $\lim_{n\to\infty} f^{nk}(x) = x^*$.

If $f : I \to I$ is differentiable in the neighborhood of a period k fixed point $x^* \in I$, then we can make statements about the stability of x^* quite easily. In the one dimensional case $I \subset \mathbb{R}$ this can be seen most easily:

Make a linear approximation of f around the fixed point x^*

$$f(x) \approx f(x^*) + f'(x^*)(x - x^*) \tag{1.12}$$

Thus, using $f(x^*) = x^*$

$$\begin{aligned} & x_{n+1} - x^* = f(x_n) - x^* \approx f(x^*) + f'(x^*)(x_n - x^*) - x^* = f'(x^*)(x_n - x^*) \\ \Rightarrow & \frac{x_{n+1} - x^*}{x_n - x^*} = f'(x^*) \end{aligned} \tag{1.13}$$

which means that the derivative measures the rate at which successive iterations approach the fixed point. If $|f'(x^*)| < 1$, then the fixed point is asymptotically stable. If $|f'(x^*)| > 1$ the fixed point is unstable. If $|f'(x^*)| = 0$ the fixed point is *superstable*. Only at $|f'(x^*)| = 1$ do we need more information such as higher order derivatives to decide stability.

The Lyapunov stability concept remains valid for difference equations

Definition 1.15 (Lyapunov Stability)**.** A period k fixed point x^* of a difference equation is Lyapunov stable if, for every small enough neighborhood $\mathcal{V}_0(x^*)$ of x^*, there is a neighborhood $\mathcal{V}_1(x^*) \subset \mathcal{V}_0(x^*)$ such that $f^{nk}(x) \in \mathcal{V}_0(x^*)$, for every $x \in \mathcal{V}_1(x^*)$, $n \in \mathbb{N}$.

In higher dimensions the stability of difference equations depends on the eigenvalues Jacobian matrix, which can be used for a linear approximation around the fixed point. If the eigenvalues of the Jacobian are within the unit circle of the complex plain, so

$|\lambda_i| < 1$, the fixed point is Lyapunov stable. The fixed points can be further classified depending on whether the eigenvalues are real, purely imaginary etc. For details see [1, Ch 1.4, Appendix A.3]

1.2 Lyapunov Exponents

In many dynamical systems it can be observed that initial conditions close in phase space will diverge or contract exponentially in time. Furthermore, it can be observed that such perturbations of an initial condition behave differently along different axis. Along some axis the difference will grow exponentially and along others it will contract. In the case of exponential growth this property is called *sensitivity to initial conditions*, a characteristic of chaotic dynamics. This can be formalized by the so-called *Lyapunov exponents (LE)*, which together make up the *Lyapunov spectrum* of a system.

Let us first consider the one-dimensional case

$$x_{n+1} = f(x_n)$$

with $x_n \in I \subseteq \mathbb{R}$. The mean value theorem gives us

$$|x_{n+1} - y_{n+1}| = |f(\xi)| \cdot |x_n - y_n| \tag{1.14}$$

for some $\xi \in I$. Then

$$\ln|x_{n+p} - y_{n+p}| = p\frac{1}{p}\ln|f^{p'}(\xi_p)| + \ln|x_n - y_n| \tag{1.15}$$

Now define

$$\lambda := \lim_{p\to\infty} \frac{1}{p}\ln|f^{p'}(\xi_p)|. \tag{1.16}$$

For large enough p, we have

$$|x_{n+p} - y_{n+p}| \simeq e^{\lambda p}|x_n - y_n| \tag{1.17}$$

If $\lambda < 0$, the dynamical system is contracting and has no sensitivity to the initial conditions. If $\lambda > 0$, this dynamical system is locally expansive and sensitive to the initial condition.

Therefore, in analogy to Eq. 1.16, in discrete one-dimensional dynamical systems the Lyapunov exponent is defined as

$$\lambda(x) := \lim_{n\to\infty} \frac{1}{n} \ln |f^{n'}(x)| = \lim_{n\to\infty} \frac{1}{n} \sum_{p=0}^{n-1} \ln |f'\left(f^p(x)\right)|. \tag{1.18}$$

Now we can try to expand this idea to higher dimensions. Consider the m-dimensional discrete dynamical system

$$x_{n+1} = F(x_n) \tag{1.19}$$

where $x_n \in \mathbb{R}^m$. Let x_0 be an initial condition and δx_0 a small perturbation. Then we make the following linear approximation

$$F(x_0 + \delta x_0) = F(x_0) + \boldsymbol{J}(x_0)\delta x_0 + \mathcal{O}(\|\delta x_0\|^2) \tag{1.20}$$

$$\boldsymbol{J}(x_0)\delta x_0 \approx F(x_0 + \delta x_0) - F(x_0) \tag{1.21}$$

Thus, the time evolution of the perturbation is given as $\delta x_{n+1} = J(x_0)\delta x_n$. Due to linearity this can be written as

$$\delta x_n = J(x_0) \cdots J(x_0)\delta x_0 = J^n(x_0)\delta x_0. \tag{1.22}$$

If $|J^n(x_0)\delta x_0| \simeq |e^{n\lambda}\delta x_0|$, i.e. δx_0 is an eigenvector of $J^n(x_0)$, one can define a Lyapunov exponent of the system as

$$\begin{aligned}\lambda(x_0) &= \lim_{n\to\infty}\frac{1}{n}\ln\left|J^n(x_0)\frac{\delta x_0}{|\delta x_0|}\right| \\ &= \lim_{n\to\infty}\frac{1}{2n}\ln\left|\frac{\delta x_0}{|\delta x_0|}^{\dagger} J^{n\dagger}(x_0)J^n(x_0)\frac{\delta x_0}{|\delta x_0|}\right| \\ &= \lim_{n\to\infty}\frac{1}{2n}\ln\left|\frac{\delta x_0}{|\delta x_0|}^{\dagger} H^n\frac{\delta x_0}{|\delta x_0|}\right| \qquad (1.23)\end{aligned}$$

$$(1.24)$$

where † is the adjoint and $H^n(x_0) := J^{n\dagger}(x_0)J^n(x_0)$. $H^n(x_0)$ is a Hermitian matrix, its eigenvalues $\{\lambda_1, \cdots, \lambda_m\}$ are real and non-negative. We can then take perturbations parallel to the eigenvectors of $H^n(x_0)$. This way we get m different limits, the logarithms of the eigenvalues of $H^n(x_0)$. If at least one of the eigenvalues is positive the dynamical system has sensitivity to initial conditions because the $|e^{n\lambda}\delta x_0|$ will below up exponentially fast.

These ideas are formalized and the limits guaranteed by the following theorem

Theorem 1.16 (Oseledets Theorem)**.** *Let $f : \mathbb{R}^m \to \mathbb{R}^m$ be a differentiable function and ρ the density of an invariant measure for the dynamical system $x_{n+1} = f(x_n)$. Consider the matrix*

$$T_x^n = Df(f^{n-1}(x))Df(f^{n-2}(x))\cdots Df(x) \qquad (1.25)$$

and its adjoint $T_x^{n\dagger}$. Then, the limit

$$\lim_{n\to\infty}(T_x^{n\dagger}T_x^n)^{\frac{1}{2n}} = \Lambda_x \qquad (1.26)$$

exists for almost every $x \in \mathbb{R}^m$. The logarithm of the eigenvalues of Λ_x are the Lyapunov exponents of the dynamical system $x_{n+1} = f(x_n)$. Arrange the eigenvalues in descending order: $\lambda_1 > \lambda_2 > \cdots > \lambda_m$. Associated with these eigenvalues of Λ_x, there are vector subspaces E_x^i with $i = 1, \cdots, m$, such that

$$\mathbb{R}^m = E_x^1 \subset E_x^2 \subset \cdots$$

and for which, if $v_i \in E_x^i \setminus E_x^{i+1}$,

$$\lim_{n\to\infty}\frac{1}{n}\ln\|T_x^n v_i\| = \lambda_i$$

For $v_1 \in E_x^1 \setminus E_x^2$, then λ_1 is the largest eigenvalue of Λ_x.

For differential equations, Lyapunov exponents are calculated with the solutions evaluated at discrete values of the continuous time variable t and the above theorem holds.

In general, the sum of Lyapunov exponents in dissipative systems is negative. In conservative systems, the sum is 0. In the case of strange attractors and limit cycles, there is always a zero Lyapunov exponent.

Obtaining Lyapunov exponents from their definition is numerically impractical because they relate to the singular values of $T_t^{\frac{1}{t}}$ for large t. [3] presents an algorithm that allows for the numerical estimation of the Lyapunov exponents. [4] present an algorithm specifically for RNNs, which I will use in the evaluation section of this work.

Chapter 2

Recurrent Neural Networks

This chapter on recurrent neural networks (RNNs) provides a detailed exploration of RNNs and their applications, particularly in the context of fMRI time series modeling. I will begin by defining RNNs by their state equation and layout training them with the backpropagation through time algorithm. Then I will discuss various challenges in training and proposed solutions. Next, I will introduce piecewise linear RNNs and explain the basics of the BOLD fMRI signal and modeling the hemodynamic response. Lastly, I propose a BOLD observation model to replace the standard linear observation model when training on fMRI time series.

2.1 Introduction to Recurrent Neural Networks

Recurrent neural networks are a neural network architecture designed for processing sequential data, i.e. a sequence of values $x_1, ..., x_T$ with $x_t \in \mathbb{R}^N$ for each time step $t \in [\![1, T]\!]$. They are defined by a recursive function evolving an internal state $z_t \in \mathbb{R}$ at discrete time steps:

$$z_{t+1} = F_\Theta(z_t, s_t) = \Phi(\boldsymbol{W} z_t + \boldsymbol{C} s_t + b) \tag{2.1}$$

where $s_t \in \mathbb{R}^K$ are external inputs, $\boldsymbol{W} \in \mathbb{R}^{M \times M}$ is the hidden-to-hidden weight matrix, $\boldsymbol{C} \in \mathbb{R}^{M \times K}$ is the input-to-hidden weight matrix and $b \in \mathbb{R}^M$ is a bias vector. Φ is a nonlinear activation function applied to each element such as the Rectified Linear Unit ReLU or the hyperbolic tangent tanh.

$$\text{ReLU}(x) = \max(0, x) \qquad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

RNNs rely on the idea of sharing weights $\Theta = \{\boldsymbol{W}, \boldsymbol{C}, b\}$ across different parts of the model. Weight sharing allows the model to generalize across the data and not be dependent on the time index. Separate parameters at each time index would prohibit different sequence lengths at training and run time and sharing statistical strength across different sequence lengths and positions in time.

2.2 Unfolding the Computational Graph

In this section I will briefly introduce the idea of unfolding the recurrent computation of an RNN defined in 2.1 into a computational graph. Firstly, consider the simple dynamical system defined by

$$z_t = F_\Theta(z_{t-1}) \tag{2.2}$$

where z_t is called the (hidden) state of the system at time t. We can unfold Eq. 2.2 by applying the definition multiple time

$$\begin{aligned} z_t &= F_\Theta(z_{t-1}) && (2.3)\\ &= F_\Theta(F_\Theta(z_{t-1})) && (2.4)\\ &\dots \\ &= F_\Theta(\cdots(F_\Theta(z_0))) && (2.5) \end{aligned}$$

Unfolding the definition by iteratively applying the definition has led to an expression which doesn't contain any recurrence and can be represented by an acyclic computational graph. This computational graph is illustrated in Figure 2.1. The graph representation makes the idea of parameter sharing clear. Rather than needing to learn a separate model G^T for all possible time steps we have a single model F_Θ that operates on all time steps. This shared model also allows generalization to sequence lengths not seen in training.

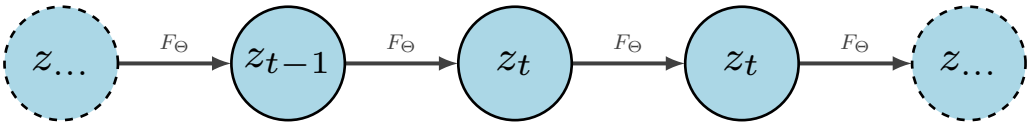


FIGURE 2.1: **Basic computational graph**: The classical discrete dynamical system illustrated as an unfolded computational graph. Each node represents the state of the system at time t. The function F_Θ maps the state at time t to the state at time $t+1$. The parameters Θ stay constant through time.

For a complete model we need to add external inputs to our graph representation and map the internal (hidden) states $z_{1\dots t}$ to outputs $\hat{x}_{1\dots t}$. These outputs can then be used to compute the training loss $\mathcal{L}$ when compared to the target values $x_{1\dots t}$.

The mapping from the latent space $\mathbb{R}^M$ to observation space $\mathbb{R}^N$ is called the observation equation or observation model.

$$\hat{x}_t = g(z_t) \tag{2.6}$$

Note that $g : \mathbb{R}^M \to \mathbb{R}^N$ is not necessarily linear.

The loss function commonly used in this context is the mean squared error (MSE), which calculates the average squared difference between the desired outputs $X = [x_1, \cdots, x_T]$ and the model outputs $\hat{X} = [\hat{x_1}, \cdots, \hat{x_T}]$, with the possibility of incorporating an additional regularizing term.

$$\mathcal{L} = \mathcal{L}_{MSE}(X, \hat{X}) + \mathcal{L}_{reg}(X, \hat{X}) = \frac{1}{T}\sum_{t=1}^{T}(x_i - \hat{x}_i)^2 + \mathcal{L}_{reg}(X, \hat{X}) \tag{2.7}$$

The full graph representation is shown in Figure 2.2. The RNN defined in Eq. 2.1 and in Figure 2.2 is universal in the sense that it is Turing-complete [5]. More important for the applications in this work, they are proven to be dynamically universal, i.e. they are able to approximate any open dynamical system to an arbitrary accuracy [6]. This approximation theorem for RNNs ensures the approximation capability of RNNs, it does not give a way to find the model parameters for good approximations. Finding globally optimal parameters for neural networks is in general a NP-complete problem [7].

2.3 Backpropagation through time

Using the computational graph representation established in the previous section the computation of the gradient through an RNN is straightforward. One simply applies the general backpropagation algorithm known from feedforward neural networks to the

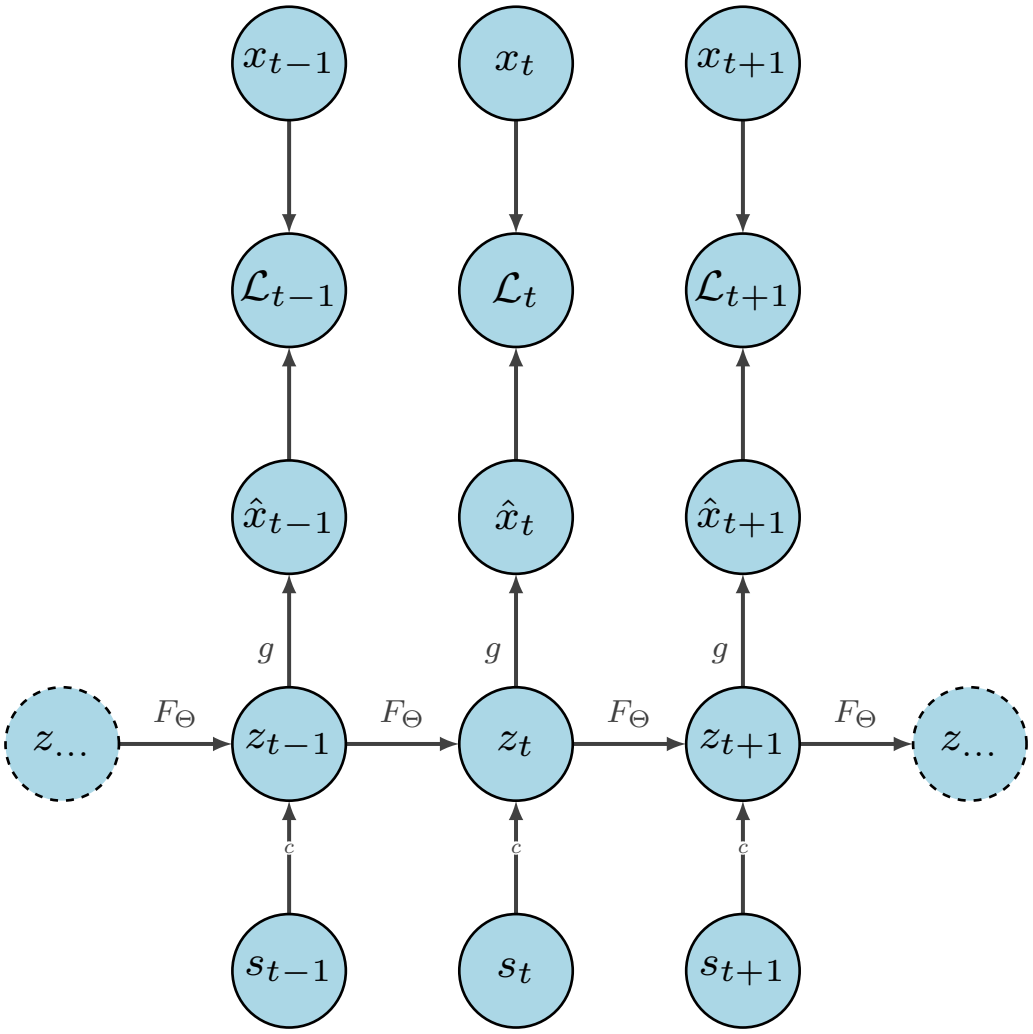


FIGURE 2.2: **Full graph of a recurrent neural network:** The RNN evolves forward in time from state z_{t-1} to z_t with F_Θ. Additional external inputs s_t are added to the state z_t via the function c. Observations $\hat{x}_t$ are produced from state z_t by the observation function g. The observations $\hat{x}_t$ and the ground truth x_t are used to compute the loss $\mathcal{L}_t$.

unrolled computational graph. This algorithm is called backpropagation through time (BPTT) and it can be used with any general-purpose gradient based methods to train the RNN. Given the loss function from Eq. 2.7 we can use BPTT to compute the gradients with respect to each parameter $\theta_i \in \Theta$ by recursively applying the chain rule backwards in time

$$(\nabla\mathcal{L})_i = \frac{\partial\mathcal{L}}{\partial\theta_i} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial\mathcal{L}_t}{\partial\theta_i} = \frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{t}\frac{\partial\mathcal{L}_t}{\partial\hat{x}_t}\frac{\partial\hat{x}_t}{z_t}\frac{\partial z_t}{\partial z_k}\frac{\partial z_k}{\partial\theta_i} \tag{2.8}$$

The term $\frac{\partial z_t}{\partial z_k}$ connects the latent state z_t with the past latent state z_{t-1}. We can expand this term using the chain rule

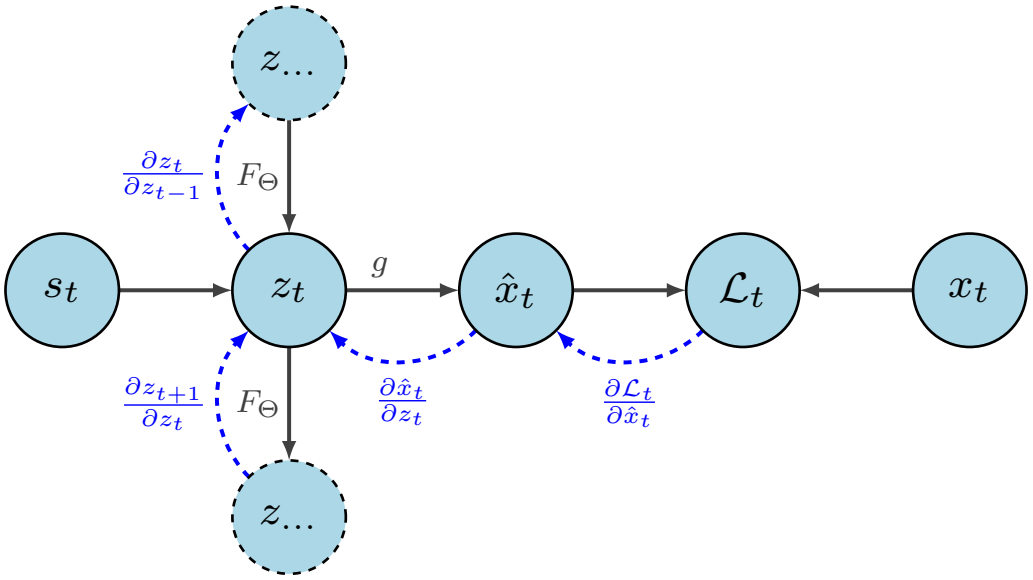


FIGURE 2.3: **Graph representation of BPTT:** The RNN evolves forward (*black*) in time with F_Θ. At time t observations $\hat{x}_t$ are produced from state z_t by the observation function g. The observations $\hat{x}_t$ and the ground truth x_t are compared in the loss $\mathcal{L}_t$. During the backwards pass (*blue*) the graph is traversed in reverse. The partial derivatives are computed along the way. The Jacobians $J_t = \frac{\partial z_t}{\partial z_{t-1}}$ express the temporal dependency of states.

$$\frac{\partial z_t}{\partial z_k} = \frac{\partial z_t}{\partial z_{t-1}} \frac{\partial z_{t-1}}{\partial z_k} = \dots = \prod_{\tau=k}^{t-1} \frac{\partial z_{\tau+1}}{\partial z_\tau} \tag{2.9}$$

This is a product of Jacobian matrices $J_t = \frac{\partial z_t}{\partial z_{t-1}}$ and is in each term of the inner sum in Eq. 2.8.

Computing the gradients of the loss function (Eq. 2.7) with respect to the parameters is computationally expensive. This computation involves a forward pass, moving from left to right in the unrolled graph shown in Figure 2.3, followed by a backward pass moving from right to left through the entire graph once again. Both the forward and backward passes are inherently sequential and cannot be parallelized since they depend on the previous hidden state. Consequently, they possess a computational complexity of $\mathcal{O}(T)$ for an input sequence length of T.
Additionally, due to the necessity of storing the states computed during the forward pass for use in the backward pass, the memory cost of BPTT is also $\mathcal{O}(T)$.

2.4 Exploding and vanishing gradients

Eq. 2.9 shows that gradients are propagated over long times if the sequence length T becomes large. This can lead to them either vanishing or exploding, causing numerical

issues during optimization. To illustrate the issue assume a simple RNN of the form

$$z_t = \boldsymbol{W} z_{t-1} \tag{2.10}$$

where the weight matrix W is diagonalizable with the decomposition $W = PDP^{-1}$, where D is a diagonal matrix with the eigenvalues λ_i of W and P is a nonsingular matrix consisting of the eigenvectors corresponding to the eigenvalues in D. The Jacobian is simply given by $J_t = W$ for all times t. Inserting this into Eq. 2.9 leads to the following

$$\frac{\partial z_t}{\partial z_k} = \prod_{\tau=k}^{t-1} J_{\tau+1} = \prod_{\tau=k}^{t-1} W = \prod_{\tau=k}^{t-1} PDP^{-1} = PD^{t-k}P^{-1} \tag{2.11}$$

We know from linear algebra that raising a diagonal matrix $D = \text{diag}(\lambda_i, \cdots, \lambda_M)$ to the power n can be done by raising the diagonal element to the power of n, so

$$D^n = \text{diag}(\lambda_i^n, \cdots, \lambda_M^n). \tag{2.12}$$

From Eq. 2.12 it is clear that the eigenvalues $|\lambda_i| < 0$ will decay towards 0 and the eigenvalues $|\lambda_i| > 0$ will diverge for large times. This phenomenon is called the exploding and vanishing gradient (EVG) problem and also occurs in more general RNNs.

To give a general criterion for EVG, first define the spectral radius of a matrix $M \in \mathbb{R}^{M\times M}$ with eigenvalues $\lambda_1, \cdots, \lambda_M$ as

$$\rho(M) = \max\left(|\lambda_1|, \cdots, |\lambda_M|\right). \tag{2.13}$$

Then the EVG criterion is given by

$$\left(\prod_{t=2}^{T} \rho(J_t)\right)^{\frac{1}{T-1}} = \begin{cases} > 1 \rightarrow \text{exploding gradients} \\ < 1 \rightarrow \text{vanishing gradients} \end{cases} \tag{2.14}$$

The general case is not as easy to compute as in the simple case of 2.10 as the Jacobians and their spectral radii change along the trajectory $z_{1:T}$.

It is important to note that the EVG problem cannot be avoided by simply constraining the parameter space to regions without EVGs. As shown in [8], whenever a model

is able to represent long-term dependencies, the gradient of the long-term interaction has exponentially smaller magnitude than the gradient of short-term interactions. This means that the gradient signal can easily be hidden by small fluctuations arising from short-term dependencies. In practice, experiments such as in [9] have shown that when using stochastic gradient descent to train vanilla RNNs, the task of learning dependencies in sequences of only lengths 10 to 20 becomes progressively more challenging.

2.5 Dealing with the exploding and vanishing gradient

Since the EVG problem was discovered in the 90s, different attempts to solve the problem have been proposed. In the following I will only give a brief overview, a more complete review of different methods can be found in [10].

2.5.1 L_1 and L_2 Regularization

Assuming the recurrent weights are initialized such that $\rho(J) < 1$, the L_1/L_2 regularizing terms can ensure that during training the eigenvalues stay small, and thus the gradients can't explode. Regularization can however limit the model by preventing it to exhibit long term memory traces or learn more complex attractor geometries in state space.

2.5.2 Alternative architectures

Alternative network architectures such as the long short-term memory (LSTM) net by [11] aim to eliminate the vanishing gradient problem by design. They rely on a special type of unit with a self connection. The flow of information through a cell is regulated by learned input, output and forget gates. The design allows for gradients to flow through the network unchanged and not vanish. This solution does not however prevent exploding gradients and networks with these units are often harder to train as they have more dynamical variables and a higher number of parameters.

2.5.3 Gradient Clipping

Gradient clipping (GC) or rescaling is a common method to deal with exploding gradients by simply rescaling them if they go past a certain threshold to prevent a floating-point number overflow.

Gradient clipping is simple and computationally efficient, but introduces the threshold value τ as another hyperparameter. In [10] it suggested setting the threshold as the

Algorithm 1 Pseudocode for gradient clipping

$\hat{g} \leftarrow \frac{\partial \mathcal{L}}{\partial \Theta}$
if $|\hat{g}| \geq \tau$ **then**
  $\hat{g} \leftarrow \frac{\text{threshold}}{\hat{g}}$
end if

average norm of gradients $\langle|\frac{\partial \mathcal{L}}{\partial \Theta}|\rangle$ across large number of updates multiplied by $\gamma \in [0.5, 10]$ such that $\tau = \gamma\langle|\frac{\partial \mathcal{L}}{\partial \Theta}|\rangle$.

2.5.4 Smart Initialization

A further technique to deal with the EVG problem is to initialize the recurrent weights in such a way that they are easy to train and are good at modeling long-range dependencies. This is the so-called initialization trick or smart initialization.

In [12] it is proposed to use the ReLU activation function and initializing the hidden-to-hidden weight matrix $\boldsymbol{W}$ to the identity. This identity initialization has the property that error derivatives remain constant when backpropagated as long as no extra error-derivatives are added. This is similar to the behavior of the LSTMs forget gates which let error gradients pass with no decay.

In [13] it is proposed to initialize the hidden-to-hidden weight matrix $\boldsymbol{W}$ such that the maximum eigenvalue is 1 and all others are smaller 1. This results in a line attractor configuration at initialization, meaning that the trajectory of the hidden states will evolve towards the principal axis with eigenvalue 1, thereby eliminating the conditions for exploding gradients.

2.6 Teacher Forcing

Teacher forcing (TF) is a technique commonly used in training RNNs in a supervised learning setting. The name teacher forcing was coined by [14], who in turn cite [15] and [16] for first introducing the method. The idea is to replace or *force* the hidden RNN state z_i by a desired target state d_i during training to guide the network towards the desired computational task. The desired target state is called teacher or forcing signal.

Next, let us define TF more formally. Let F_Θ be an RNN, $z_t \in \mathbb{R}^M$ the networks latent state and $d_t \in \mathbb{R}^M$ the desired target value. Define I as the set of indices at which the network is forced. The state equation is given by

$$z_{t+1} = \begin{cases} F_\theta(d_t) \text{ if } t \in I \\ F_\theta(z_t) \text{ else} \end{cases} \tag{2.15}$$

This form of teacher forcing forces the orbits of the RNN back on the ground truth orbits. It is important to note that in [14], the observation equation is the identity function, i.e. $\hat{x}_t = z_t$, thus the forcing signal is simply set $d_t = x_t$ for all time steps $t \in I$.

In order to retrieve useful error gradients, the loss function at forcing time t, $\mathcal{L}_t$, is computed prior to forcing, so on the generated observation $\hat{x}_t$, not on the output of the forced network. At each forcing time step $t \in I$ the gradients are truncated. To see this, evaluate the Jacobian J_{t+1}

$$J_{t+1} = \frac{\partial z_{z+1}}{\partial z_t} = \frac{\partial F_{\Theta(d_t)}}{\partial d_t}\frac{\partial d_t}{\partial z_t} = \frac{\partial F_{\Theta(d_t)}}{\partial d_t} \cdot 0 = 0. \tag{2.16}$$

The term $\frac{\partial d_t}{\partial z_t} = 0$ because the forcing signal d_t does not depend on the hidden state z_t. This means that TF circumvents the EVG problem but also has a limited capability to learn long term dependencies because the gradients are cut off at the forcing time steps $t \in I$.

In *sparse* teacher forcing, the forcing signal is only applied sparsely at time intervals τ. This of course introduces another hyperparameter into training. In [17] it is argued that the forcing interval should be chosen in accordance to the maximal Lyapunov exponent λ_{max} of the system, which gives us a predictability time $\tau = \frac{\ln 2}{\lambda_{max}}$. Given a time interval τ chosen for a time series of length T we can then define the set of forcing indices

$$I = \{t | t = k\tau < T, k \in \mathbb{N}\} \tag{2.17}$$

In this sparse approach the model is allowed to diverge from the true dynamics in the short term to learn the observed dynamics but is regularly forced back to the true behavior, cutting the gradients and avoiding EVG. But this is only true in the simplest case that the latent and observation space are of equal dimension $N = M$. In the more common case that $z_t \in \mathbb{R}^M$ and $x_t \in \mathbb{R}^N$ with $N < M$ forcing can be applied to the N "visible units" of the state while the $M - N$ other hidden units are unchanged. The risk of EVG remains as gradients can still flow through these hidden units.

In [18] a different form of teacher forcing, the so-called *weak* teacher forcing is introduced. Instead of replacing the entire hidden state at certain time steps as in 2.15, the is

generated state variable z_t and the desired target d_t are linearly interpolated using a weighting coefficient $\alpha \in [0, 1]$.

$$z_{t+1} = F_\Theta((1-\alpha)z_z + \alpha d_t) \tag{2.18}$$

In [18] it is hypothesized that the solutions found by this weak forcing regime are more likely to be stable than those found in the fully forced case. We can once again compute the Jacobian in the weak TF case

$$J_{t+1} = \frac{\partial z_{z+1}}{\partial z_t} = \frac{\partial F_\Theta(\tilde{z}_t)}{\partial \tilde{z}_t}\frac{\partial \tilde{z}_t}{\partial z_t} = (1-\alpha)F'_\Theta(\tilde{z}_t) \tag{2.19}$$

with $\tilde{z}_t = (1-\alpha)z_t + \alpha d_t$. It follows that the chain of Jacobians remains intact, hence EVG may still occur in weak TF and the hyperparameter α is introduced.

2.7 Piecewise Linear Recurrent Neural Network (PLRNN)

In this thesis I will use the so called piecewise linear recurrent neural network (PLRNN) architecture first introduced in [19]. The state equation of the PLRNN is

$$z_t = \boldsymbol{A}z_{t-1} + \boldsymbol{W}\Phi(z_{t-1}) + \boldsymbol{C}s_{t-1} + h. \tag{2.20}$$

where $z_t \in \mathbb{R}^M$ is latent state vector at time t, Φ is the ReLU activation function, $\boldsymbol{W}$ is an off-diagonal matrix of connection weights and $\boldsymbol{A}$ a diagonal matrix holding the autoregressive weights.

This model formulation is motivated by neuroscience. From this point of view, the entries of the latent state z_{i_t} can be interpreted as membrane potential, the diagonal elements in $\boldsymbol{A}$ are seen as the neurons individual membrane time constants and the off-diagonal elements in $\boldsymbol{W}$ represent synaptic connections between neurons. The ReLU activation models the fact that neurons only show spiking activities above a certain firing threshold. From a mathematical point of view, Eq. 2.20 has the form of an autoregressive model with a nonlinear basis expansion in the latent variables z_{i_t}.

A particular advantage of the PLRNN model is that many of its dynamical properties can be computed analytically while simultaneously enjoying the same universal approximation capabilities of the vanilla RNN. For example, setting external inputs to 0, fixed points z_* can be obtained analytically by solving 2^M linear equations.

$$z_* = (A + W - \mathbb{1})^{-1} \tag{2.21}$$

Besides fixed points, $k-cycles$ (including their stability) ([20], [21], [22]) and Jacobians of the system can be computed, which in turn also allows for computation of the Lyapunov spectrum ([23]).

2.8 PLRNN extensions

[22] extended the basic PLRNN structure by adding a linear spline basis expansion, called the dendritic PLRNN (dendPLRNN), to increase the expressivity of each unit and improve performance, especially for low dimensional systems. The state equation is given by

$$z_t = \boldsymbol{A} z_{t-1} + \boldsymbol{W} \sum_{b=1}^{B} \alpha_b \phi(z_{t-1} - h_b) + h_0 \tag{2.22}$$

with slope-threshold pairs $\{\alpha_b, h_b\}_{b=1}^{B}$, where B is the number of bases. It can be shown that the dendPLRNN can be reformulated as a higher dimensional conventional PLRNN, so the properties of PLRNN are preserved.

However, in this thesis I will use the shallow PLRNN (shPLRNN), a "one hidden layer" PLRNN architecture. The state equation is

$$z_t = \boldsymbol{A} z_{t-1} + \boldsymbol{W_1} \phi \left(\boldsymbol{W}_2 z_{t-1} + h_2 \right) + h_1 \tag{2.23}$$

with latent states $z_t \in \mathbb{R}^M$ and the diagonal matrix $\boldsymbol{A} \in \mathbb{R}^{M \times M}$ as in Eq. 2.20. $\boldsymbol{W}_1 \in \mathbb{R}^{M \times L}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{L \times M}$ are rectangular connectivity matrices, $h_2 \in \mathbb{R}^L$ and $h_1 \in \mathbb{R}^M$ are thresholds. Dimensions $L > M$ are used.

By expanding each unit's activation into a weighted sum of ReLU nonlinearities, this formulation appears similar to the dendPLRNN. [24] show that it is indeed possible to rewrite the shPLRNN into a dendPLRNN. Again this implies that the desired mathematical properties of the PLRNN model are preserved.

Both the dendPLRNN and the shPLRNN can be equipped with a clipping mechanism that prevents states from diverging to ∞ due to unbounded ReLU nonlinearities. For shPLRNN this clipping is given by

$$z_t = \boldsymbol{A} z_{t-1} + \boldsymbol{W_1} \left[\phi \left(\boldsymbol{W}_2 z_{t-1} + h_2 \right) - \phi \left(\boldsymbol{W}_2 z_{t-1} \right) \right] + h_1 \tag{2.24}$$

This mechanism guarantees bounded orbits provided the eigenvalues of $\boldsymbol{A}$ are smaller than 1.

2.9 Linear Observation model

To map the latent states to observation space the PLRNN is typically equipped with an affine observation equation

$$\hat{x}_t = \boldsymbol{B} z_t + b \tag{2.25}$$

with weights $\boldsymbol{B} \in \mathbb{R}^{N \times M}$ and a bias $b \in \mathbb{R}^N$. The bias term is typically dropped when dealing with preprocessed time series data which has been standardized, as is the case in this work. Hence, equation 2.25 simplifies to a linear observation equation.

2.10 Identity Teacher Forcing

The simplest case of the linear observation equation is an identity mapping between a subspace of the latent states $z_t \in \mathbb{R}^M$ and the outputs of the model $\hat{x}_t \in \mathbb{R}^N$. This achieved by setting B to a fixed matrix $\mathcal{I} \in \mathbb{R}^{N \times M}$, which maps the first N units of z_t directly on to x_t. $\mathcal{I}$ can be defined as

$$\mathcal{I}_{ij} = \begin{cases} 1, & \text{if } i = j \text{ and } i, j \leq N \\ 0, & \text{else} \end{cases} \tag{2.26}$$

This implies splitting z_t into hidden and observation or readout units

$$z_t = \begin{bmatrix} z_t^{obs} = \hat{x}_t \\ z_t^{hid} \end{bmatrix} = \begin{bmatrix} z_t^1 = \hat{x}_t^1 \\ \vdots \\ z_t^N = \hat{x}_t^N \\ \vdots \\ z_t^M \end{bmatrix} \tag{2.27}$$

such that

$$\hat{x}_t = \mathcal{I} z_t. \tag{2.28}$$

This allows for *direct teacher forcing* to be applied by simply supplying the teacher signal directly to the observation neurons z_t^{obs}. The forcing signal does not change the hidden units z_t^{hid}. This gives us

$$d_t = \begin{bmatrix} x_t \\ z_t^{hid} \end{bmatrix} \tag{2.29}$$

as the desired target value d_t which is applied in different TF variants (see Eq. 2.15 and Eq. 2.18). Note that this makes training susceptible to the EVG problem regardless or forcing protocol because gradients can flow freely without truncation through the space of hidden neurons z^{hid}.

2.11 Inversion Teacher Forcing

To allow for more expressivity in the observation model, we do not want to constrain the $B \in \mathbb{R}^{N \times M}$ matrix. In this case we can no longer map the observations back to the latent state to obtain the target values d_t we need as forcing signals. In [23] a TF method is introduced for this case which I will refer to as *Inversion Teacher Forcing*. The inversion of the linear out mapping is approximated using the psuedo-inverse B^+ (also known as the Moore-Penrose inverse) of B. Hence, the desired target values d_t are computed as follows:

$$d_t = B^+ x_t = (B^T B)^{-1} B^T x_t \tag{2.30}$$

In this case the full latent state vector z_t is replaced by the target values d_t, which means gradient are truncated in the case of sparse TF. In the paper [23] itself, sparse TF is used and the forcing interval is estimated using the maximal Lyapunov exponent λ_{max} of the system.

$$\tau = \frac{\ln 2}{\lambda_{max}} \tag{2.31}$$

Now that I have introduced the PLRNN and its implementation with the linear observation model, it's time to adapt this existing framework to model fMRI time series and define a new observation model.

2.12 BOLD observation model

The aim of this thesis is to expand the existing PLRNN training framework to fMRI data. For this I will next briefly introduce the physical basics behind fMRI recordings and how to the model the "BOLD" response.

2.12.1 Physical principals of fMRI

The term *functional magnetic resonance imaging* (fMRI) generally refers to the imaging of brain activation detectable by changes in regional cerebral blood flow.

To support the cells of the brain with oxygen (O_2), it is transported within the red blood cells bound in hemoglobin molecules. When hemoglobin molecules bind to O_2, they form oxyhemoglobin (OHb), which has no unpaired electrons and is weakly diamagnetic. When oxygen is released, deoxyhemoglobin (DHb) is left with 4 unpaired electrons exposed at each iron center, causing it to become strongly paramagnetic. The presence of paramagnetic deoxyhemoglobin within the blood cells creates local magnetic field distortions. These distortions cause regional $T2$ and $T2^*$ relaxations times to decrease and suppress the MR signal. These Signal intensity variations in the MR image are called the *blood oxygen level dependent* (BOLD) contrast. Functional MRI is a temporally resolved sequence of $T2^*$ weighted MR images.

During cerebral activation, regional cerebral blood flow (CBF) increases, but the cerebral metabolic rate of oxygen consumption (CMRO_2) is not proportionally elevated. CBF and CMRO_2 are thus said to be "uncoupled" ([25]). Instead, more freshly oxygenated blood is supplied to active regions of the brain than is required for its immediate metabolic needs. This is means that the relative concentration of DHb in activated areas will decrease, local field inhomogenities diminish and the BOLD signal in activated areas increases.

2.12.2 Modelling the Hemodynamic Response

The BOLD signal has been shown to have a consistent response to a short stimulus, peaking after around 6 seconds and then falling back to baseline over the next several seconds. For the purposes of estimating the BOLD signal in an experimental paradigm, the SPM framework ([26]) makes use of a canonical hemodynamic response function (HRF). This function is assumed to be the response of the system (as reflected by the MR signal) to a brief, intense period of neural stimulation. The canonical HRF in SPM is defined as the difference of two Gamma function, see Eq. 2.32. The implementation,

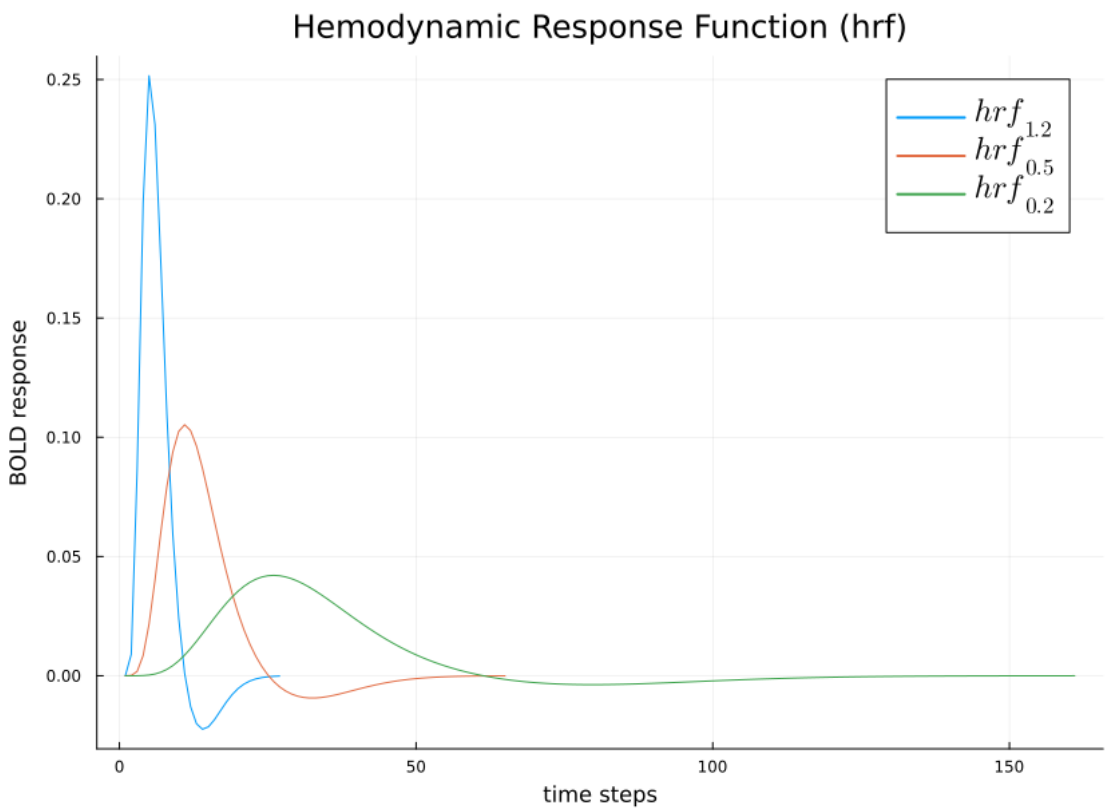


FIGURE 2.4: **Comparison of hrf functions with varying repetition times**

including the shape parameters, was ported from the SPM MATLAB implementation into julia for this thesis.

$$h(t) = A\left(\frac{t^{\alpha_1-1}\beta^{\alpha_1}e^{-\beta_1 t}}{\Gamma(\alpha_1)} - c\frac{t^{\alpha_2-1}\beta^{\alpha_2}e^{-\beta_2 t}}{\Gamma(\alpha_2)}\right) \tag{2.32}$$

The shape parameters are already estimated by the SPM implementation, only the repetition or repeat time (RT) parameter is passed to the hrf function. It is the temporal resolution at which the fMRI time series was acquired. In the following I will refer to an hrf function with a given RT as hrf_{RT} for clarity. hrf functions with varying repetition times are shown in figure 2.4.

The observed BOLD signal $x[t]$ is assumed to be driven by underlying neural event signal $s[t]$. A convolution with the hrf function and added measurement noise $\epsilon[t]$ is used to model the relationship between the BOLD response and the underlying neural event. The BOLD signal in a given voxel (or region of interest) at time t, $x[t]$, is thus given by the following equation:

$$x[t] = \sum_{\tau=1}^{M} h[\tau] s(t-\tau) + \epsilon[t] \qquad \epsilon[t] \sim \mathcal{N}(0, \Sigma) \tag{2.33}$$

where $\epsilon[t] \sim \mathcal{N}(0, \Sigma)$ is a Gaussian white noise term with diagonal covariance matrix $\Sigma = diag(\sigma_{11}^2, \cdots, \sigma_{NN}^2)$. The convolution operation will be covered in more depth in section 3.4.

2.12.3 Defining the BOLD observation equation

An appealing feature of the state space model framework is that different measurement modalities can be accommodated by connecting different observation models to the same latent model. To model fMRI time series we therefore simply need to adapt the observation equation 2.25 to the BOLD response model in equation 2.32. The latent states z_t of the PLRNN with its universal approximation capabilities should then learn to model the (unknown/unobservable) neural activity when trained on fMRI time series.

The modified observation equation is as follows

$$\hat{x}_t = \boldsymbol{B}\left((hrf * z)_t\right) + \boldsymbol{J} r_t + \epsilon_t \tag{2.34}$$

where $\hat{x}_t \in \mathbb{R}^N$ is the estimated BOLD signal in N voxels at time t generated from the latent states $z \in \mathbb{R}^M$ convolved with the hrf function. The convoluted latent states are observed through a linear-Gaussian model and additional nuisance predictors $r_t \in \mathbb{R}^n$, which account for movement artifacts, are added. $\boldsymbol{J} \in \mathbb{R}^{N \times P}$ is the coefficient matrix of these nuisance variables.

This observation equation however introduces a major complication. The observations in Eq. 2.34 do not just depend on the current state z_t as in the simple linear model, but on a set of states $z_{\tau:t}$ across several previous time steps due to the convolution operation. It is now no longer clear how to calculate the forcing signal because the model outputs $\hat{x}_t$ depend on multiple latent states, therefore the target value d_t for the latent state z_t cannot simply be calculated from one data point x_t anymore.

To solve this problem I will introduce the mathematical frameworks of the Fourier and the Wavelet Transformation in the next two chapters. These will give us the mathematical tools needed to handle convolution and *deconvolution*, the inverse operation to convolution.

Chapter 3

Fourier Transform

The Fourier transform is a mathematical operation that decomposes a function or signal into its constituent frequencies. In this chapter I will give a short overview of the Fourier transform (FT) and properties important to this work. A precise mathematical approach including the proofs of the properties given here can be found in [27], more applied information focused on discrete FT and signal processing can be found in [28] and [29].

3.1 Continuous Fourier Transform

Definition 3.1 (Continuous Fourier Transform)**.** For a function $f \in L^1(\mathbb{R}^d)$ the Fourier transform $\mathscr{F}$ is defined as

$$\mathscr{F}\{f(x)\}(k) = \widehat{f}(k) = \int_{\mathbb{R}^d} f(x) e^{-ik^T x} d^d x \tag{3.1}$$

the output of the transform is a complex-valued function of frequency. The term Fourier transform refers to both this complex-valued function $\widehat{f}$ and the mathematical operation $\mathscr{F}$.

In the following I will list some important properties of the Fourier transform. First I will define two operators that will clear up notation.

Definition 3.2. The translation operator τ_y is defined for $y \in \mathbb{R}^d$ as $\tau_y f = f(y + \cdot)$. The dilatation operator σ_A is defined for $A \in \mathbb{R}^{d \times d}$ as $\sigma_A f = f(A\cdot)$.

Proposition 3.3 (Properties of the Fourier transform)**.** *For $f, g \in L^1(\mathbb{R}^d)$ the following properties hold*

1. *Linearity: For $a, b \in \mathbb{C}$*

$$\mathscr{F}\{a \cdot f(x) + b \cdot g(x)\} = a \cdot \mathscr{F}\{f(x)\} + b \cdot \mathscr{F}\{g(x)\}$$

2. *Translation: For $y \in \mathbb{R}^d$*

$$\widehat{(\tau_y f)}(k) = e^{iy^T k}\widehat{f}(k)$$

3. *Modulation: For $a \in \mathbb{R}$*

$$\mathscr{F}\{e^{-iax} f(x)\} = \widehat{\tau_a f}$$

4. *Time scaling: For $A \in \mathbb{R}^{d\times d}$ an invertable matrix*

$$\widehat{\sigma_A f}(k) = \frac{\widehat{f}(A^{-T}k)}{\|\det A\|}$$

   Note that in one dimension $d = 1$ this formula simplifies to $\widehat{\sigma_a f}(k) = \frac{1}{|a|}\widehat{f}(\frac{k}{a})$ with $a \in \mathbb{R}$.

5. *Inverse Fourier Transform:*

$$f(x) = \mathscr{F}^{-1}\{\widehat{f}(k)\}(x) := \frac{1}{(2\pi)^d}\int_{\mathbb{R}^d} \widehat{f}(k)e^{ix^T k}dk$$

3.2 Discrete Fourier Transform

The discrete Fourier transform (DFT) is the discrete version of the Fourier transform. It transforms a signal represented as a discrete sequence into its equivalent representation in the frequency domain.

Definition 3.4 (Discrete Fourier Transform)**.** Given a signal $\{x[n]\}$ of length $N \in \mathbb{N}$ with $x[n] \in \mathbb{C} \quad \forall n \in [\![0, N]\!]$. The discrete Fourier Transform $\mathscr{F}$ is defined as

$$\mathscr{F}\{x[n]\}[k] = \widehat{x}[k] = \sum_{n=0}^{N-1} x[n]e^{-\frac{2\pi i}{N}kn} \tag{3.2}$$

Linearity, modulation and translation properties also hold for the DFT, given the arguments are integer valued and remain in the definition space. The inverse discrete Fourier transform(IDFT) also exists and is defined as follows:

Definition 3.5 (Inverse Discrete Fourier Transform)**.** Given a complex valued signal $\{\widehat{x}[k]\}$ of length $N \in \mathbb{N}$ with $\widehat{x}[k] \in \mathbb{C} \quad \forall k \in [\![0, N]\!]$. The inverse discrete Fourier Transform $\mathscr{F}^{-1}$ is defined as

$$\mathscr{F}^{-1}\{\widehat{x}[k]\}[n] = x[n] = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{x}[k] e^{\frac{2\pi i}{N}kn} \tag{3.3}$$

It holds that $x[n] = \mathscr{F}^{-1}\{\mathscr{F}\{x[n]\}\}$ and $\widehat{x}[k] = \mathscr{F}^{-1}\{\mathscr{F}\{\widehat{x}[k]\}\}$

3.2.1 Unitary DFT

Another way of representing the DFT is as a matrix transform. If we represent out signal as vector $X = \Big[x[1], \cdots, x[N]\Big]^T$, the DFT can be expressed as the DFT matrix

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix} \tag{3.4}$$

with $\omega = e^{-\frac{2\pi i}{N}}$. The inverse transform has the same matrix structure but with opposite-sign exponents. Please also note that in this matrix representation the normalization factor $\frac{1}{\sqrt{N}}$ is present for both forward and transform. The placing of the normalization factor and the sign are just convention and differ across the literature.

This choice however makes the DFT matrix W and its inverse $W^{-1} = W^*$ unitary. This allows us to interpret the DFT simply as basis transformation of the signal into the Fourier basis. It also follows immediately that the DFT conserves the energy of the signal. This is formalized in the following theorem.

Theorem 3.6 (Plancherel theorem)**.** *Given two complex valued signals $\{x[n]\}$, $\{y[n]\}$ of length $N \in \mathbb{N}$ and their Fourier transforms $\{\widehat{x}[n]\}$, $\{\widehat{y}[n]\}$, the following equality holds:*

$$\sum_{n=0}^{N-1} x[n]y^*[n] = \sum_{k=0}^{N-1} \widehat{x}[k]\widehat{y}^*[k] \tag{3.5}$$

In the special case of $\{x[n]\} = \{y[n]\}$ *for all* n*, we get* $\sum_{n=0}^{N-1} |x[n]|^2 = \sum_{k=0}^{N-1} |\widehat{x}[k]|^2$*. This result can also be extended to the continuous case. For* $f, g \in L^1(\mathbb{R}^d) \cap L^2(\mathbb{R}^d)$ *it holds that*

$$\int_{\mathbb{R}^d} f(x)g^*(x)d^dx = \int_{\mathbb{R}^d} \widehat{f}(k)\widehat{g}^*(k)d^dk \tag{3.6}$$

again in the case that $f = g$ *this simplifies to energy conservation* $\int_{\mathbb{R}^d} |f(x)|^2 d^dx = \int_{\mathbb{R}^d} |\widehat{f}(k)|^2 d^dk$

3.3 Fast Fourier Transform

The naive computation of the DFT 3.4 is very computationally expensive. It requires N^2 complex multiplications and $N^2 - N$ complex additions, giving the algorithm a complexity of $\mathcal{O}(N^2)$. By exploiting the symmetry ($e^{\frac{2\pi i}{N}(k+N/2)} = -e^{\frac{2\pi i}{N}k}$) and the periodicity ($e^{\frac{2\pi i}{N}(k+N)} = e^{\frac{2\pi i}{N}k}$) of the phase factor a more efficient algorithm can be derived.

3.3.1 Cooley–Tukey FFT algorithm

The Cooley-Turkey FFT algorithm is the most common FFT algorithm. It reexpresses the DFT of a signal of composite length $N = N_1 N_2$ as N_1 DFTs of size N_2. This is applied recursively to reduce the computational complexity to $\mathcal{O}(N \log(N))$ for highly composite numbers. Subsequently, I will outline the fundamental concept behind the algorithm.

Assume we have a signal $\{x[n]\}$ of length N where N is not prime and can be written as the product of two integers $N = LM$. Note that this assumption is not restrictive since the signal can be padded with zeros to ensure the condition is met.

First the sequence is stored as a two-dimensional array indexed by the row index $l \in [\![0, L-1]\!]$ and the column index $m \in [\![0, M-1]\!]$. Choose the mapping $n = Ml + m$ to map the index n to the indices (l, m). The computed DFT values can be mapped in a similar fashion by mapping the index k to the pair (p, q) with $k = Mp + q$.

Now suppose the signal is mapped as $x[l, m]$ and the results as $\widehat{x}[p, q]$. Then the DFT equation 3.4 can be written as double sum of the array elements

$$\widehat{x}[p, q] = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x[l, m] e^{\frac{2\pi i}{N}(Mp+q)(mL+l)} \tag{3.7}$$

Now we can use the following identities of the phase factor: $e^{\frac{2\pi i}{N}mqL} = e^{\frac{2\pi i}{N/L}mq} = e^{\frac{2\pi i}{M}mq}$, $e^{\frac{2\pi i}{N}Mpl} = e^{\frac{2\pi i}{N/M}pl} = e^{\frac{2\pi i}{L}pl}$, $e^{\frac{2\pi i}{N}Nmp} = 1$. With these identities, equation 3.7 can be written as

$$\widehat{x}[p,q] = \sum_{l=0}^{L-1} \left\{ e^{\frac{2\pi i}{N}lq} \left[\sum_{m=0}^{M-1} x[l,m] e^{\frac{2\pi i}{M}mq} \right] \right\} e^{\frac{2\pi i}{L}lp} \tag{3.8}$$

This expression now involves the computation of DFTs of length M and L. The number of complex multiplications have been reduced from N^2 to $N(M+L+1)$ and the number of complex additions from $N(N-1)$ to $N(M+L+2)$. When N is a highly composite number, i.e. N can be factored into a product of prime numbers

$$N = r_1 r_2 \dots r_\nu \tag{3.9}$$

then the decomposition can be repeated $\nu - 1$ more times, reducing the computational complexity each time.

In the special case of $r_1 = r_2 = \cdots = r_\nu = 2$, so that $N = 2^\nu$ the computation of the N-point DFT has a regular pattern. The number r is called the *radix* of the FFT algorithm. For $r = 2$ the resulting FFT algorithm is termed the radix-2 FFT algorithm.

Consider the case of signal length $N = 2^\nu$. Select $M = N/2$ and $L = 2$. This selection results in splitting the signal into two subsignals $f_1[n]$ and $f_2[n]$ corresponding to the even even-numbered and odd-numbered samples of $\{x[n]\}$.

$$\begin{aligned} f_1[n] &= x[2n] && n \in [\![0, N/2-1]\!] \\ f_2[n] &= x[2n+1] \end{aligned}$$

Now the DFT can be expressed in terms of DFTs of the decimated sequences as follows:

$$\widehat{x}[k] = \sum_{m=0}^{N/2-1} x[2m] e^{\frac{2\pi i}{N}2mk} + \sum_{m=0}^{N/2-1} x[2m+1] e^{\frac{2\pi i}{N}(2m+1)k} \tag{3.10}$$

using $e^{\frac{2\pi i}{N}2} = e^{\frac{2\pi i}{N/2}}$ we can rewrite this to

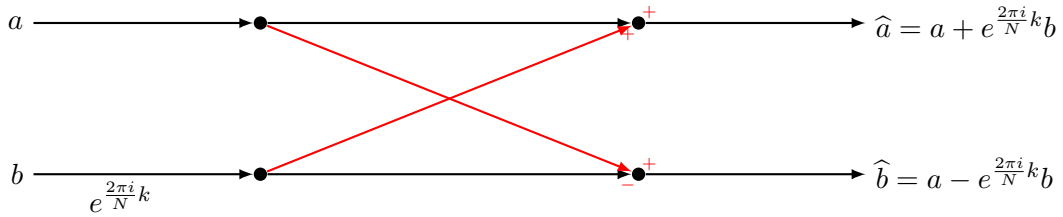


FIGURE 3.1: Basic butterfly diagram in the decimation-in-time FFT algorithm

$$\widehat{x}[k] = \sum_{m=0}^{N/2-1} f_1[m]e^{\frac{2\pi i}{N/2}mk} + \sum_{m=0}^{N/2-1} f_2[m]e^{\frac{2\pi i}{N/2}mk} \tag{3.11}$$

$$= \widehat{f}_1[k] + e^{\frac{2\pi i}{N}k}\widehat{f}_2[k] \qquad k \in [\![0, N-1]\!] \tag{3.12}$$

where $\widehat{f}_1[k]$ and $\widehat{f}_2[k]$ are the $N/2$ DFTs of $f_1[k]$ and $f_2[k]$. Using the fact that $\widehat{f}_1[k]$ and $\widehat{f}_2[k]$ are periodic with period $N/2$. In addition, $e^{\frac{2\pi i}{N}(k+N/2)} = -e^{\frac{2\pi i}{N}k}$. Hence, we can rewrite 3.12 as

$$\begin{aligned} \widehat{x}[k] &= \widehat{f}_1[k] + e^{\frac{2\pi i}{N}k}\widehat{f}_2[k] \\ \widehat{x}[k + \frac{N}{2}] &= \widehat{f}_1[k] - e^{\frac{2\pi i}{N}k}\widehat{f}_2[k] \end{aligned} \tag{3.13}$$

The basic computation in 3.13 is called the *butterfly* structure because the flow graph resembles a butterfly as shown in Figure 3.1. It is the basic computation performed in the radix-2 FFT. It can be seen that by this process the number of complex multiplications has been reduced from N^2 to $N^2/2 + N/2$. We can repeat this process, the so-called *decimation-in-time*, recursively for both $f_1[n]$ and $f_2[n]$. For $N = 2^\nu$, this decimation can be performed ν times reducing the number of multiplications to $\frac{N}{2}\log_2 N$ and the number of additions to $N \log_2 N$.

For illustration, Figure 3.2 depicts the computation of an $N = 8$-point DFT. In three stages, first 4 two-point DFTs are computed, then 2 four-point DFTS and lastly one 8-point DFT. These smaller DFTs are combined in a butterfly structure to form the larger DFT.

Implementing the FFT efficiently poses further challenges that I will not cover here. In this work I will use the FFTW library, implementation and design details can be found in [30].

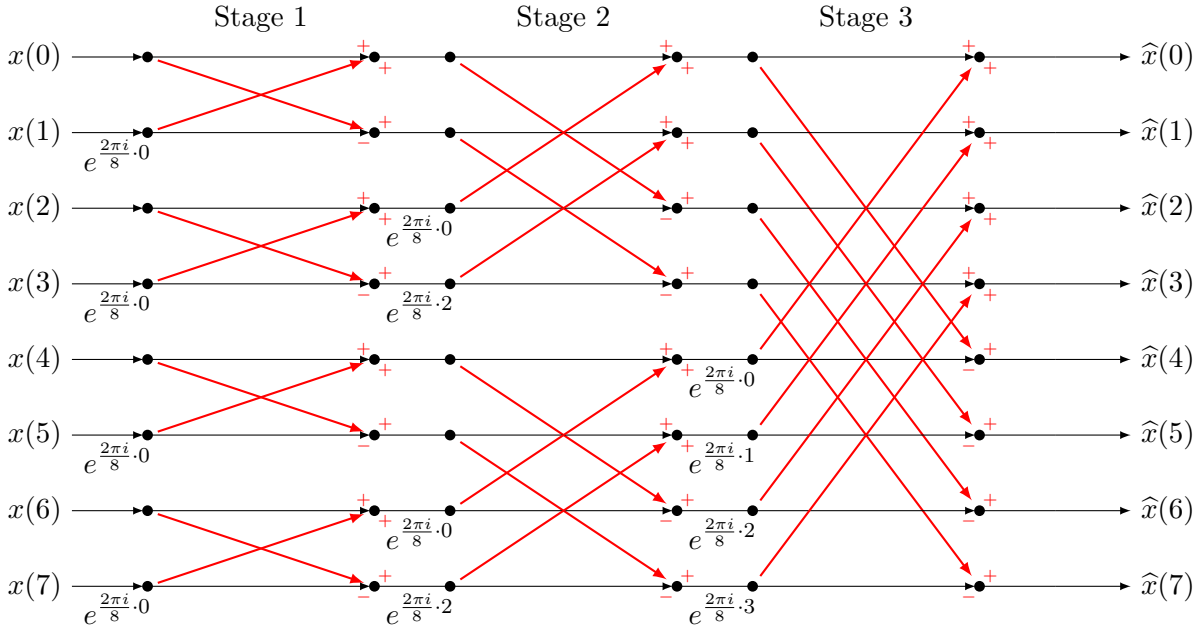


FIGURE 3.2: Visualization of a 3 stage radix-2 FFT algorithm

3.4 Convolution and the Convolutional Theorem

Having established the basic properties of the Fourier transform, I will now introduce the convolution operation and its important properties.

Definition 3.7 (Convolution)**.** Let $f, g \in L^1(\mathbb{R}^d)$ be integrable functions. The convolution of f and g, written as $(f * g)$, is defined as

$$(f * g)(t) := \int_{\mathbb{R}^d} f(\tau) g(t - \tau) d^d\tau \tag{3.14}$$

The convolution operation is also defined for discrete functions.

Definition 3.8 (Discrete Convolution)**.** Let f, g be complex-valued functions defined on $\mathbb{Z}$. The convolution of f and g is defined as

$$(f * g)[n] := \sum_{m=-\infty}^{\infty} f[m] g[n - m] \tag{3.15}$$

Note that in the case that g is non-zero only on a finite interval $[\![-M, M]\!]$ the series in the above definition can be replaced with a finite summation

$$(f * g)[n] = \sum_{m=-M}^{M} f[m] g[n - m]. \tag{3.16}$$

The definition of the discrete convolution in Eq. 3.16 can be rewritten as a matrix vector multiplication. Write the signal as a vector $x = [x[1], \cdots, x[N]]^T$. The convolution with the impulse response $h = [h[1], \cdots, h[M]]^T$ is given by the multiplication with the $(M + N - 1) \times N$ convolution matrix

$$y = h * x = \begin{bmatrix} h[1] & 0 & \cdots & 0 & 0 \\ h[2] & h[1] & \cdots & 0 & 0 \\ h[3] & h[2] & \cdots & 0 & 0 \\ \vdots & h[3] & \cdots & h[1] & 0 \\ h[m-1] & \vdots & \ddots & h[2] & h[1] \\ h[m] & h[m-1] & & \vdots & h[2] \\ 0 & h[m] & \ddots & h[m-2] & \vdots \\ 0 & 0 & \cdots & h[m-1] & h[m-2] \\ \vdots & \vdots & & h[m] & h[m-1] \\ 0 & 0 & \cdots & 0 & h[m] \end{bmatrix} \begin{bmatrix} x[1] \\ x[2] \\ x[3] \\ \vdots \\ x[N] \end{bmatrix} \tag{3.17}$$

This depiction of convolution as matrix transformation is useful as an alternative mathematical perspective on the convolution operation, and it was used in the implementation part of this work as calculating the loss gradient over matrix multiplication is already built into the Julia Flux framework.

Proposition 3.9 (Properties of the Convolution operation)**.** *The following properties hold in both the continuous and the discrete case. For either $f, g, h \in L^1(\mathbb{R}^d)$ or f, g, h complex-valued functions on $\mathbb{Z}$.*

1. *Commutativity*

$$f * g = g * f$$

2. *Associativity*

$$f * (g * h) = (f * g) * h$$

3. *Distributivity*

$$f * (g + h) = (f * g) + (f * h)$$

4. *Associativity with scalar multiplication: For* $a \in \mathbb{C}$

$$a(f * g) = (af) * g$$

5. *Identity: Convolution with the delta distribution* δ *is simply the identity operation*

$$f * \delta = f$$

Convolution and the Fourier transform are connected by the following convolutional theorem, which states that the Fourier transform translates the convolution operation in the time domain to a simple multiplication in the frequency domain.

Theorem 3.10 (Convolutional Theorem)**.** *Let* $f, g \in L^1(\mathbb{R}^d)$ *be integrable functions on* $\mathbb{R}^d$ *with Fourier transforms* $F = \mathscr{F}\{f\}$ *and* $G = \mathscr{F}\{g\}$*. Define their convolution* $h = f * h$ *and its Fourier transform* $H = \mathscr{F}\{f * h\}$*. The convolutional theorem states that*

$$H(k) = F(k) \cdot G(k) \tag{3.18}$$

where $\cdot$ *denotes pointwise multiplication.*

Applying the inverse FT $\mathscr{F}^{-1}$ *immediately gives us the corollary*

$$h(x) = \mathscr{F}^{-1}\{F \cdot G\}(x). \tag{3.19}$$

The convolutional theorem also holds in the discrete case where f, g *complex-valued signals and* $\mathscr{F}$ *denotes the DFT and* $\mathscr{F}^{-1}$ *the IDFT.*

The convolutional theorem allows us to efficiently compute convolutions. Note that both the definition in Eq. 3.16 and the matrix multiplication in Eq. 3.17 require N^2 operations for N output values. Using the convolutional theorem and the FFT algorithm the computational complexity can be reduced to $\mathcal{O}(N \log N)$.

If we recall the depiction of the DFT as a unitary transformation in section 3.2.1 and the convolution matrix in Eq. 3.17, it follows from the convolutional theorem that the DFT diagonalizes convolutional matrices. The Fourier basis is the eigenbasis of the convolution operation.

3.5 Deconvolution

Deconvolution is the inverse operation to convolution. Given the functions g and h, the objective of deconvolution is to find f given the convolution equation

$$f * g = h. \tag{3.20}$$

In the simple case of 3.20 we can simply use the convolutional theorem 3.10 to solve the equation in Fourier space.

$$F(k)G(k) = H(k) \tag{3.21}$$

$$F(k) = \frac{H(k)}{G(k)} \tag{3.22}$$

$$f(x) = \mathscr{F}^{-1}\left(\frac{H}{G}\right)(x) \tag{3.23}$$

In practice, we however do not have the ideal case shown in 3.20. Usually, h is a recorded signal of a physical system and f is the signal we wish to recover, but that has been convolved by a *filter* or *impulse response* g when measuring the signal. In real measured systems we typically have a measurement error term that has to be added to 3.20, resulting in

$$(f * g) + \epsilon = h \tag{3.24}$$

In this case ϵ is the noise term added to our recorded signal. Using the *inverse filtering* approach in Eq. 3.23 is now no longer correct. In fact, the signal-noise-ratio is typically worsened and high frequency terms are erroneously amplified. The problems of deconvolution have long been known in the physics community, a discussion can be found in [31].

3.5.1 Wiener Deconvolution

To address the issues of the simple inverse filtering approach, the so-called Wiener Deconvolution was introduced in [32]. This approach again is applied in the frequency domain and attempts to minimize the impact of deconvolved noise at frequencies which have a poor signal-to-noise ratio.

Given a system as in Eq. 3.24, the Wiener deconvolution attempts to provide filter k to estimate f as follows

$$\hat{f}(t) = (k * h)(t) \tag{3.25}$$

where $\hat{f}(t)$ is an estimate of $f(t)$ that minimizes the mean squared error (MSE). The Wiener deconvolution filter is naturally written in the Fourier domain as

$$K(k) = \frac{G^*(k)S(k)}{|G(k)|^2 S(k) + N(k)} \tag{3.26}$$

where

- $K(k)$ and $G(k)$ are the Fourier transforms of $k(t)$ and $g(t)$
- $S(k) = \mathbb{E}[|F(k)|^2]$ is the mean power spectral density of the original signal $f(t)$
- $N(k) = \mathbb{E}[|\mathcal{E}(k)|^2]$ is the mean power spectral density of the noise $\epsilon(t)$
- the superscript * denotes complex conjugation.

The filtering operation can then be immediately carried out in the Fourier domain:

$$\hat{F}(k) = K(k)H(k) \tag{3.27}$$

Applying the inverse Fourier transform then gives us the estimate of the original signal

$$\hat{f}(t) = \mathscr{F}^{-1}(\hat{F})(t) \tag{3.28}$$

An important limitation of this approach is that estimates of the exact impulse response of the system and the mean power spectral density (PSD) of both the noise and the original, unknown signal are needed to perform the deconvolution. In the case of BOLD time series the impulse response is known, it is the hemodynamic response function. The PSDs of the underlying signal and the noise are however not known. In the next section on wavelets I will introduce the mathematical tools needed to obtain estimates of the original signal and the noise. These will allow us to apply Wiener deconvolution to BOLD time series.

Chapter 4

Wavelet Transformation

This chapter provides a comprehensive overview of both the continuous and discrete wavelet transforms. Firstly, I will define Wavelets and showcase the mathematical theory behind the continuous wavelet transform (CWT). From there I will cover the discrete wavelet transform (DWT) and the fast wavelet transform, which allows for efficient computation of the DWT. With all that mathematical machinery in place I will then discuss how to use the wavelet transformation for noise estimation and denoising in a simple and effective manner.

I will note here that definitions and notations of wavelet theory are not consistent across literature. This issue is compiled by the fact that authors also use different definitions for the Fourier transform (i.e. different placing of the normalization factor), which underlies much of the wavelet theory. I followed [33], [34] and [35] for the definitions and theorems presented in this chapter but I did have to adapt them in part to be consistent.

4.1 Continuous Wavelet Transformation

The fundamental idea behind the wavelet transformation is to analyze through the scale. Wavelets are used in the wavelet transformation the same way sin and cos are used in the Fourier transformation. In the Fourier transformation, a function is transformed to a new basis given by sin and cos functions in frequency space. Note that the sin and cos functions do not appear in the definition of FT introduced earlier because we only used the e^{ikx}. These functions are connected with the identity $e^{ikx} = cos(kx) + i\sin(kx)$.

The basis functions of the Fourier transform are localized in the frequency domain, each sin/cos has a single frequency, but not in the time domain, where they are periodic and non-vanishing. This means that for periodic, continuous signals the Fourier transform

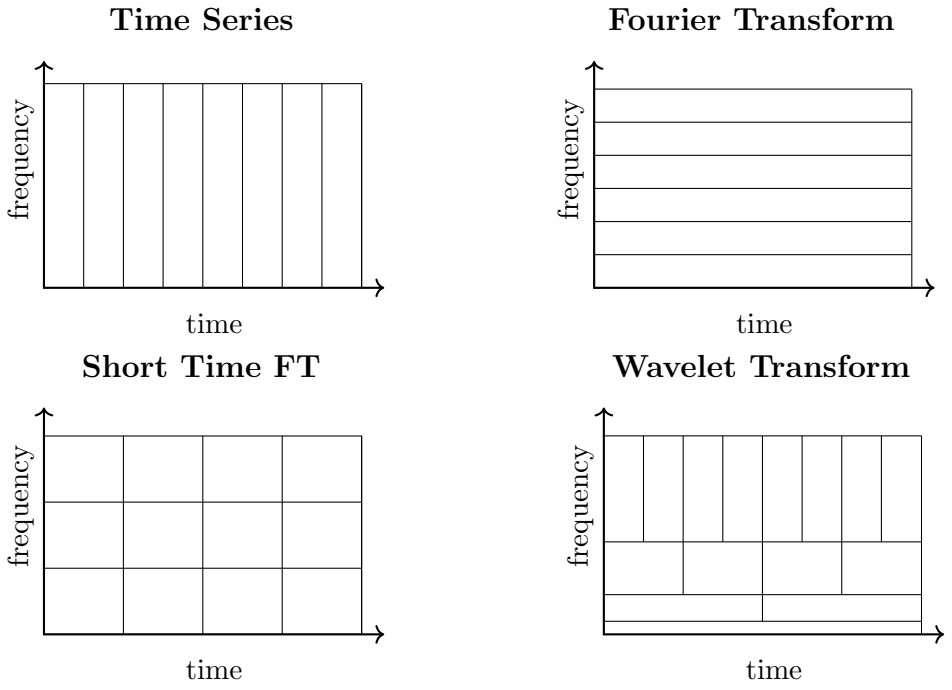


FIGURE 4.1: A schematic overview of the time and frequency resolutions of the different transformations in comparison with the original time-series dataset. The size and orientation of the blocks indicate how small the features are that we can distinguish in the time and frequency domain.

only needs a small amount of coefficients to obtain a good approximation of the signal. In addition, noise limited to certain frequencies can be easily detected and filtered.

Local signals containing discontinuities, on the other hand, cannot be approximated easily and one needs a large amount of Fourier coefficients to approximate them well. Furthermore, localized time information is stored in the phase of the Fourier transformation, so modifications to the signals in frequency space e.g. filtering may corrupt transient time information, leading to unwanted side effects.

These problems are addressed by wavelet transformations. Wavelets are, just like sin and cos in the Fourier transformation, used as basis functions to represent a signal in the frequency domain. The major difference is that wavelets are localized in time and frequency domain. Therefore, wavelets are well-suited to approximate data containing sharp discontinuities. Another advantage of wavelets is that their width in time and frequency can vary. This way one has long low frequency basis functions for frequency analysis and short basis functions to isolate discontinuities. The idea is illustrated in figure 4.1.

For a more mathematically precise formulation, let us first define a Wavelet

Definition 4.1 (Wavelet)**.** A function $\psi \in L^2(\mathbb{R})$ is called a Wavelet if it satisfies the following conditions

1. mean free $\int_{\mathbb{R}} \psi(t)dt = 0$
2. normalized $\|\psi\|_2 = \sqrt{\int_{\mathbb{R}} \psi(t)\psi^*(t)dt} = 1$

In many cases an additional condition is required:

3. admissibility $\int_{\mathbb{R}} \frac{\hat{\psi}(\omega)\hat{\psi}^*(\omega)}{|\omega|}d\omega < \infty$

A function ψ satisfying the definition is called the *mother-wavelet*. From this *mother-wavelet* we can generate a family of functions, the so-called *Daughter-Wavelets*, as follows

$$\Psi = \left\{ \psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) | a \in (0,\infty), b \in \mathbb{R} \right\} \tag{4.1}$$

Note that $\psi \in L^2(\mathbb{R})$ implies $\psi_{a,b} \in L^2(\mathbb{R})$. Furthermore, the normalization is preserved

$$\|\psi_{a,b}\|_2^2 = \frac{1}{a}\int_{\mathbb{R}} \left|\psi\left(\frac{t-b}{a}\right)\right|^2 = \int_{\mathbb{R}} |\psi(u)|^2 du = \|\psi\|_2^2 \tag{4.2}$$

and the Fourier transform of the *daughter-wavelet* $\psi_{a,b} \in L^2(\mathbb{R})$ is given by

$$\begin{aligned} \hat{\psi}_{a,b}(w) &= \int_{\mathbb{R}} \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) e^{-i\omega t}dt \\ &= \sqrt{a}e^{ib\omega}\int_{\mathbb{R}} \psi(\tau)e^{-ia\omega\tau}d\tau \\ &= \sqrt{a}e^{ib\omega}\hat{\psi}(a\omega) \end{aligned} \tag{4.3}$$

The parameter a is the scaling parameter, which measures the degree of compression or scale, and b is the translation parameter, measuring the time location of the wavelet.

Using these definitions, we can formally define the wavelet transform as follows:

Definition 4.2 (Continuous wavelet transform)**.** Let ψ be a Wavelet as defined in 4.1 and $f \in L_2(\mathbb{R})$. The wavelet transform of f is defined as

$$\mathscr{W}_\psi\{f\}(a,b) = \int_{\mathbb{R}} f(t)\frac{1}{\sqrt{a}}\psi^*\left(\frac{t-b}{a}\right)dt \tag{4.4}$$

where * is the complex conjugate, $a \in (0,\infty)$ and $b \in \mathbb{R}$.

In the following proposition I have compiled some fundamental properties of the wavelet transform.

Proposition 4.3 (Properties of the wavelet transform)**.** *Let $\alpha, \beta \in \mathbb{C}, \delta \in \mathbb{R}, \gamma > 0$ be scalars, ψ, ϕ admissible wavelets and $f, g \in L^2(\mathbb{R})$. Then the following properties hold*

1. *Linearity* $\mathscr{W}_\psi\{\alpha f + \beta g\}(a,b) = \alpha \mathscr{W}_\psi\{f\}(a,b) + \beta \mathscr{W}_\psi\{g\}(a,b)$
2. *Translation* $\mathscr{W}_\psi\{\tau_\delta f\}(a,b) = \mathscr{W}_\psi\{f\}(a, b+\delta)$
3. *Dilation* $\mathscr{W}_\psi\{\sigma_\gamma f\}(a,b) = \mathscr{W}_\psi\{f\}(\frac{a}{\gamma}, \frac{b}{\gamma})$
4. *Symmetry* $\mathscr{W}_\psi\{f\}(a,b) = \mathscr{W}_f\{\psi\}^*(\frac{1}{a}, \frac{b}{a})$
5. *Parity* $\mathscr{W}_{P\psi}\{Pf\}(a,b) = \mathscr{W}_\psi(a,-b)$
6. *Anti-linearity* $\mathscr{W}_{\alpha\psi+\beta\phi}\{f\}(a,b) = \alpha^* \mathscr{W}_\psi\{f\}(a,b) + \beta^* \mathscr{W}_\phi\{f\}(a,b)$
7. *Translation in wavelet:* $\mathscr{W}_{\tau_\delta\psi}\{f\}(a,b) = \mathscr{W}_\psi\{f\}(a, b-a\delta)$
8. *Dilation in wavelet:* $\mathscr{W}_{\sigma_\gamma\psi}\{f\}(a,b) = \mathscr{W}_\psi(\gamma a, b)$.

Some important notes on the definition 4.2

1. The definition reminds us of the definition of the Fourier transform 3.1. The kernel $\psi_{a,b}$ plays the same role as the kernel e^{ikx} in the FT. The kernel of the wavelet transform is however a two-parameter family of functions determining the scale and location of the wavelet.
2. Both the Fourier transform and the wavelet transform are linear transformations. But unlike the Fourier transform, the wavelet transform is not a single transform, but a group of transforms determined by the underlying wavelet.
3. For a fixed scale a, the wavelet transform $\mathscr{W}_\psi\{f\}(a,b)$, as a function of b, represents the detail information contained in the signal f at scale a.
4. If the mother wavelet $\psi(t)$ posses n-vanishing moments, we can use the Taylor expansion of f at $a = b$ and insert it in 4.4

$$\mathscr{W}_\psi\{f\}(a,b) = f(b)\int_{\mathbb{R}} \psi^*_{a,0}(t-b)dt + f^{(1)}(b)\int_{\mathbb{R}}(t-b)\psi^*_{a,0}(t-b) + \frac{f^{(2)}(b)}{2}\int_{\mathbb{R}}(t-b)^2\psi^*_{a,0}(t-b) + \cdots + \frac{f^{(n)}(b)}{n!}\int_{\mathbb{R}}(t-b)^n\psi^*_{a,0}(t-b) + \cdots \quad (4.5)$$

It follows that the first n terms of 4.5 vanish and do not contribute to $\mathscr{W}_\psi\{f\}(a,b)$.

5. We can rewrite the wavelet transform using the Fourier transform.

$$\mathscr{W}_\psi\{f\}(a,b) = \sqrt{a}\int_{\mathbb{R}} \hat{f}(k)\hat{\psi}^*(ak)e^{ibk}dk. \tag{4.6}$$

6. We can Fourier transform the wavelet transform $\mathscr{W}_\psi\{f\}(a,b)$ as a function of b using the equation 4.6 above

$$\mathscr{F}\left\{\mathscr{W}_\psi\{f\}(a,b)\right\}(k) = \sqrt{a}\hat{f}(k)\hat{\psi}^*(ak) \tag{4.7}$$

Before continuing we need another important definition that relates two wavelets to another:

Definition 4.4 (Cross admissibility)**.** The wavelets $\psi, \phi \in L_2(\mathbb{R})$ are called cross admissible if

$$C_{\psi,\phi} := \int_{\mathbb{R}} \frac{\hat{\psi}^*(\omega)\hat{\phi}(\omega)}{|\omega|}d\omega < \infty \tag{4.8}$$

$C_{\psi,\phi}$ is called the admissibility constant. In the case of $\psi = \phi$ we simply write C_ψ.

If a wavelet ψ is real-valued, then its Fourier transform $\hat{\psi}$ is symmetric and the admissibility constant can be calculated by only integrating over positive values

$$C_{\psi,\phi} = 2\int_{\mathbb{R}^+} \frac{\hat{\psi}^*(\omega)\hat{\phi}(\omega)}{\omega}d\omega \tag{4.9}$$

Using this definition we can get an orthogonality relation for the wavelet transform similar to Theorem 3.6 for the Fourier transform.

Theorem 4.5. *Let $\mathscr{W}_\psi\{f\}$ and $\mathscr{W}_\phi\{g\}$ be the wavelet transforms of a pair of function $f, g \in L_2(\mathbb{R})$ with respect to the wavelets ψ and ϕ. Then, the following orthogonality relation holds*

$$\int_{\mathbb{R}}\int_{\mathbb{R}} \mathscr{W}_\psi\{f\}(a,b)\mathscr{W}_\phi\{g\}^*(a,b)\frac{da\,db}{a^2} = C_{\psi,\phi}\langle f, g\rangle_2 \tag{4.10}$$

If ψ and ϕ are real wavelets, then we can modify the integration such that only the "natural" positive values of the scaling factor a are needed.

$$\int_{\mathbb{R}}\int_{\mathbb{R}^+} \mathscr{W}_\psi\{f\}(a,b)\mathscr{W}_\phi\{g\}^*(a,b)\frac{da\,db}{a^2} = \frac{C_{\psi,\phi}}{2}\langle f, g\rangle_2 \tag{4.11}$$

The proof is given in A.1.1 and is very straightforward, simply applies the frequency representation of the WT and properties of the Dirac delta. Note that choosing $\psi = \phi$ and $f = g$ yields the energy preserving relation (up to a constant)

$$\int_{\mathbb{R}}\int_{\mathbb{R}} |\mathscr{W}_\psi\{f\}(a,b)|^2 \frac{da db}{a^2} = C_\psi \|f\|_2^2 \tag{4.12}$$

This orthogonality relation 4.5 allows a simple reconstruction of the original signal, i.e. we can define the inverse wavelet transform.

Definition 4.6 (Inverse continuous wavelet transform)**.** Let ψ, ϕ be wavelets as defined in 4.1 with cross admissibility constant $C_{\psi,\phi} \neq 0$ and $f \in L_2(\mathbb{R})$. The inverse wavelet transform of f is defined as

$$f(t) = \frac{1}{C_{\psi,\phi}} \int_{\mathbb{R}}\int_{\mathbb{R}} W_\psi\{f\}(a,b) \frac{1}{\sqrt{|a|}} \phi\left(\frac{t-b}{a}\right) \frac{da\, db}{a^2} \tag{4.13}$$

If ψ, ϕ are real-valued, then we can again simplify the expression to only include the integration over positive a values.

$$f(t) = \frac{2}{C_{\psi,\phi}} \int_{\mathbb{R}}\int_{\mathbb{R}^+} W_\psi\{f\}(a,b) \frac{1}{\sqrt{|a|}} \phi\left(\frac{t-b}{a}\right) \frac{da\, db}{a^2} \tag{4.14}$$

The proof is shown in the appendix in A.1.2. Note that the inverse wavelet theorem with these assumptions only applies in the L_2 sense, meaning the inverse is equal almost everywhere, but not pointwise.

Stronger, pointwise inversion theorems also exist, but they have more requirements for the wavelets. For more details see e.g. [36]. An interesting property of the wavelet transformation is that can also be inverted with a different wavelet, given they fulfill the *cross admissibility* criterion. The wavelet with which the transformation to wavelet space is applied is called the *analyzing wavelet*, whereas the wavelet used for backtransform is called the *reconstruction wavelet*. This indicates that, in some sense, analysis and reconstruction are independent of the choice of the mother wavelet.

4.2 Discrete Wavelet Transformation

In this section we consider the *discrete wavelet transform*. Rather than using continuous integrals, as required for the CWT, we will switch to sums of discrete wavelet coefficients. This will allow us to transform discrete input signals we encounter in practice.

4.2.1 Frames and orthogonal wavelet basis

In the previous section we defined a wavelet at scale a and location b as

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{4.15}$$

A natural way to discretize this wavelet is to link b to a. We move in discrete steps to each location b which are proportional to the scale a. This discretization has the form

$$\psi_{m,n}(t) = \frac{1}{\sqrt{a_0^m}}\psi\left(\frac{t-nb_0a_0^m}{a_0^m}\right) \tag{4.16}$$

where the integers m and n control the dilation and translation respectively; $a_0 > 1$ is a specified fixed dilation step parameter and $b_0 > 0$ is the location parameter. From the above equation it can be seen that the size of the translation steps $\delta b = b_0 a_0^m$ is directly proportional to the wavelet scale a_0^m.

The most common choice is a dyadic grid with parameters $a_0 = 2$ and $b_0 = 1$ as it is simple and allows for efficient practical implementation as it lends itself to a fast computer algorithm. Eq. 4.16 on the dyadic grid can be compactly written as

$$\psi_{m,n} = 2^{-m/2}\psi(2^{-m}t - n) \tag{4.17}$$

From here on in this work we will use $\psi_{m,n}(t)$ to denote the dyadic grid scaling.

The wavelet transform of a continuous signal $x(t) \in L_2(\mathbb{R})$ using discrete wavelets is then

$$T_{m,n} = \langle x, \psi_{m,n}\rangle_2 = \int_{\mathbb{R}} x(t)\frac{1}{a_0^{m/2}}\psi^*(a_0^{-m}t - n_b0)\,dt \tag{4.18}$$

where $T_{m,n}$ are the discrete wavelet transform values given on a scale-location grid of index m, n. The values $T_{m,n}$ are known as *wavelet* or *detail coefficients*. The family of wavelets given by Eq. 4.16 over all integers $m, n \in \mathbb{Z}$ define a *frame* on $L_2(\mathbb{R})$ if the energy of the resulting wavelet lies within a certain bounded range of energy of the original signal.

$$A\|x\|_2^2 \leq \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty}|T_{m,n}|^2 \leq B\|x\|_2^2 \tag{4.19}$$

The quality of the signal representation in wavelet space can be measured by the *tightness* of the frame. If $A = B$ a frame is called *tight* and tight frames have the following reconstruction formula

$$x(t) = \frac{1}{A} \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} T_{m,n} \psi_{m,n}(t) \tag{4.20}$$

For $A > 1$ the tight frame is redundant, A being a measure of the redundancy. When $A = B = 1$ the wavelet family forms an orthonormal basis. These wavelets are both orthogonal to each other and are normalized

$$\langle \psi_{m',n'}, \psi_{m,n} \rangle_2 = \delta_{m',m} \delta_{n',n}. \tag{4.21}$$

This means that the information stored in the wavelet coefficient $T_{m,n}$ is not repeated elsewhere and allows for complete reconstruction of the signal without redundancy.

For $A \neq B$ a reconstruction formula can still be given

$$\tilde{x}(t) = \frac{2}{A+B} \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} T_{m,n} \psi_{m,n}(t) \tag{4.22}$$

but the reconstructed signal $\tilde{x}(t)$ will have an error, which depends on the frame bounds A, B. The error decreases for tighter frame bounds, so for $\frac{B}{A}$ closer to 1.

4.2.2 The scaling function

Orthonormal dyadic discrete wavelets are associated with *scaling functions* and their dilation equations. The scaling function is associated with the smoothing of the signal and has the same form as the wavelets

$$\phi_{m,n}(t) = 2^{-m/2} \phi(2^{-m} t - n) \tag{4.23}$$

and they have the property

$$\int_{\mathbb{R}} \phi_{0,0}(t)\, dt = 1 \tag{4.24}$$

where $\phi_{0,0} = \phi$ is referred to as the *father wavelet*. Note the difference to the wavelet definition 4.1 in which the integral vanishes. The scaling function is orthogonal to

translations of itself, but not to dilations of itself. The scaling function can be convolved with the signal to produce *approximation coefficients* as follows

$$S_{m,n} = \int_{\mathbb{R}} x(t)\phi_{m,n}(t)\,dt \tag{4.25}$$

The approximation coefficients at a specific scale m are known as the *discrete approximation* of the signal at scale m. A *continuous approximation* of the signal at scale m can be calculated by summing a sequence of scaling functions at this scale multiplied by the approximation coefficients:

$$x_m(t) = \sum_{n=-\infty}^{\infty} S_{m,n}\phi_{m,n}(t) \tag{4.26}$$

where $x_m(t)$ is a smooth, scaling-function-dependent version of the signal $x(t)$ at scale m. This approximation approaches $x(t)$ at small scales $m \to -\infty$.

This can be used to represent $x(t)$ as a series expansion of both the detail and approximation coefficients at a scale m_0

$$x(t) = \sum_{n=-\infty}^{\infty} S_{m0,n}\phi_{m_0,n}(t) + \sum_{n=-\infty}^{m0}\sum_{n=-\infty}^{\infty} T_{m,n}\psi_{m,n}(t) \tag{4.27}$$

Using Eq. 4.26 and signal detail at scale m

$$d_m(t) = \sum_{n=-\infty}^{\infty} T_{m,n}\psi_{m,n}(t) \tag{4.28}$$

we can rewrite Eq. 4.27 into a simpler form

$$x(t) = x_{m_0} + \sum_{m=-\infty}^{m_0} d_m(t). \tag{4.29}$$

From this equation it is easy to show that

$$x_{m-1}(t) = x_m(t) + d_m(t) \tag{4.30}$$

This means that adding the signal detail at an arbitrary scale m to the approximation at that scale results in the signal approximation at a smaller scale $m-1$. This is called *multiresolution representation.*

4.2.3 The scaling equation

To connect the scaling function to the wavelet equation, we use the *scaling equation* describe the scaling function $\phi(t)$ to in terms of contracted and shifted versions of itself:

$$\phi(t) = \sum_k c_k \phi(2t - k) \tag{4.31}$$

where $\phi(2t - k)$ is a contracted version of $\phi(t)$ shifted along the time axis by an integer k and c_k is an associated scaling coefficient. For simplicity, we will limit ourselves to wavelets with compact support for the rest of this section as these have finite nonzero scaling coefficients. The scaling coefficients must fulfill the following constraint

$$\sum_k c_k = 2 \tag{4.32}$$

In addition, in order to create an orthogonal system we require that

$$\sum_k c_k c_{k+2k'} = \begin{cases} 2, & \text{if } k' = 0 \\ 0, & \text{else} \end{cases} \tag{4.33}$$

The scaling coefficients c_k are used in reverse with alternate signs when creating the associated wavelet equation

$$\psi(t) = \sum_k (-1)^k c_{1-k} \phi(2t - k) \tag{4.34}$$

This wavelet equation is commonly seen in literature. Because we limited ourselves to wavelets with compact support here, we can rewrite it to

$$\psi(t) = \sum_k (-1)^k c_{N_k - 1 - k} \phi(2t - k). \tag{4.35}$$

where N_k is the finite number of nonzero scaling coefficients. This ordering of scaling coefficients allows for our wavelets and their corresponding scaling equation to have support over the same interval $[0, N_k - 1]$. The coefficients in Eq. 4.35 are more compactly written as $b_k = (-1)^k c_{N_k - 1 - k}$. Combining the equation of the scaling function 4.23 and the scaling equation 4.31 yields

$$2^{-(m+1)/2}\phi\left(\frac{t}{2^{m+1}-n}\right) = 2^{-(m+1)/2}\sum_k c_k\phi\left(\frac{2t}{2\times 2^m} - 2n - k\right). \tag{4.36}$$

This equation may be written more compactly as

$$\phi_{m+1,n}(t) = \frac{1}{\sqrt{2}}\sum_k c_k\phi_{m,2n+k}(t) \tag{4.37}$$

Thus, the scaling function at an arbitrary scale is composed of a sequence of shifted scaling functions at the next smaller scale. Similarly, one obtains

$$\psi_{m+1,n}(t) = \frac{1}{\sqrt{2}}\sum_k b_k\phi_{m,2n+k}(t) \tag{4.38}$$

4.2.4 The fast wavelet transform

Now we have the pieces in place to formulate the fast wavelet transform. For this we rewrite the approximation coefficients at scale $m+1$ defined in Eq. 4.25 with Eq. 4.37

$$\begin{aligned} S_{m+1,n} &= \int_{\mathbb{R}} x(t)\phi_{m+1,n}(t)\,dt \\ &= \int_{\mathbb{R}} x(t)\left[\frac{1}{\sqrt{2}}\sum_k c_k\phi_{m,2n+k}(t)\right]\,dt \\ &= \frac{1}{\sqrt{2}}\sum_k c_k\left[\int_{\mathbb{R}} x(t)\phi_{m,2n+k}(t)\,dt\right] \end{aligned} \tag{4.39}$$

The integral in brackets gives the approximation coefficients $S_{m,2n+k}$ for each k allowing us to rewrite it as follows

$$S_{m+1,n} = \frac{1}{\sqrt{2}}\sum_k c_k S_{m,2n+k} = \frac{1}{\sqrt{2}}\sum_k c_{k-2n}S_{m,k} \tag{4.40}$$

Hence, using this equation we can generate the approximation coefficients at the scale $m+1$ using the scaling coefficients at the previous scale m. In the same manner we arrive at a recursive equation for the wavelet coefficients

$$T_{m+1,n} = \frac{1}{\sqrt{2}}\sum_k b_k S_{m,2n+k} = \frac{1}{\sqrt{2}}b_{k-2n}S_{m,k} \tag{4.41}$$

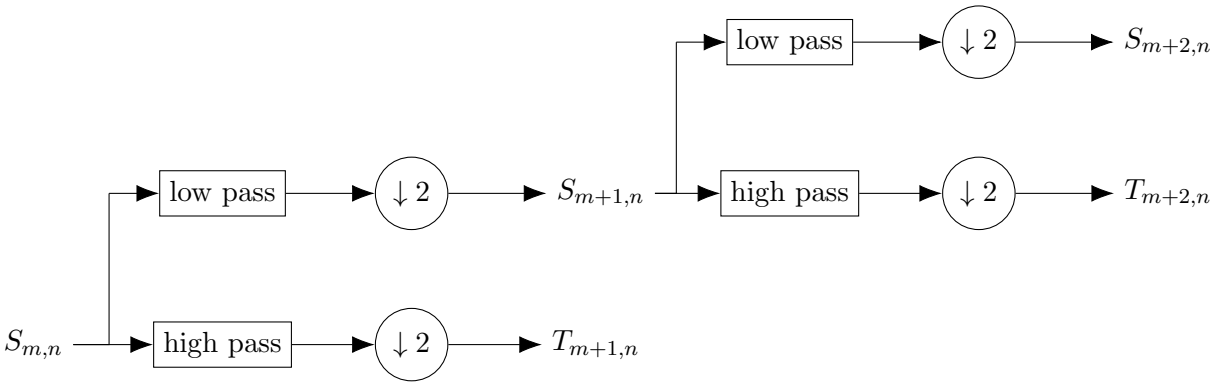


FIGURE 4.2: **Schematic of DWT filtering.** Diagram of the filtering of the approximation coefficients to produce the approximation and detail coefficients at successive scales. The subsample symbold $\downarrow 2$ means take every second value of the filter signal.

With the equations, given the approximation coefficients $S_{m_0,n}$ at a scale m_0, we can compute the detail and approximation coefficients for all scales $m > m_0$. Notice we do not even need the underlying signal $x(t)$ anymore, only $S_{m_0,n}$.

Equations 4.40 and 4.41 represent the *multiresolution decomposition algorithm*, the first half of the fast wavelet transform. Iterating these equations performs a high-pass and low-pass filtering of the input (i.e. the coefficients $S_{m,2n+k}$) to get the outputs ($S_{m+1,n}$ and $T_{m+1,n}$). The vectors containing $(1/\sqrt{2})c_k$ and $(1/\sqrt{2})b_k$ represent filters: $(1/\sqrt{2})c_k$ is a low-pass filter, letting through low signal frequencies and $(1/\sqrt{2})b_k$ is a high-pass filter, letting through the high frequencies corresponding to the signal details. The resulting coefficients, both approximation and detail, are then subsampled (or downsampled) where every second value is chosen. The detail components $T_{m+1,n}$ are kept and the approximation components $S_{m+1,n}$ are again passed through the next set of low-pass and high-pass filters. The process is repeated over all scales to give the full decomposition of the signal. This process is shown schematically in figure 4.2.

We can also go the opposite direction and reconstruct $S_{m,n}$ from $S_{m+1,n}$ and $T_{m+1,n}$. After some algebra one obtains the equation for the reconstruction algorithm

$$S_{m-1,n} = \frac{1}{\sqrt{2}} \sum_k c_{n-2k} S_{m,k} + \frac{1}{\sqrt{2}} \sum_k b_{n-2k} T_{m,k} \tag{4.42}$$

At the scale $m-1$ the approximation coefficients can be found in terms of a combination of approximation and detail coefficients at the next scale m. The reconstruction algorithm is the second half of the *fast wavelet transform* (FWT).

So far we have worked with the assumption of a continuous input signal $x(t)$ for the entire derivation of the DWT. To begin the fast wavelet transform algorithm we need an initial set of approximation coefficients $S_{0,n}$ which are calculated using the continuous signal,

see Eq. 4.25. In practice, we don't have access to $x(t)$ but only to a discrete sample $x[t]$. It is common practice to simply use the input signal $x[t]$ directly as approximation coefficients at scale $m = 0$. This is formally not correct, as $S_{0,n}$ is a weighted average of $x(t)$ in the vicinity of n, but it works well in most cases. So as initial input to the filter bank simple set

$$S_{0,n} = x[n] \tag{4.43}$$

with both coefficient index and signal index have the same range $[0, T]$ and are equal to each other. There are also more elaborate preprocessing schemas to obtain a better set of initial approximation coefficients. A discussion of the issue can, for example, be found in [37].

Note that in literature the fast wavelet transform, discrete wavelet transform, decomposition/reconstruction algorithm, fast orthogonal wavelet transform, multiresolution algorithm, pyramid algorithm etc. are used interchangeably. Also note that other discretizations of the wavelet transform are also referred to as discrete wavelet transform.

4.2.5 Noise estimation and Denoising

Now we have the introduced the mathematical machinery of the wavelet transform, we can use it to address the problems we encountered in section 3.5.1. To be more precise, we needed an estimate of the noise level and an estimate of the signal without noise (or at least of the respective power spectra) as inputs to the Wiener deconvolution filter. Based on papers by Donoho [38], [39] I will present a method for denoising called *wavelet thresholding* or *VISUSHRINK*.

This method assumes we have a signal with added Gaussian white noise $f + \epsilon$. It can be observed that the noisy components of the transform signal are mostly contained in the smallest scale wavelet coefficients with low absolute values. These small coefficients are simply set to zero and then the signal is reconstructed, giving an estimate of the denoised signal $\tilde{f}$. Donoho shows in their work that this thresholding technique gives a good estimator of f both in terms of minimizing the mean-squared error and $\tilde{f}$ being smooth. Because of the smooth results wavelet thresholding was greatly used in the image processing domain.

The fact that the empirical wavelet coefficients at the finest scale are essentially pure noise is also used for the noise estimation. A robust noise estimator is the median of the absolute deviation (MAD) of the smallest scale wavelet coefficients. In the Gaussian case this is calibrated to $\tilde{\sigma} = \frac{\text{MAD}}{0.6745}$. The threshold value is then set accordingly to

$$\lambda_U = (2 \ln N)^{1/2} \tilde{\sigma} = \frac{(2 \ln N)^{1/2} \mathrm{MAD}}{0.6745} \tag{4.44}$$

This is the so-called *universal threshold* and is both simple and commonly used. There are also different wavelet denoising schemas based on thresholding, such as *soft-thresholding*, the *minimax* method, various Bayesian approaches and others but hard thresholding showed good results for our use case while being both simple and robust.

A full description of the algorithm used to estimate the noise level and the denoised data is given in Algorithm 2.

Algorithm 2 The VISUSHRINK Algorithm

Input: time series data x_i of length N; analyzing wavelet ψ

1. Apply the DWT to the input data x_i to obtain the wavelet coefficients Θ_i^0

$$\Theta_i^0 = \mathscr{W}_\psi\{x_i\} \tag{4.45}$$

2. Calculate the MAD of the fine scale coefficients Θ_i^0, the estimate of the noise level $\tilde{\sigma}$ and the universal estimator λ_U. Let $\bar{\Theta} = \mathrm{median}(\Theta_i^0)$ be the median of the fine scale coefficients and N the length of the time series x_i

$$\mathrm{MAD} = \mathrm{median}(|\Theta_i^0 - \bar{\Theta}|) \tag{4.46}$$

$$\tilde{\sigma} = \frac{\mathrm{MAD}}{0.6745} \tag{4.47}$$

$$\lambda_U = (2 \ln N)^{1/2} \tilde{\sigma} \tag{4.48}$$

3. Apply hard thresholding to the fine scale wavelet coefficients

$$\Theta_i^1 = \begin{cases} 0, & |\Theta_i^0| < \lambda_U \\ \Theta_i^0, & |\Theta_i^0| \geq \lambda_U \end{cases} \tag{4.49}$$

4. Apply the inverse DWT to the tresholded coefficients Θ_i^1 to obtain an estimate of the denoised signal $\tilde{x_i}$

$$\tilde{x_i} = \mathscr{W}_\psi^{-1}\{\Theta_i^1\} \tag{4.50}$$

Output: estimate of the noise level $\tilde{\sigma}$; estimate of denoised signal $\tilde{x_i}$

Now we have established all the tools necessary for the inversion of the BOLD observation model. In the next chapter I will combine these and present the full algorithm for training on fMRI data.

Chapter 5

Datasets, Training and Evaluation

In this chapter I will first give a full description of the BOLD inversion teacher forcing algorithm I developed. Then I will showcase the datasets on which the models were trained, delve into the specifics of the training routines utilized, and discuss the evaluation measures I employed to assess the performance of the trained models.

5.1 BOLD inversion teacher forcing

We recall from section 2.12.3 the BOLD observation equation

$$\hat{x}_t = O(z_t, r_t) = \boldsymbol{B}\left((hrf * z)_t\right) + \boldsymbol{J}r_t + \epsilon_t. \tag{5.1}$$

To apply the teacher forcing algorithm we need to generate desired target values d_t for the latent states z_t from the ground truth x_t

$$\{d_t\} = O^{-1}(\{x_t, r_t\}) = \left\{\text{Conv}^{-1}\left(\boldsymbol{B}^{+}\left(x_t - \boldsymbol{J}r_t\right), hrf\right)\right\} \tag{5.2}$$

where $\text{Conv}^{-1}(\cdot, hrf)$ is the deconvolution of data with the hrf. Note that we to perform these steps on an entire series $\{x_t\}$ because there is no one-to-one correspondence between a time point t of the unconvoluted and convoluted time series.

The issue with solving the inversion with equation 5.2 is that the deconvolution has to be performed in every epoch of training because B is learned and doesn't stay constant. The trick is to notice that the same hrf function is applied to all latent space dimensions, hence one can interchange the matrix multiplication with $\boldsymbol{B}$ and the convolution step.

Let $Z_{i,t} \in \mathbb{R}^{M \times T}$ be the data matrix of the M-dimensional time series of length T, $hrf[t] = h_t$ be the hrf vector at an arbitrary but fixed repeat time and $\boldsymbol{B} = b_{ij} \in \mathbb{R}^{N \times M}$ the observation matrix.

$$\begin{aligned}
\boldsymbol{B}(hrf * Z)_t &= \sum_{i=1}^{M} b_{ij} \sum_{\tau=1}^{T} h_{t-\tau} Z_{j,\tau} && (5.3)\\
&= \sum_{i=1}^{M} \sum_{\tau=1}^{T} h_{t-\tau} b_{ij} Z_{j,\tau} && (5.4)\\
&= \sum_{\tau=1}^{T} h_{t-\tau} \sum_{i=1}^{M} b_{ij} Z_{j,\tau} && (5.5)\\
&= (hrf * (\boldsymbol{B} \cdot Z))_t && (5.6)
\end{aligned}$$

Furthermore, we can pull the nuisance variables r_t through the convolution using linearity (see Proposition 3.9). Let r_t be the nuisance variables and $r_{\text{deconv}_t} = \text{Conv}^{-1}(r_t, hrf)$ the deconvoluted nuisance variables.

$$\begin{aligned}
O(z_t, r_t) &= \boldsymbol{B}\left((hrf * z)_t\right) + \boldsymbol{J} r_t && (5.7)\\
&= (hrf * (\boldsymbol{B} \cdot z))_t + \boldsymbol{J} r_t && (5.8)\\
&= \left(hrf * (\boldsymbol{B} \cdot z + \boldsymbol{J} \cdot \text{Conv}^{-1}(r_t, hrf)\right)_t && (5.9)
\end{aligned}$$

Equation 5.9 now has an inversion, which is computationally much more efficient:

$$\{d_t\} = O^{-1}(\{x_t, r_t\}) = \{\boldsymbol{B}^{+}\left(\text{Conv}^{-1}(x_t, hrf) - \boldsymbol{J} \cdot \text{Conv}^{-1}(r_t, hrf)\right)\} \tag{5.10}$$

This form lets us compute the deconvoluted time series $\text{Conv}^{-1}(x_t, hrf)$ and $\text{Conv}^{-1}(x_t, hrf)$ once before training. At train time only the psuedo-inverse $\boldsymbol{B}^{+}$ has to be computed and applied as in inversion TF (see Eq. 2.30).

To compute the deconvolution, first the VISUSHRINK algorithm (see Algorithm 2) is applied to the time series $\{x_t\}$ to obtain an estimate of the noise level $\tilde{\sigma}$ and the denoised signal $\tilde{x_i}$. Then the power spectra of the measured signal, the denoised signal, the noise and the hrf are calculated and used in the Wiener deconvolution filter (see Eq. 3.26). Note that the noise is assumed to be Gaussian white noise, hence the power spectrum is flat $P_i = \tilde{\sigma}^2$.

The Wiener filter is then applied to the observed signal in Fourier space and the result transformed back to the time domain, resulting in the deconvoluted time series $\{x_t\}_{deconv}$.

5.1.1 Boundary issues and minimum noise level

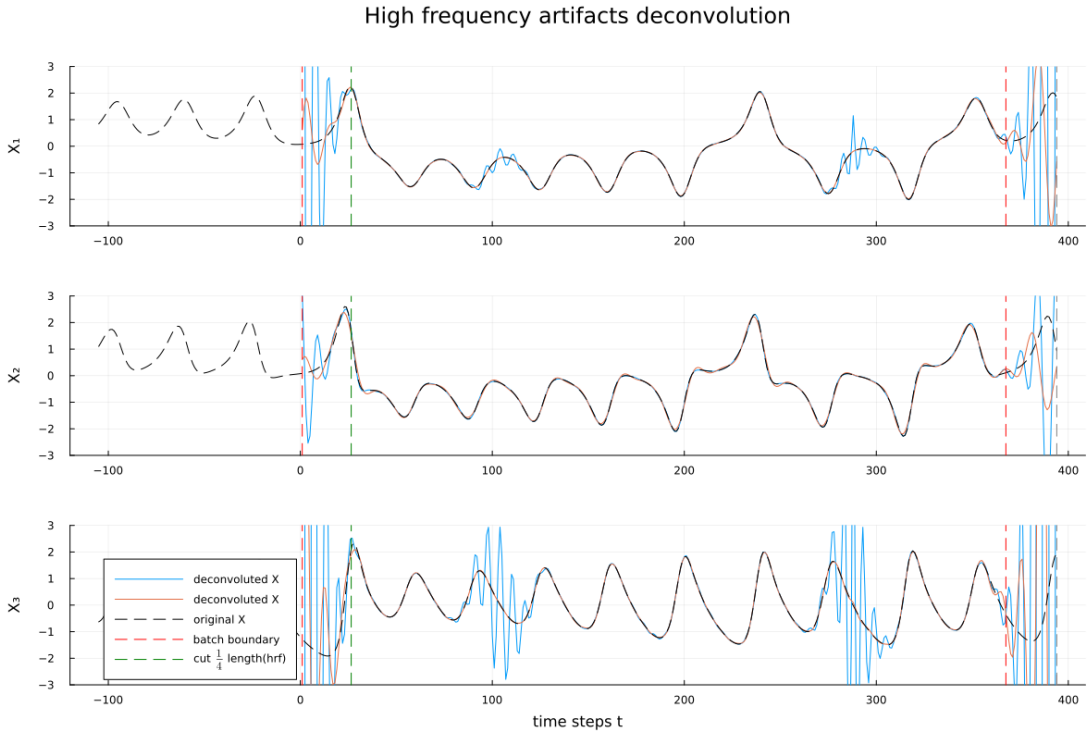
There are issues at the boundaries of a signal batch when calculating the deconvolution there. A batch refers to a subset of the dataset. Instead of feeding the entire dataset into the model at once in training, the time series data is divided into several batches, and the model is trained on each batch one at a time.

The reason is that values from outside the batch were used to calculate the values at the edges. The convolution effectively lets information smear out across time. We do however not have access to values from outside the batch and thus there are typically large errors at the edges of the deconvoluted signal.

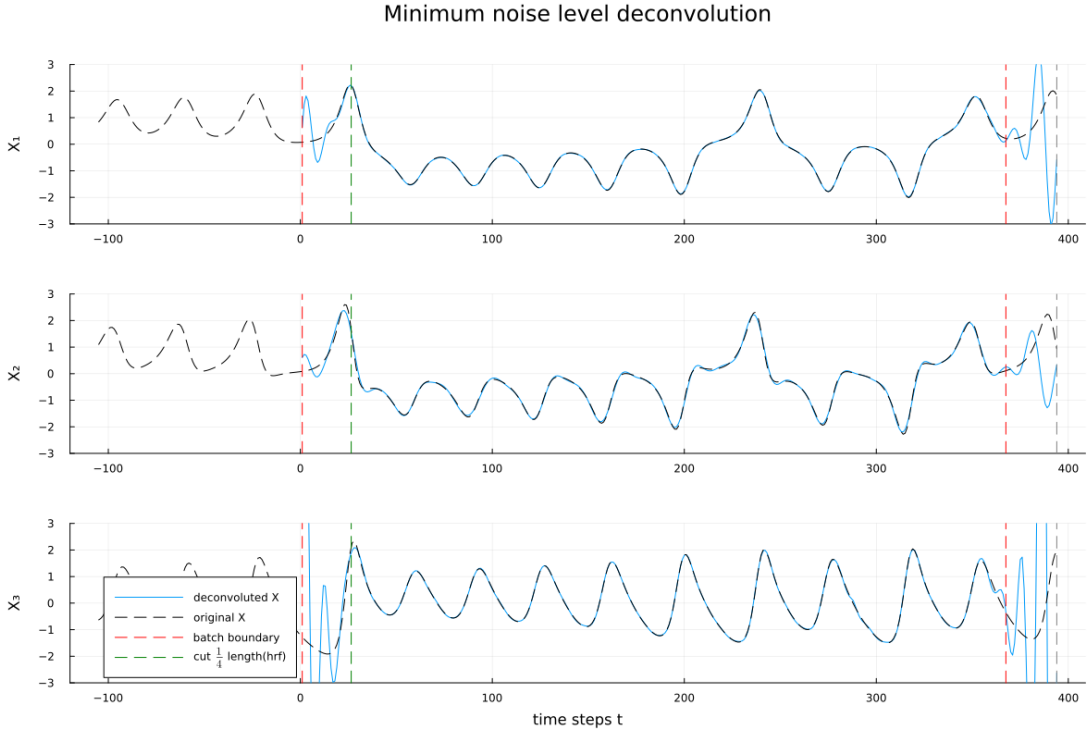
Ideally we would cut off the boundaries and only use valid points, i.e. ignore the first and last length(hrf) points of each batch. In the context of fMRI we are however already heavily data constrained with time series of length $\mathcal{O}(10^3)$. In relation to this the length of the hrf impulse is not negligible. For example, we have length($hrf_{2.0}$) = 17, length($hrf_{1.4}$) = 27 and length($hrf_{0.8}$) = 41. We can't afford to drop that many data points *in each batch*.

From visual inspection of the deconvolution results, I decided to cut $\frac{1}{4}$length(hrf) at the boundaries, as the biggest error occur there. The hyperparameters **cut_l** and **cut_r** set how much of the left and right boundary is cut. The cutoffs of a batch are visualized in the figures 5.1a and 5.1b.

In practice, the noise estimation only has a finite precision. For very small noise levels, for example in strongly preprocessed data or even no noise in artificial benchmark data, the deconvolution described above encounters numerical issues. The Wiener filter begins to return high frequency artifacts if we let the noise level approach 0 (see figure 5.1a). In most experimental cases this should not occur, but noiseless benchmark data and heavily smoothed data cause the noise estimation to return values close to 0. A hyperparameter **min_conv_noise** is set which is the minimum $\tilde{\sigma}$ the Wiener filter uses to avoid high frequency artifacts. Figures 5.1a and 5.1b offer a comparison between the simple Wiener filter and one with added minimum noise level. The higher frequency oscillations disappear, only the boundaries still contain large deviations from the original signal.



(A) Deconvolution of a Lorenz63 time series with no noise component. Deconvolution returns high frequency artifacts both at the boundaries and throughout the signal.



(B) Deconvolution of a Lorenz63 time series with no noise component, but minimum noise level set. High frequency artifacts are much smaller and only appear close to the boundaries within the $\frac{1}{4}\text{length}(hrf)$ cutoffs.

FIGURE 5.1: Deconvolution issues exemplified by a Lorenz time series

A full description of the deconvolution algorithm which was finally implemented and used for the experiments is given in Algorithm 3.

Algorithm 3 The full deconvolution Algorithm

Input: time series data x_i of length N; analyzing wavelet ψ;
minimum noise level $\tilde{\sigma}_{min}$; boundary cutoffs cut_l, cut_r;

1. Apply the VISUSHRINK algorithm (see Algorithm 2) to the time series $\{x_t\}$ to obtain an estimate of the noise level $\tilde{\sigma}$ and the denoised signal $\tilde{x_i}$
2. **If** $\tilde{\sigma} < \tilde{\sigma}_{min}$ **then** Set $\tilde{\sigma} := \tilde{\sigma}_{min}$
3. Compute the Fourier transform of the hrf and the power spectra of the observed signal, the denoised signal and the noise. The noise is assumed to be Gaussian white noise, hence the power spectrum is flat $P_i = \tilde{\sigma}^2$.
4. Compute the Wiener filter (see Eq. 3.26). The power spectrum of the denoised signal is used as input $S(k)$, the power spectrum of the original signal in Eq. 3.26.
5. Apply the Wiener deconvolution filter (see Eq. 3.27) to measured signal in Fourier space
6. Transform the result back to the time domain to obtain $\{x_t\}_{deconv}$.
7. Edges of signal are set to NaN as they contain artifacts and can't be used. $x\,[\text{begin} : \text{begin} + \text{cut\_l}] = \text{NaN}$, $x\,[\text{end} - \text{cut\_r} : \text{end}] = \text{NaN}$

Output: estimate of the deconvoluted signal $\{x_t\}_{deconv}$;

5.2 Evaluation metrics

There is a major problem when evaluating the performance of models on chaotic time series. It can easily be seen when remembering the concept of Lyapunov exponents (c.f. 1.2). If the maximum Lyapunov exponent of a system is larger than 0, $\lambda_{\max} > 0$, which is a necessary condition of chaos, nearby trajectories diverge exponentially quickly. This limits the applicability of n-step ahead prediction errors, as even small numerical errors will lead to exponentially growing prediction errors.

We therefore need different tools to evaluate the performance of models trained on chaotic time series which depend on properties invariant to the initial conditions. In this work, I will use two measures that have been established ([23]) to robustly evaluate the performance on chaotic data. The first is a state space measure that focuses on geometric properties of the time series and the other is based on the power spectrum.

5.2.1 State Space Distance D_{stsp}

The general idea of this measure was introduced in [20]. Given an observed N-dimensional time series $\{x_t\}$ of length T and a time series $\{\hat{x}_t\}$ with the same dimension/length generated by a model aiming to reconstruct the dynamical system underlying $\{x_t\}$, D_{stsp} measures the geometrical overlap of orbits in state space.

For low dimensional systems $N \leq 6$ the state space is separated into k^N bins where k is the number of bins per dimension. Each bin is given a label i and the number of points n_i of the time series is within that bin is measured. This gives us the relative frequency of that bin

$$p_i = \frac{n_i}{T}. \tag{5.11}$$

Combining these frequencies across all bins in space results in a probability distribution which approximates the state space distribution of the underlying dynamical system. Then the Kullback-Leibler-Divergence (KLD) of the empirical distributions can be calculated to give a measure of the overlap of both systems in state space:

$$D_{stsp} = \sum_{i=1}^{k^N} p_i \log\left(\frac{p_i}{q_i}\right) \tag{5.12}$$

where p_i are the relative frequencies of the measured time series and q_i are the relative frequencies of the generated time series. Note that KLD is not a metric in the mathematical sense, but instead it is a divergence which establishes the separation of probability distributions.

The complexity of this binning approach scales exponentially with the observation dimension N and thus becomes intractable with growing N. To compute D_{stsp} in higher dimensional systems, [22] use a Gaussian mixture model with centers (means) x_t and diagonal covariance $\mathbf{\Sigma} = \text{diag}(\sigma^2, \cdots, \sigma^2)$ where σ is a hyperparameter. The GMMs of a time series are given by

$$f(y) = \frac{1}{T}\sum_{t=1}^{T} \mathcal{N}(y; x_t, \mathbf{\Sigma}). \tag{5.13}$$

Following [40], the KLD of the two GMMS can be computed using a Monte-Carlo sampling approach

$$D_{stsp} \approx \frac{1}{K}\sum_{i=1}^{K}\log\left(\frac{f_{obs}(y_i)}{f_{gen}(y_i)}\right) = \frac{1}{K}\sum_{i=1}^{K}\log\left(\frac{1/T\sum_{t=1}^{T}\mathcal{N}\left(y_i; x_t; \mathbf{\Sigma}\right)}{1/T\sum_{t=1}^{T}\mathcal{N}\left(y_i; \hat{x}_t; \mathbf{\Sigma}\right)}\right) \tag{5.14}$$

where K is the number of samples drawn, f_{obs} is the distribution of the observed time series and f_{gen} is the distribution of the generated time series. This approach can be interpreted as an adaptive binning approach only considering bins with points in them. While the two measures correlate strongly in low dimensions, it is difficult to validate the GMM approach in high dimensional systems.

5.2.2 Power Spectrum Error D_{PSE}

The previously introduced state space measures discards all temporal structure of the data. To include temporal information in the model evaluation I will also use the power spectra of the observed and generated time series. For each dimension $i \in [\![0, N]\!]$ the scalar time series $\{x_t^{(i)}\}$ can be used to calculate a power spectrum density (PSD) $\{S_k^{(i)}\}$. The components are calculated from the Fourier transform of $\{x_t^{(i)}\}$.

$$S_k^{(i)} = \frac{|\widehat{x}_k^{(i)}|}{T} = \frac{|\mathscr{F}\{x_t^{(i)}\}_k|}{T} \tag{5.15}$$

To evaluate the agreement of power spectra the Hellinger Distance (HD) is used as suggested by [23]. Since HD is a probability measure the computed power spectra have to be normalized:

$$\bar{S}_k^{(i)} = \frac{S_k^{(i)}}{\sum_{j=1}^{T} S_j^{(i)}} \tag{5.16}$$

The Hellinger Distance is then calculated as follows using the power spectra of the observed time series $p_k^{(i)} = \bar{S}_k^{(i)}(\{x_t^{(i)}\})$ and the generated time series $q_k^{(i)} = \bar{S}_k^{(i)}(\{\hat{x}_t^{(i)}\})$

$$D_H^{(i)} = \sqrt{1 - \sum_{k=1}^{T}\sqrt{p_i^{(i)} q_i^{(i)}}} \tag{5.17}$$

The power spectrum error is computed by averaging the Hellinger Distances across all N dimensions of the observed system.

$$D_{PSE} = \frac{1}{N}\sum_{i=1}^{N} D_H^{(i)} \tag{5.18}$$

Before any computations the power spectra typically have to be smoothed with a Gaussian kernel with width σ to reduce the noise. This is another hyperparameter that has to be chosen to evaluate a model.

5.3 Datasets

The methods developed in this thesis are applied to both synthetic benchmark datasets and real world datasets. As benchmark system I used the Lorenz63 system which was convoluted different hrf functions and obfuscated with different noise levels. The advantage of such a benchmark system is that the latent, (unconvoluted and noiseless) trajectories of the system are known. This allows us to evaluate the model outputs both in the latent space and the observation or measurement space. As real fMRI data I used the publicly available dataset by the MPI Leipzig ([41]), which contains task-free resting-state fMRI (rs-fMRI) of 227 healthy participants together with range of physiological measurements, cognitive tests and emotion/personality tests.

5.3.1 Lorenz63 System

The Lorenz63 introduced in [42] was designed to describe atmospheric convection based on three observables. A two-dimensional fluid layer is uniformly warmed from below and cooled from above. It is a continuous dynamical system given by the following set of differential equations:

$$\frac{dx_1}{dt} = \sigma(x_2 - x_1) \tag{5.19}$$

$$\frac{dx_2}{dt} = x_1(\rho - x_3) - x_2 \tag{5.20}$$

$$\frac{dx_3}{dt} = x_1 x_2 - \beta x_3 \tag{5.21}$$

where x_1 is proportional to the rate of convection, x_2 to the horizontal temperature variation and x_3 to the vertical temperature variation. The constants are α, ρ and β are system parameters proportional to the Prandt number, Rayleigh number and certain physical dimensions of the fluid-layer. To produce chaoticity the following parameters are typically used: $\sigma = 10$, $\rho = 28$ and $\beta = \frac{8}{3}$. These settings produce the so-called "butterfly attractor" characteristic of the Lorenz system. Example trajectories are shown in Figure 5.2.

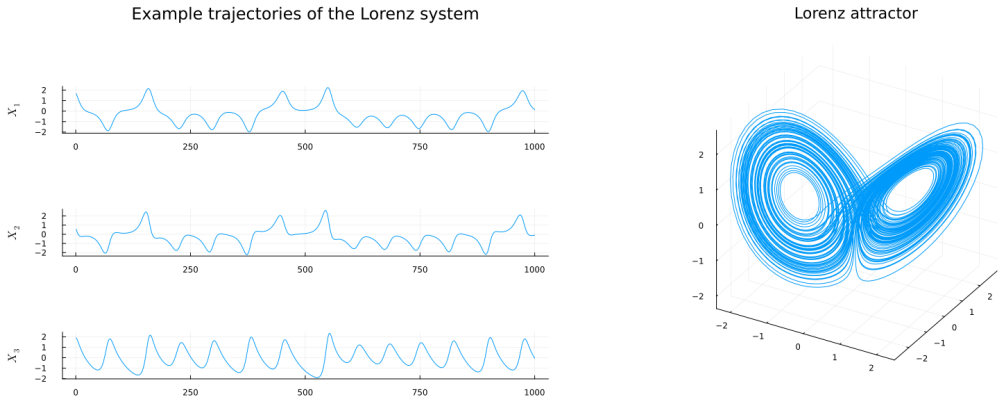


FIGURE 5.2: **Simulated data of the Lorenz System:** Example trajectories with 1000 time steps. Lorenz attractor with characteristic butterfly wing shape in 3D state space

5.3.2 LEMON Dataset

The LEMON ("Leipzig Study for Mind-Body-Emotion Interactions") dataset is a publicly available dataset published by the MPI Leipzig [41]. It consists of 227 healthy participants divided into a young ($N = 153$) and an elderly group ($N = 74$). Each of the participants completed a battery of tests for physiological assessment, including 6 cognitive tests and 18 emotion- and personality-related questionnaires, a 62-channel EEG experiment at rest and a task-free resting-state fMRI scan while further physiological measures were continuously acquired (blood-pressure, heart-rate, pulse, respiration). The full details of the dataset and the data acquisition can be found in [41].

In this work I will solely focus on the rs-fMRI data and not use the other data modalities for training the models. Utilizing the preprocessed fMRI data in the dataset, time series from 16 regions of interest (ROIs) were extracted. These time series were subsequently smoothed, band-pass filtered and standardized to have a mean of 0 and a standard deviation of 1. This routine was copied from [20] which provides more details on the preprocessing. Example trajectories are shown in figure 5.3.

FIGURE 5.3: First 8 dimensions of a resting state fMRI time series from the LEMON dataset

5.4 Experimental Setup: Benchmarking on Lorenz63 data

To benchmark the BOLD inversion algorithm and provide a proof of concept I used the following two setups:

5.4.1 Benchmarking on data without noise

1. I simulated the Lorenz63 system in the chaotic regime to create a dataset containing 10^5 time steps.

2. I trained a shallowPLRNN model with 5000 epochs on this dataset using weak identity teacher forcing and the identity mapping as observation model.

3. Using the model training in the previous step I created 4 datasets, each containing 100 trajectories of length $1.1 \cdot 10^6$ drawn from the model. The initial conditions were slightly perturbed with Gaussian white noise ($\mu = 0, \sigma = 1$) and the first 10^5 time steps discarded to ensure the model has reached its limiting set.

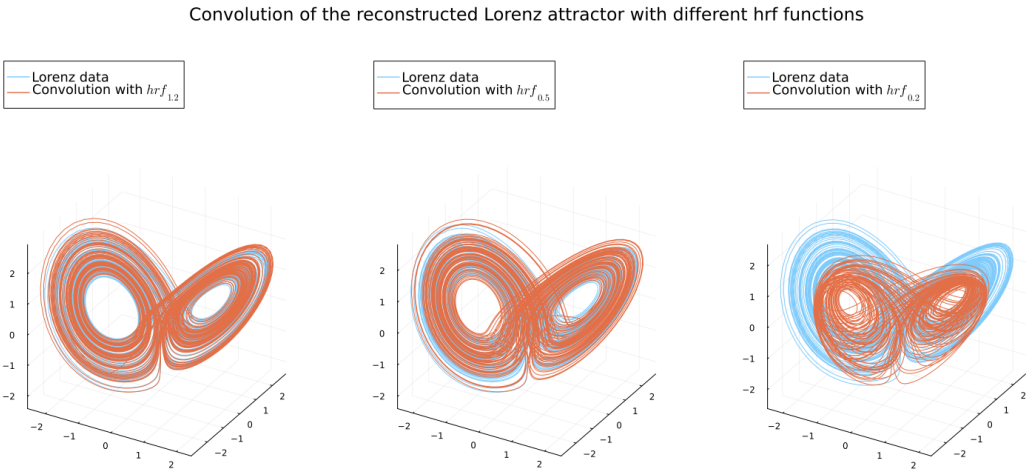


FIGURE 5.4: Convolution of the reconstructed Lorenz attractor with different hrf-functions

In the linear case I did not further modify the data. In the other cases I convoluted the data with an $hrf_{1.2}$, an $hrf_{0.5}$ and an $hrf_{0.2}$ respectively. These datasets were split 50:50% as training and test set for the benchmark.

4. To benchmark, I trained 100 models with the linear identity mapping on each of the datasets. Next, I trained 100 models with BOLD observation model on each dataset. For the impulse response I used the same hrf used to generate the dataset. In the case of the linear data without any convolution I used the $hrf_{1.2}$. Each model was trained for 1000 epochs.

5. After every 25 training epochs a trajectory is generated by the model and the state space distance D_{stsp} and the power spectrum error D_{PSE} are calculated so that we have access to the evolution of the Loss, D_{stsp} and D_{PSE} across training when evaluating the benchmark.

A visualization of these convoluted datasets is shown in figure 5.4. The complete hyperparameter settings of the training, such as the used optimizer, batch size etc. is given in the appendix, table A.1.

5.4.2 Benchmarking on data with noise

1. I simulated the Lorenz63 system in the chaotic regime to create a dataset containing 10^5 time steps

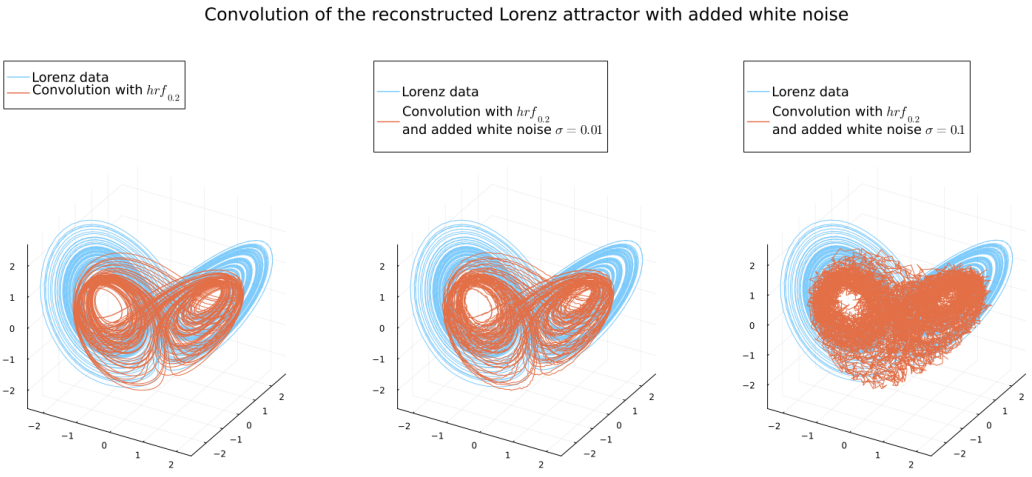


FIGURE 5.5: Example of noisy convoluted Lorenz attractor

2. I trained a shallowPLRNN model with 5000 epochs on this dataset using weak identity teacher forcing and the identity mapping as observation model.

3. Using the model trained in the previous step I created 8 datasets, each containing 100 trajectories of length $1.1 \cdot 10^5$ drawn from the model. The initial conditions were slightly perturbed with Gaussian white noise ($\mu = 0, \sigma = 1$) and the first 10^4 time steps discarded to ensure the model has reached its limiting set.

   In the linear case I added Gaussian white noise to two datasets, once with $\sigma = 0.01$ and once with $\sigma = 0.1$. For the others I convoluted two datasets each with an $hrf_{1.2}$, an $hrf_{0.5}$ and an $hrf_{0.2}$ respectively. Then I again added Gaussian white noise to the datasets, once with $\sigma = 0.01$ and once with $\sigma = 0.1$. These datasets were split 50:50% as training and validation set for the benchmark.

4. To benchmark, I trained 100 models with the BOLD observation model on each dataset with the respective impulse response used to create the dataset. I also trained with all three impulse responses, $hrf_{1.2}$, $hrf_{0.5}$ and $hrf_{0.2}$ on the unconvoluted datasets.

5. After every 25 training epochs a trajectory is generated by the model and the state space distance D_{stsp} and the power spectrum error D_{PSE} are calculated so that we have access to the evolution of the Loss, D_{stsp} and D_{PSE} across training when evaluating the benchmark. The generated trajectories have the same length as the trajectory from the test set, $5 \cdot 10^4$ in this case.

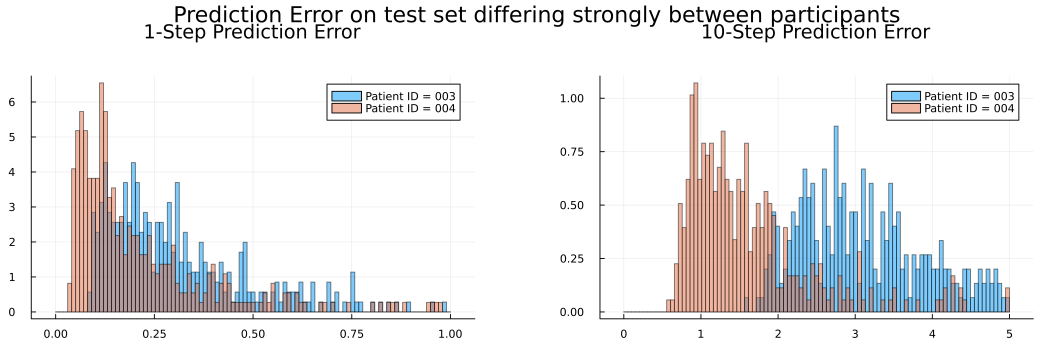


FIGURE 5.6: **Prediction Error on test set differing strongly between participants:** Distribution of prediction errors on two different participants ($N = 398$). The prediction capability of the models trained on participant 003 (*blue*) is significantly worse than those trained on participant 004 (*orange*). Set of hyperparameters for training did not differ between participants.

An example of these noisy trajectories is shown in figure 5.5. Again, the list of hyperparameters are given in table A.1.

5.5 Experimental Setup: Filtering LEMON datasets

When investigation the time series of the LEMON datasets I noticed the training results varied greatly between participants. Evaluating models trained with the same hyperparameters on different participants returned different very large differences on the validation set, which was chosen as the last 25% of the data points.

Following a suggestion by Alena Brändle I looked at the moving averages and moving variances of the time series to see how they evolve with time. Given a window size w, an N dimensional time series $\{x_t\}$ and a dimension $i \in [\![0, N]\!]$ the simple moving average (SMA) and variance (SMV) are calculated as follows

$$SMA_k^{(i)} = \frac{1}{w} \sum_{t=k}^{k+w} x_t^{(i)} \tag{5.22}$$

$$SMV_k^{(i)} = \frac{1}{w} \sum_{t=k}^{k+w} (x_t^{(i)} - SMA_k)^2 \tag{5.23}$$

Then we average over all dimensions:

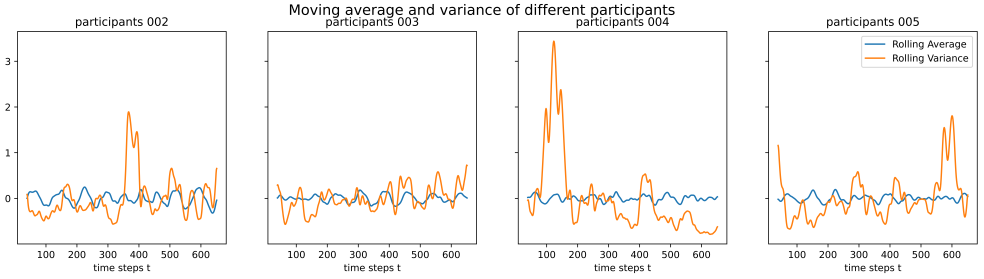


FIGURE 5.7: **Simple moving average and variance of different participants:** Example of SMA and SMV plotted for the first 4 participants in the LEMON dataset (participant 1 is not in the data). Window size $w = 40$, the mean of the SMV is subtracted to center it around 0 and improve presentation.

$$SMA_k = \frac{1}{N}\sum_{i=1}^{N} SMA_k^{(i)} \tag{5.24}$$

$$SMV_k = \frac{1}{N}\sum_{i=1}^{N} SMV_k^{(i)} \tag{5.25}$$

This results in two scalar time series that can easily be visualized and give us an intuition how the underlying fMRI time series evolves. This does however introduce a hyperparameter, the window size w. There is no simple criterion to decide on a window size. A large window will oversmooth the data while a small window will result many noise artifacts remaining. I chose a window size $w = 40$ are is delivered good results upon visual inspection. In figure 5.7 the SMA and SMV for different participants are shown. Clearly, the variance is not constant but varies greatly throughout time. Constant variance, however, is a necessary condition for stationarity.

To quantify the relationship between variance and time I calculated the correlation between them for each participant's SMV. The result is shown in the figure 5.8.

I chose to filter the LEMON dataset by only including the participants with a correlation of less than 0.16 (the 8 center bins around 0 in the histogram). This leaves 51 participants for the further experiments in this thesis. The time series are always split 75:25% into training and test set and the test set is always the last 25% of time points.

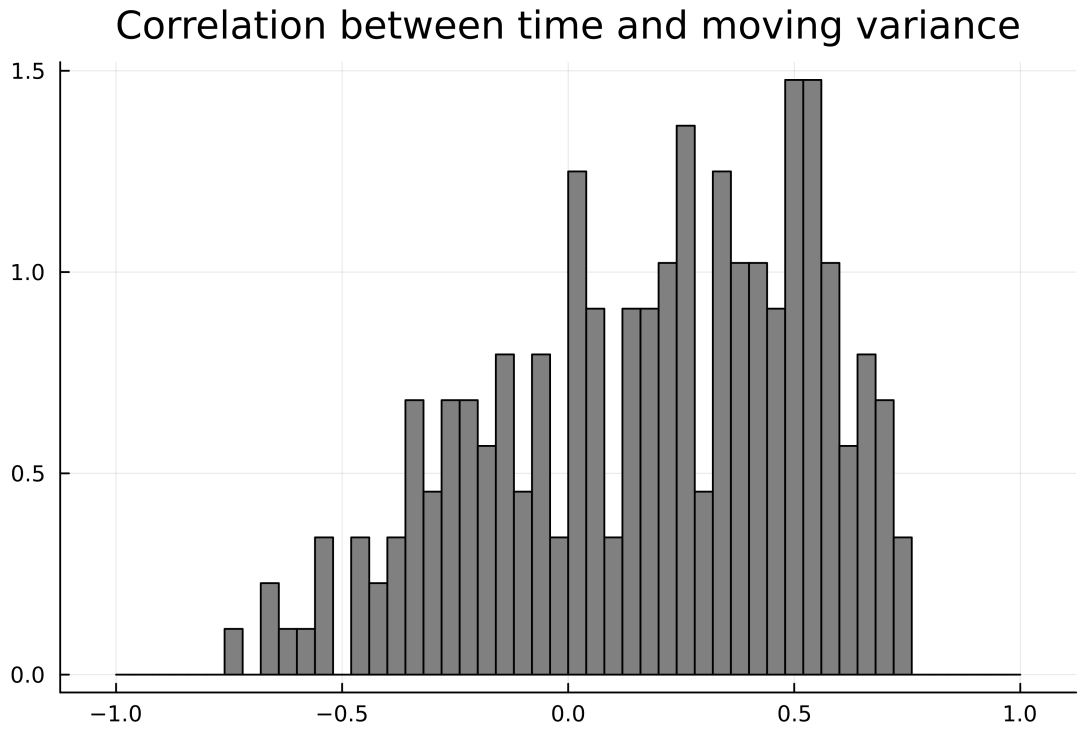


FIGURE 5.8: **Correlation between time and moving variance:** Normalized histogram of the Pearson correlation between time and moving variance of the time series in the LEMON dataset. Window size $w = 40$, bin size $b = 0.04$.

5.6 Evaluation Setup: Choosing hyperparameters for D_{stsp} and D_{PSE}

As mentioned in the introduction of the metrics D_{stsp} and D_{PSE} in sections 5.2.1 and 5.2.2, they each require a hyperparameter. In the case of D_{stsp} this is the scaling parameter of the GMM, in the case of D_{PSE} this is the smoothing σ Gaussian kernel used to smooth the power spectra. In [22] these parameters are set to 1.0 and 20 respectively. Unlike the datasets used in [22], the fMRI time series have the disadvantage of being rather short, $\mathcal{O}(10^3)$ time steps, which has implications for the choice of metric hyperparameters.

To illustrate this, I employed a model trained on the LEMON dataset to generate trajectories for two distinct time lengths: $T = 100$ and $T = 10^5$. In each case, I perturbed the initial conditions and applied a noise level of $\sigma = 0.07$ to these trajectories. Additionally, I generated purely random white noise with a standard deviation of $\sigma = 0.7$, approximately matching the standard deviation of the model-generated trajectories.

Subsequently, I computed the metric D_{stsp} for these diverse time series, using various scaling parameters. Specifically, I chose scaling factor values *scal* within the interval $[0.1, 2]$, using a step size of $\delta = 0.01$. For each value of *scal*, I generated 100 distinct

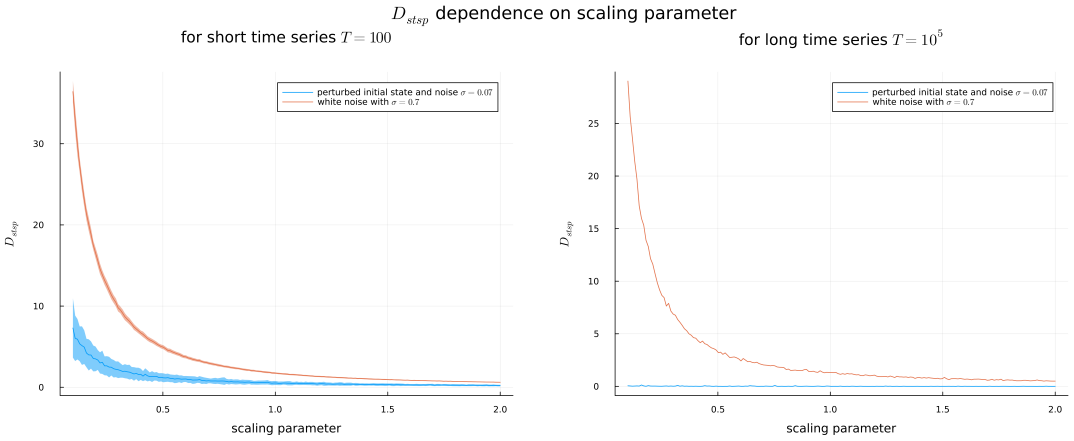


FIGURE 5.9: **D_{stsp} dependence on scaling parameter:** D_{stsp} values in dependence of the scaling parameter. Generating model was trained on a 16-dimensional rs-fMRI time series from the LEMON dataset.

sets of perturbed sample trajectories and white noise samples. I then recorded both the mean and standard deviation of D_{stsp} for each set. The comprehensive results are shown in Figure 5.9.

It is clear that D_{stsp} values vary much stronger in the short time series and for lower *scal* values. For higher *scal*-values it however becomes harder to differentiate between purely random noise and meaningful trajectories. For lower *scal*-values the variance of D_{stsp} is rises, but the white noise data is differentiated more clearly. In this evaluation I chose a value of $scal = 0.3$.

To pick a good σ which is used in the Gaussian kernel to smooth the power spectra I plotted the normalized smoothed power spectra of a long time series and a short time series in figure 5.10. It is clear that $\sigma = 20$, while a good choice in the case of long time series, is too large for short time series as it results in an over-smoothed spectrum. For the evaluation I chose the value $\sigma = 2$, as this value still preserves the main peaks on the left plot while still reducing the small fluctuations very effectively.

5.7 Grid search on LEMON dataset

To find good hyperparameters for training on fMRI datasets as opposed to the benchmark datasets, I decided to run a grid search for the weak teacher forcing α and the latent dimension ℓ. These two parameters are of special interest to us. The α-value determines the influence of the here developed BOLD inversion algorithm, which we want to test in this work. The latent dimension ℓ is of interest because we would like to know how many latent variables are needed to successfully reproduce the dynamics observed

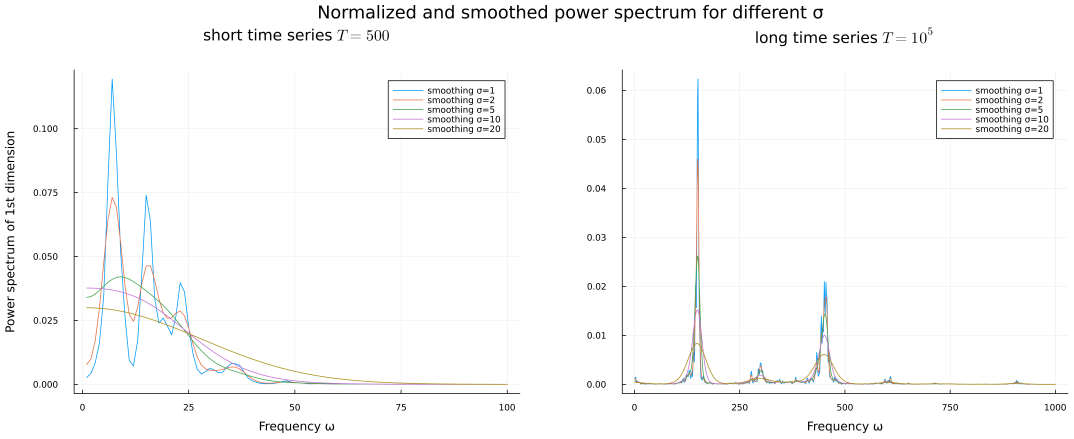


FIGURE 5.10: **Normalized and smoothed power spectra for different σ:** Power spectra of the first observation dimension for different smoothing σ values. Generating model was trained on a 16-dimensional rs-fMRI time series from the LEMON dataset.

in the fMRI trajectories and how many of the observed regions of interest (ROIs) contain redundant information.

The latent dimension ℓ was varied from 8 to 24 in steps of 4 (remember that we are working with 16 observables), the values for α were chosen from $\{0, 0.01, 0.03, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$ to compare no teacher forcing with α-values over two orders of magnitude ($\mathcal{O}(10^{-2})$ - $\mathcal{O}(10^{-1})$).

The grid search was run on the first 10 participants from the filtered LEMON dataset with 10 models being trained per participant and parameter configuration and train-test split of 75:25. The detailed hyperparameter settings are given in the appendix in table A.2.

Using results from this grid search, see section 6.2 for the evaluation, I trained on the entire (filtered) LEMON dataset. On each of the 51 participants 20 models were trained. The results are discussed in section 6.3.

Chapter 6

Results

6.1 Results on the benchmark systems

6.1.1 Noiseless data

These are the results for the trajectories generated from the noiseless Lorenz systems described in section 5.4.1. For the analysis, I chose the models trained for 1000 epochs. I treated a model run as not converged if the D_{stsp} is larger than 1. Those runs were not included in the quantitative analysis summarized in table 6.1. The distributions of the state space divergence and the power spectrum error are additionally displayed in figures 6.1 and 6.2. The non-convergent runs are grouped together at -0.2 on the histograms, thereby constraining the range of bins to enhance visualization.

The results overall are as intended. On the unconvoluted dataset, the linear observation model performs better while on the convolutional datasets the convolutional observation model outperforms. The performance disparity grows with decreased repeat time as a lower repeat time degrades the Lorenz attractor more heavily. It is noticeable for both models that the number of failed model runs increases with a stronger degradation of the observed trajectories. At $hrf_{0.2}$ the linear model doesn't converge reliably at all anymore (defined as $D_{stsp} < 1$). Reviewing the training in this case shows a lack of stability. There are individual epochs with good fits, but the model never converges there, but jumps back to a worse fit a few training epochs later. This is showcased qualitatively in figure 6.3.

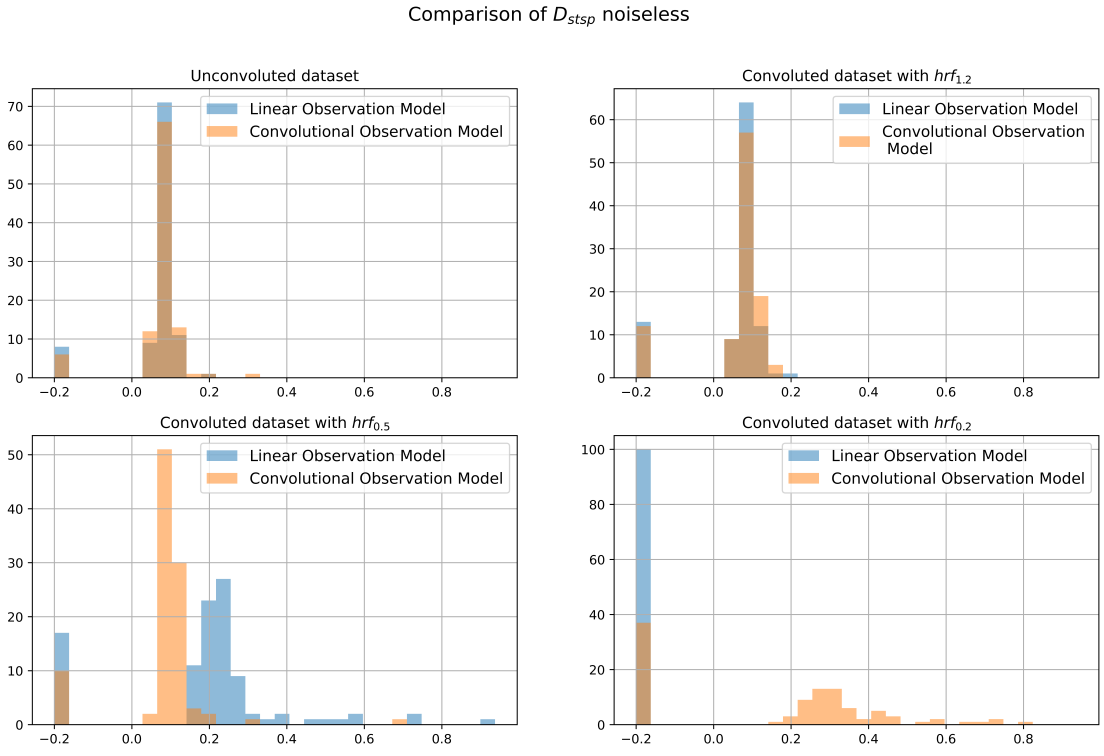


FIGURE 6.1: **Comparison of D_{stsp} noiseless:** Distributions of D_{stsp} of the convolutional observation model and the linear identity model. The bins at -0.2 are the model runs which did not converge, which is defined as models with $D_{stsp} > 1$.

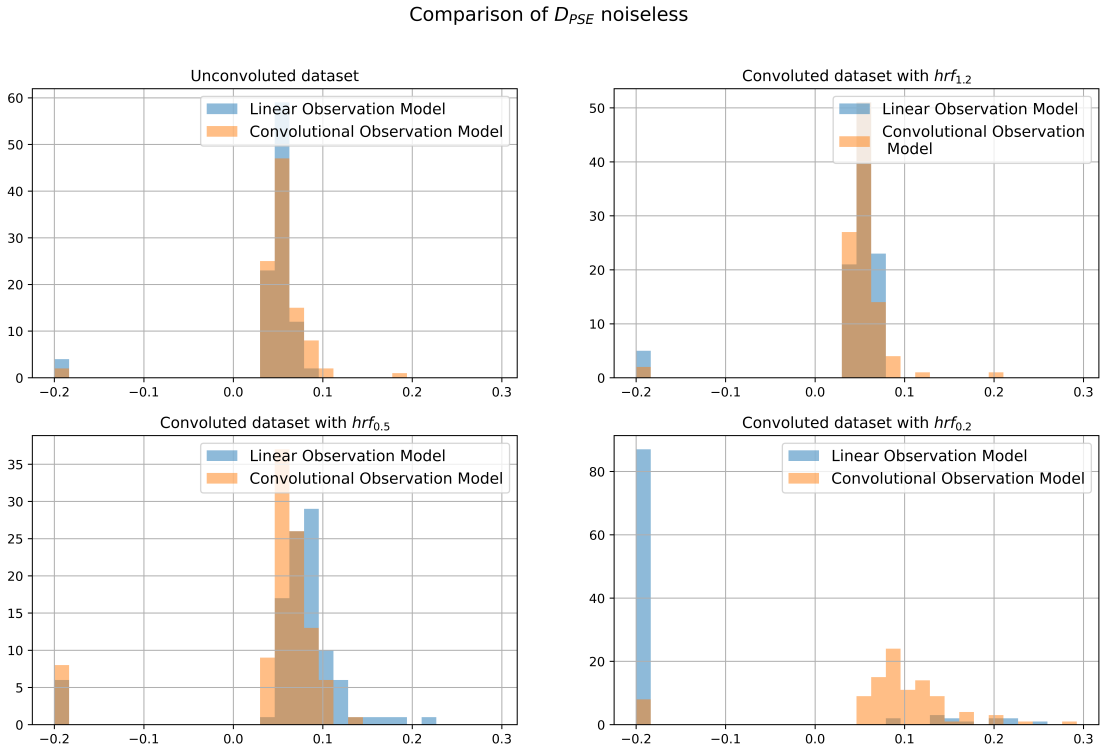
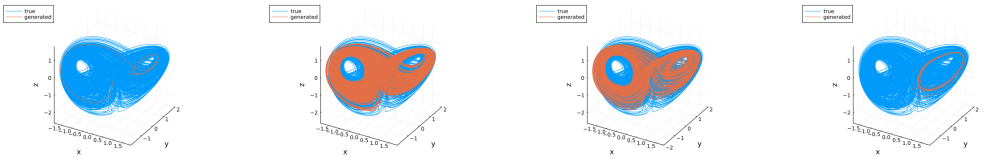


FIGURE 6.2: **Comparison of D_{PSE} noiseless:** Distributions of D_{PSE} of the convolutional observation model and the linear identity model. The bins at -0.2 are the model runs which did not converge, which is defined as models with $D_{stsp} > 1$.

TABLE 6.1: Quantitative comparison between linear and convolutional observation model

Dataset	Obs model	$N_{converged}$	20 step PE	D_{stsp}	D_{PSE}
Unconv	Lin	92	0.0004 ± 0.0001	0.09 ± 0.02	0.06 ± 0.01
Unconv	Conv	94	0.0004 ± 0.0001	0.09 ± 0.03	0.05 ± 0.01
$hrf_{1.2}$	Lin	87	0.0006 ± 0.0001	0.09 ± 0.02	0.05 ± 0.02
$hrf_{1.2}$	Conv	88	0.0007 ± 0.0001	0.09 ± 0.02	0.05 ± 0.01
$hrf_{0.5}$	Lin	83	0.0105 ± 0.0016	0.27 ± 0.14	0.08 ± 0.03
$hrf_{0.5}$	Conv	90	0.0009 ± 0.0001	0.11 ± 0.07	0.07 ± 0.02
$hrf_{0.2}$	Lin	0	$- \pm -$	$- \pm -$	$- \pm -$
$hrf_{0.2}$	Conv	63	0.0028 ± 0.0004	0.35 ± 0.14	0.11 ± 0.04



(A) At epoch 125 (B) At epoch 150 (C) At epoch 175 (D) At epoch 200

FIGURE 6.3: **Training instability in the presence of strong convolution:** Trajectories of the training data (a Lorenz attractor convoluted with an $hrf_{0.2}$) and of the model with linear observation equation at different training epochs.

6.1.2 Noisy data

These are the results for the trajectories generated from the noisy Lorenz systems described in section 5.4.2. For the analysis, I chose the models trained for 1000 epochs. I treated a model run as not converged if the D_{stsp} is larger than 1. Those runs were not included in the quantitative analysis summarized in table 6.2. The distributions of the metrics, state space divergence, power spectrum error and 20-step prediction error (PE), in the case of $hrf_{0.5}$ are additionally displayed in figure 6.4. The non-convergent runs are grouped together at -0.2 on the histograms, thereby constraining the range of bins to enhance visualization.

The results are analogous to the noiseless case. The stronger the degradation of the underlying trajectories by both noise and convolution, the better the convolutional observation model performs compared to the linear.

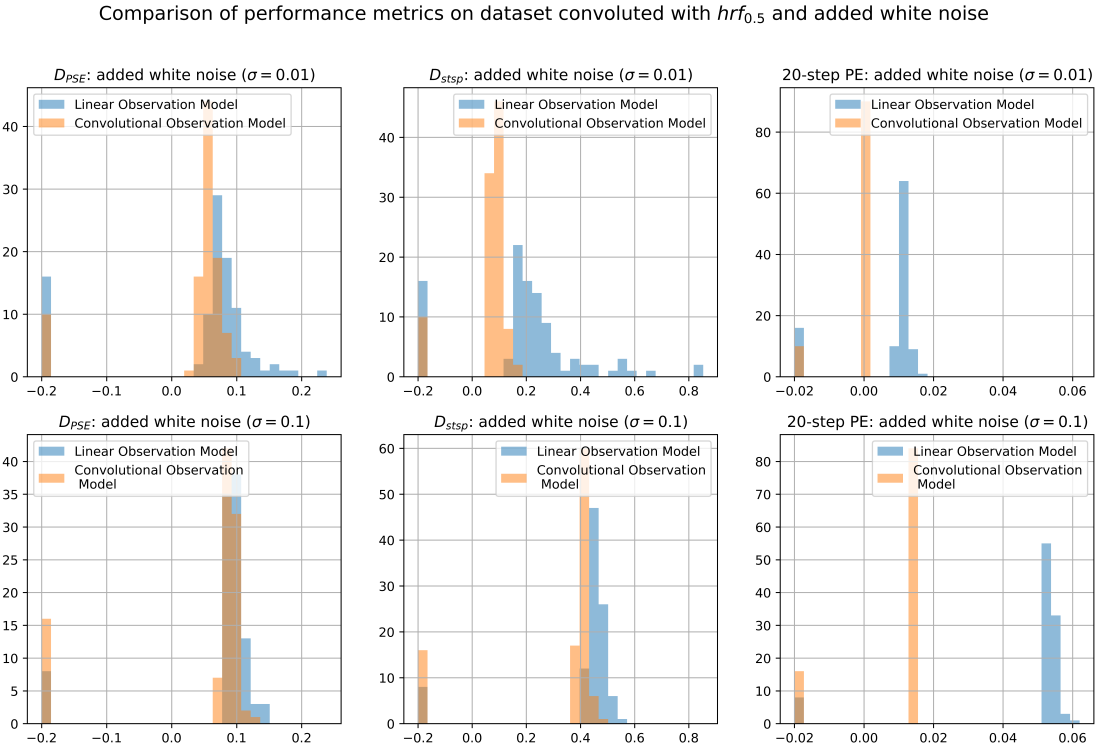


FIGURE 6.4: **Comparison of performance metrics on dataset convoluted with $hrf_{0.5}$ and added white noise:** Distributions of D_{PSE} and D_{stsp} of the convolutional observation model and the linear identity model. The bins at -0.2 are the model runs which did not converge, which is defined as models with $D_{stsp} > 1$.

TABLE 6.2: Quantitative comparison between linear and convolutional observation model on noisy data

Convolution	σ	Obs model	$N_{converged}$	20 step PE	D_{stsp}	D_{PSE}
$hrf_{1.2}$	0.01	Lin	78	0.0013 ± 0.0001	0.10 ± 0.10	0.05 ± 0.01
$hrf_{1.2}$	0.01	Conv	80	0.0011 ± 0.0001	0.09 ± 0.02	0.06 ± 0.01
$hrf_{0.5}$	0.01	Lin	84	0.0114 ± 0.0014	0.27 ± 0.14	0.09 ± 0.03
$hrf_{0.5}$	0.01	Conv	90	0.0012 ± 0.0001	0.09 ± 0.02	0.06 ± 0.01
$hrf_{0.2}$	0.01	Lin	1	0.0283 ± 0.0000	0.93 ± 0.00	0.19 ± 0.00
$hrf_{0.2}$	0.01	Conv	90	0.0023 ± 0.0001	0.22 ± 0.04	0.14 ± 0.02
$hrf_{1.2}$	0.1	Lin	86	0.0387 ± 0.0002	0.45 ± 0.02	0.09 ± 0.01
$hrf_{1.2}$	0.1	Conv	85	0.0158 ± 0.0001	0.44 ± 0.02	0.09 ± 0.01
$hrf_{0.5}$	0.1	Lin	92	0.0538 ± 0.0015	0.46 ± 0.03	0.10 ± 0.01
$hrf_{0.5}$	0.1	Conv	84	0.0145 ± 0.0001	0.41 ± 0.02	0.09 ± 0.01
$hrf_{0.2}$	0.1	Lin	2	0.0813 ± 0.0007	0.97 ± 0.01	0.18 ± 0.00
$hrf_{0.2}$	0.1	Conv	84	0.0191 ± 0.0002	0.70 ± 0.08	0.24 ± 0.02

6.2 Grid search on LEMON dataset

I compiled the results of the hyperparameter search in the following tables and figures. Figure 6.5 shows the mean and standard deviation for varying the latent dimension ℓ. The quantitative results are also displayed in table 6.3. The results show that the model performs similarly for all the different tested ℓ-values. The largest differences in performance are seen in the D_{stsp} values. The value $\ell = 16$ has by far the lowest variation in D_{stsp} across both train and test set. Hence, I chose $\ell = 16$ as the value for training on the entire dataset.

Figure 6.6 shows the mean and standard deviation for varying the teacher forcing α. The quantitative results are also displayed in table 6.4. The results are not very straightforward to interpret. D_{stsp}-value improve for very low (or even 0) α-values. D_{PSE} is higher at low α-values and is lowest $\alpha = 0.1$. The prediction errors are also the lowest around $\alpha = 0.1$, hence I chose that α-value for training on the entire dataset.

TABLE 6.3: Results of the parameter search for ℓ values. Model runs where excluded if the 1-step PE > 1

ℓ	$N_{converged}$	Dataset	1-step PE	5-step PE	10-step PE	D_{stsp}	D_{PSE}
8	247	Train	0.16 ± 0.14	0.77 ± 0.64	0.93 ± 0.53	5.90 ± 8.61	0.35 ± 0.11
8	247	Test	0.26 ± 0.17	1.44 ± 0.66	1.96 ± 0.54	9.85 ± 9.67	0.20 ± 0.08
12	244	Train	0.16 ± 0.15	0.75 ± 0.72	0.87 ± 1.14	3.87 ± 3.76	0.32 ± 0.11
12	244	Test	0.25 ± 0.18	1.46 ± 0.60	1.95 ± 1.02	7.86 ± 5.21	0.19 ± 0.07
16	226	Train	0.20 ± 0.20	0.77 ± 0.77	0.78 ± 0.54	3.08 ± 2.08	0.30 ± 0.10
16	226	Test	0.32 ± 0.25	1.53 ± 0.67	1.89 ± 0.51	6.86 ± 3.13	0.18 ± 0.08
20	245	Train	0.28 ± 0.17	1.36 ± 1.44	1.40 ± 1.48	4.49 ± 4.77	0.34 ± 0.12
20	245	Test	0.36 ± 0.22	1.88 ± 1.44	2.11 ± 1.50	8.71 ± 7.05	0.21 ± 0.08
24	241	Train	0.25 ± 0.15	1.17 ± 0.95	1.25 ± 0.69	4.47 ± 4.08	0.34 ± 0.13
24	241	Test	0.32 ± 0.19	1.66 ± 1.04	1.92 ± 0.61	8.95 ± 7.45	0.21 ± 0.10

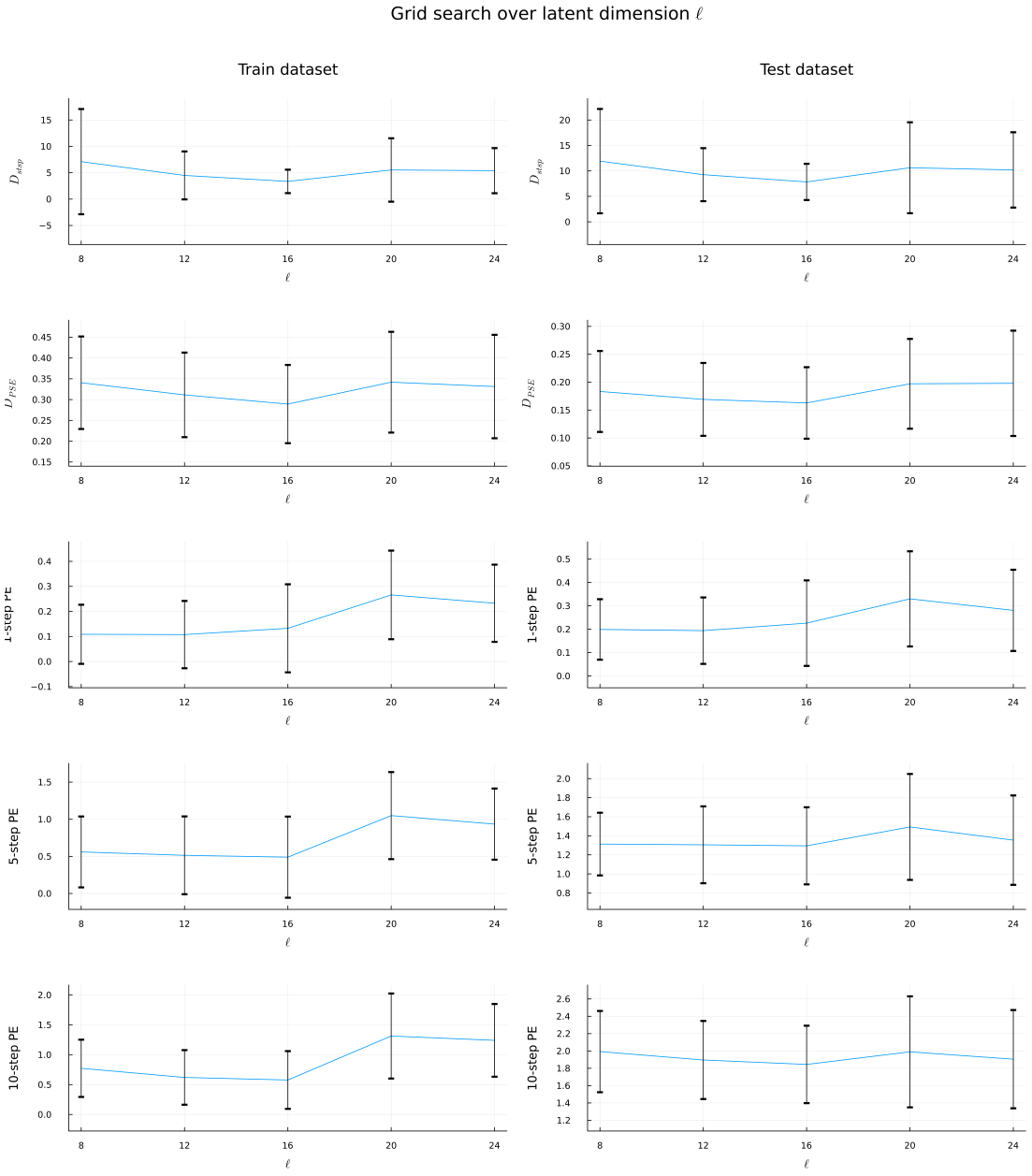


FIGURE 6.5: **Grid search over latent dimension ℓ:** Mean and standard deviation for the different metrics depending on the latent dimension ℓ. Model runs where excluded if the 1-step PE > 1

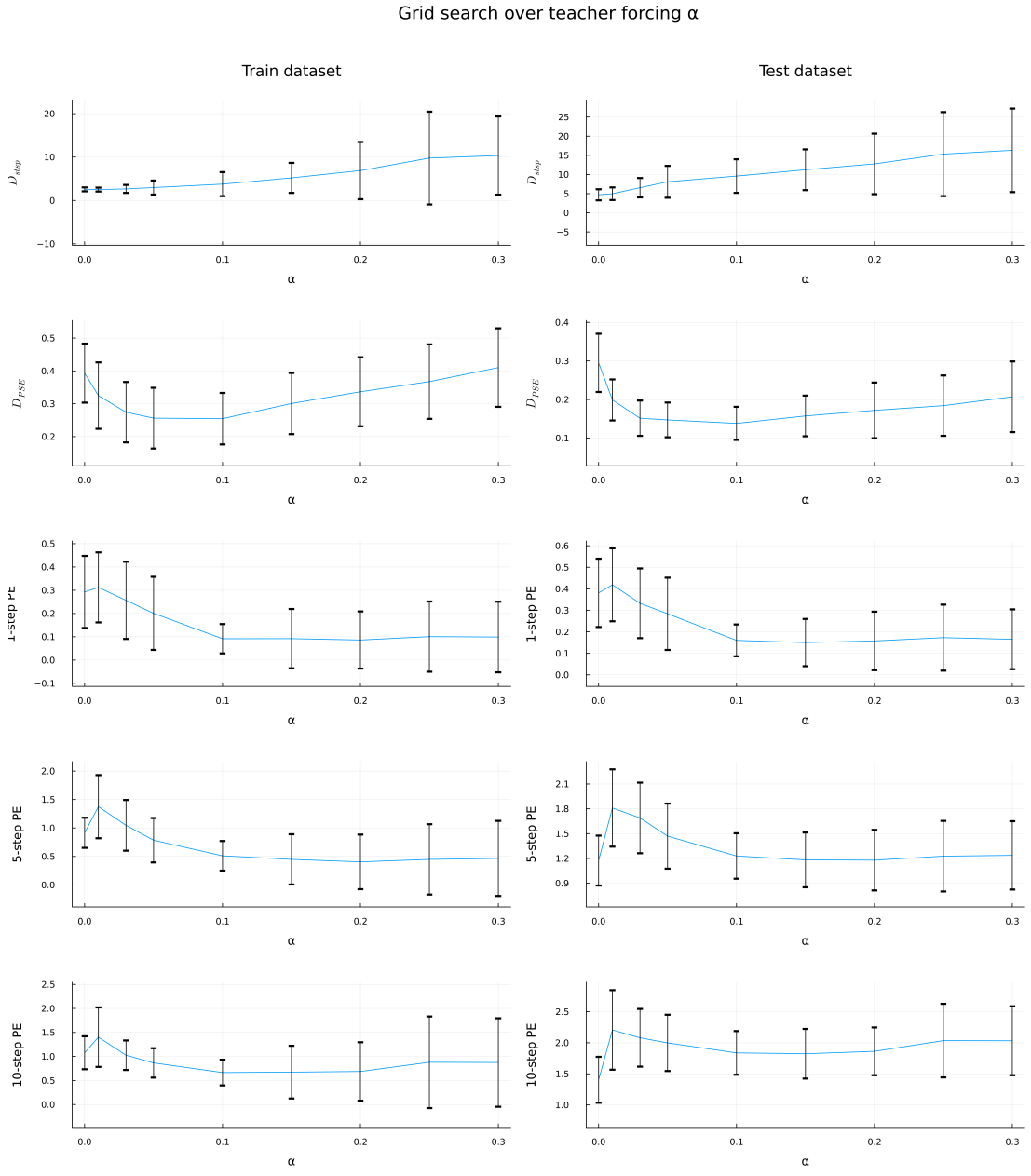


FIGURE 6.6: **Grid search over teacher forcing α:** Mean and standard deviation for the different metrics depending on the teacher forcing α. Model runs where excluded if the 1-step PE > 1

TABLE 6.4: Results of the parameter search for α values. Model runs where excluded if the 1-step PE > 1

α	$N_{converged}$	Dataset	1-step PE	5-step PE	10-step PE	D_{stsp}	D_{PSE}
0.00	221	Train	0.29 ± 0.15	1.05 ± 1.09	1.26 ± 1.76	2.55 ± 0.45	0.39 ± 0.09
0.00	221	Test	0.41 ± 0.24	1.31 ± 1.02	1.58 ± 1.71	4.70 ± 1.45	0.29 ± 0.08
0.01	247	Train	0.31 ± 0.15	1.63 ± 1.13	1.42 ± 0.70	2.50 ± 0.45	0.32 ± 0.10
0.01	247	Test	0.43 ± 0.19	2.26 ± 1.27	2.25 ± 0.77	4.98 ± 1.64	0.20 ± 0.05
0.03	244	Train	0.26 ± 0.17	1.13 ± 0.92	1.03 ± 0.31	2.65 ± 0.92	0.27 ± 0.09
0.03	244	Test	0.34 ± 0.17	1.81 ± 0.81	2.08 ± 0.46	6.55 ± 2.52	0.15 ± 0.05
0.05	247	Train	0.20 ± 0.16	0.84 ± 0.62	0.87 ± 0.31	2.96 ± 1.61	0.26 ± 0.09
0.05	247	Test	0.28 ± 0.17	1.57 ± 0.65	2.00 ± 0.45	8.09 ± 4.16	0.15 ± 0.05
0.10	250	Train	0.09 ± 0.06	0.51 ± 0.26	0.67 ± 0.27	3.76 ± 2.77	0.25 ± 0.08
0.10	250	Test	0.16 ± 0.07	1.25 ± 0.33	1.84 ± 0.35	9.59 ± 4.38	0.14 ± 0.04
0.15	246	Train	0.09 ± 0.13	0.51 ± 0.64	0.67 ± 0.55	5.20 ± 3.46	0.30 ± 0.09
0.15	246	Test	0.17 ± 0.18	1.28 ± 0.66	1.85 ± 0.50	11.23 ± 5.32	0.16 ± 0.05
0.20	246	Train	0.09 ± 0.12	0.44 ± 0.59	0.69 ± 0.61	6.88 ± 6.60	0.34 ± 0.11
0.20	246	Test	0.16 ± 0.15	1.24 ± 0.54	1.86 ± 0.38	12.76 ± 7.90	0.17 ± 0.07
0.25	243	Train	0.10 ± 0.15	0.50 ± 0.74	0.88 ± 0.95	9.77 ± 10.69	0.37 ± 0.11
0.25	243	Test	0.18 ± 0.19	1.33 ± 0.72	2.05 ± 0.64	15.32 ± 10.97	0.18 ± 0.08
0.30	241	Train	0.10 ± 0.15	0.52 ± 0.78	0.90 ± 0.98	10.35 ± 9.02	0.41 ± 0.12
0.30	241	Test	0.17 ± 0.17	1.33 ± 0.63	2.05 ± 0.60	16.31 ± 10.90	0.21 ± 0.09

6.3 Analysis of the results on the LEMON dataset

In this section I have compiled the results of training on the entire filtered (see section 5.5) LEMON dataset. For that I trained 20 models on each participant with the hyperparameters $\ell = 16$ and $\alpha = 0.1$, the other hyperparameters can be found in A.2.

The results of the training are shown in figure 6.7, which shows the distributions of the different metrics on both the training and the test datasets. Table 6.5 also summarizes these results in terms of mean and standard deviation of the different metrics.

Looking at the histograms in figure 6.7 we can see that results on the test set appear worse overall, having both higher mean and higher variation, across all metrics, with the notable exception of D_{PSE} where this relationship appears to be reversed. I employed

TABLE 6.5: Quantitative Results of the PLRNN with BOLD observation model on the LEMON dataset. Model runs where excluded if the 1-step PE > 1. $N_{converged} = 1007$

metric	Train Data	Test Data
D_{stsp}	2.84 ± 2.49	8.78 ± 3.46
D_{PSE}	0.21 ± 0.07	0.14 ± 0.04
1-step PE	0.06 ± 0.07	0.17 ± 0.33
5-step PE	0.30 ± 0.26	1.20 ± 0.41
10-step PE	0.41 ± 0.33	1.78 ± 0.38

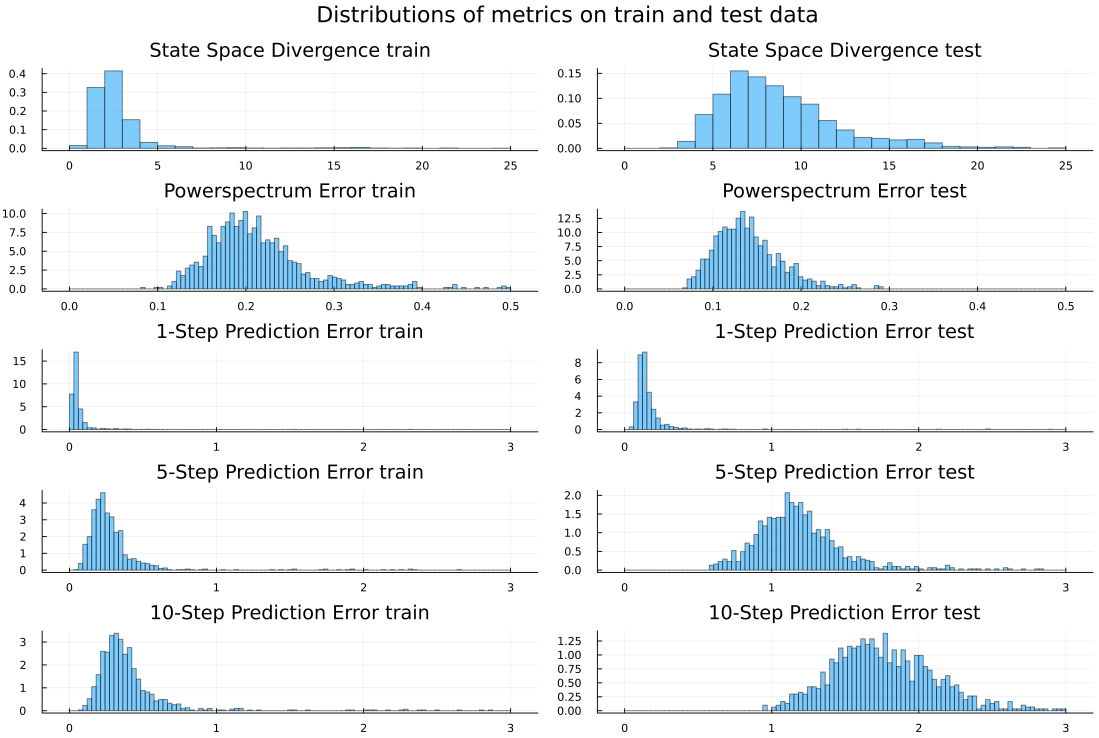


FIGURE 6.7: **Distributions of metrics on train and test data:** Histograms of the results of each model run on the train and test data. Binning of the predictions errors is identical for better visual comparison.

TABLE 6.6: Wilcoxon signed-rank test applied to the distribution of metrics on training and test set. Model runs where excluded if the 1-step PE > 1. $N_{converged} = 1007$

Metric	Result	p-Value	Rank sum	Z-statistic
D_{stsp}	Reject H_0. Distributions differ.	< 0.001	2490	-27.22
D_{PSE}	Reject H_0. Distributions differ.	< 0.001	20630	-25.25
1-step PE	Reject H_0. Distributions differ.	< 0.001	3578	-27.10
5-step PE	Reject H_0. Distributions differ.	< 0.001	164	-27.47
10-step PE	Reject H_0. Distributions differ.	< 0.001	99	-27.48

the Wilcoxon signed-rank test (method found in [43], used implementation by [44]) to establish the statistical significance of the disparities in metric distributions between the training and test sets. The test results are gathered in table 6.6 and as expected the distributions between train and test set all differ significantly.

To better illustrate the results I plotted an example reconstruction for all 16 dimensions in figure 6.8. As it is not easy to get a good overview over all 16 dimensions plotted next to each other, I used principal component analysis (PCA) to project the trajectories down to 3 dimensions, see figure 6.9. Unfortunately the results of the PCA are also not much more insightful and don't provide much intuition on the quality of the reconstruction.

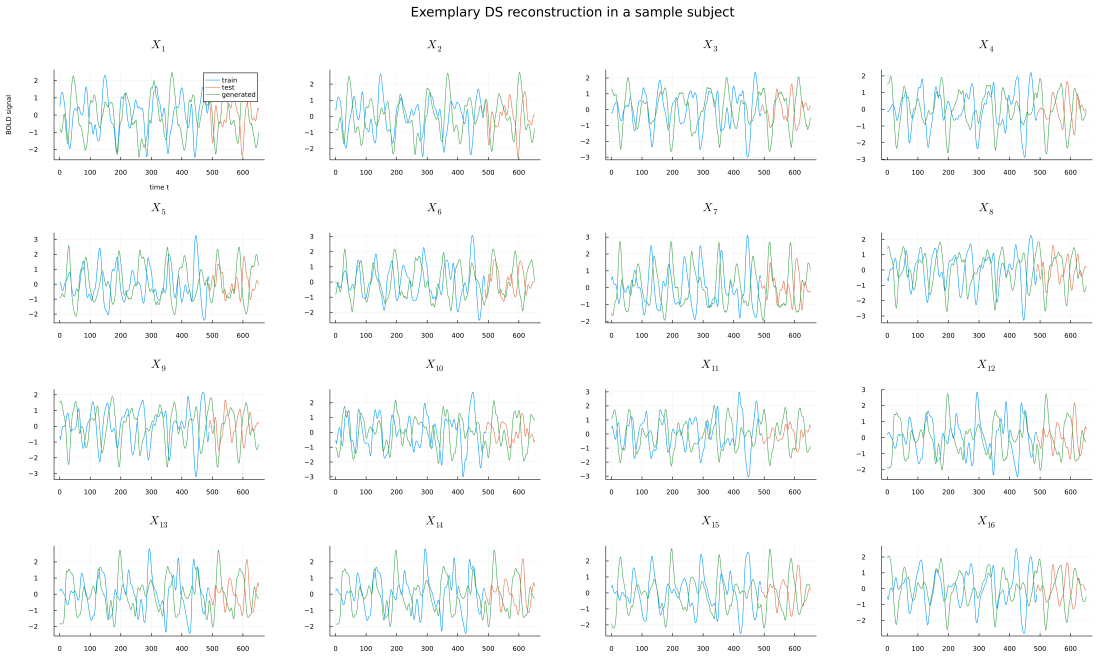


FIGURE 6.8: **Exemplary DS reconstruction in a sample subject:** Trajectory generated by a model (*green*) compared to a real trajectory. The model was trained on the *blue* data. The test data (*orange*) was not seen by the model in training.

When looking at different model runs it can be observed that a low D_{stsp} value and a low D_{PSE} often don't coincide. In figure 6.10 I plotted some examples of trajectories with different combinations of high/low D_{stsp} and D_{pse} values to examine what these differences mean for the reconstruction.

From looking at the different examples shown in figure 6.10 it is immediately apparent that a high D_{stsp} and D_{PSE} result in a bad model which isn't able to reconstruct the dynamics at all. In the case of a low D_{stsp} combined with a high D_{PSE} the model is missing the dominant frequency of the participant data, which can be seen in the power spectrum. Instead, there is a peak at 0, meaning the model learned a constant offset instead of the dominant frequency of the training trajectory. The reconstructions with a low D_{PSE} both look quite good as the dominant frequency is captured well and no artificial mode at 0 was learned. It is not apparent from the plot why D_{stsp} is so high or what part of state space the model is not capturing. But considering that D_{stsp} has a high variance in short trajectories, as noted in section 5.6, it seems that the power spectrum error D_{PSE} is the more important and expressive metric for fMRI data.

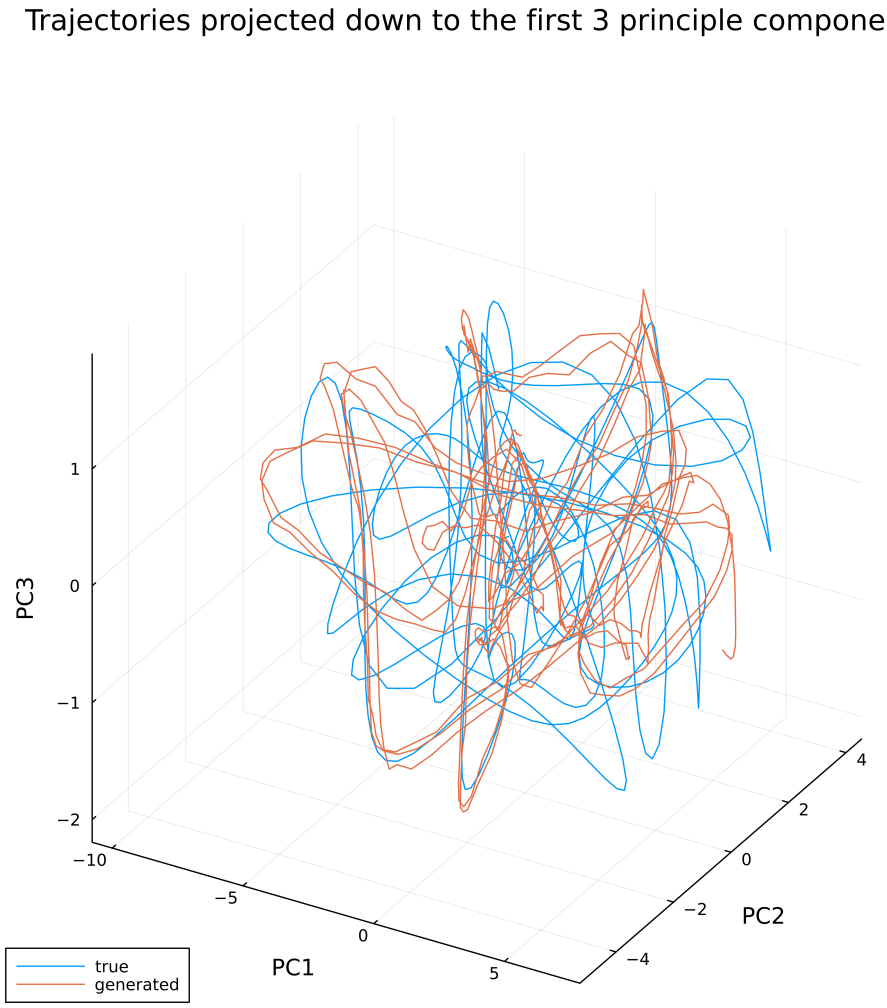


FIGURE 6.9: **Trajectories projected down to the first 3 principal components:** Trajectory generated by a model (*orange*) compared to a real trajectory (*blue*). The first 3 principal components of the true data are used.

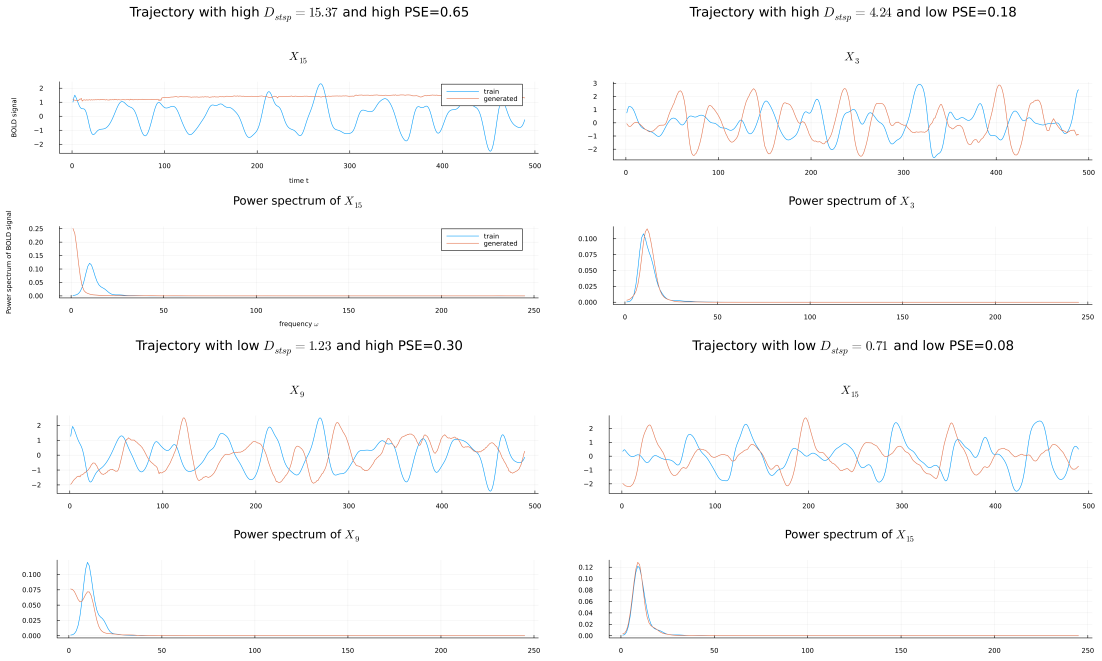


FIGURE 6.10: **Comparison of trajectories with different D_{stsp} and D_{PSE} combinations:** Trajectories and power spectra generated by models (*orange*) with different D_{stsp} and D_{PSE} values compared to the real trajectories (*blue*).

6.3.1 Further investigation of dynamical features

To further characterize the models, I calculated the Lyapunov spectrum (see section 1.2) of the trained PLRNNs. The Lyapunov spectrum of a general RNN can be calculated using the algorithm given in [4]. The mean and standard deviation of the maximum Lyapunov exponent λ_{max} over all runs for each participant is shown in figure 6.11.

Figure 6.11 clearly shows that most of the models learn a positive λ_{max} in training, i.e. chaotic dynamics. Only for a few participants, such as Nr. 282, are there several runs with an entirely negative Lyapunov spectrum. The models with negative Lyapunov spectrum typically performance issues if evolved freely over time. They either decay towards a constant value or they, at some point, fall into a simple oscillatory cycle. Examples of these behaviors are shown in figure 6.12 and figure 6.13. These findings suggest that λ_{max} needs to be taken into account when determining model quality.

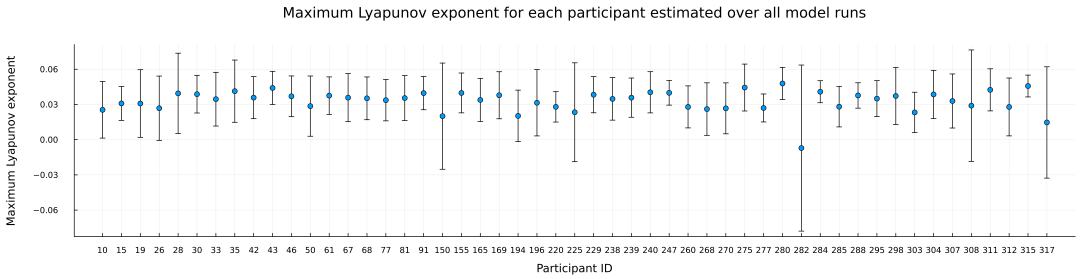


FIGURE 6.11: **Maximum Lyapunov exponents estimated over all model runs:** λ_{max} mean and standard deviation across model runs. Values of non-converged models were excluded.

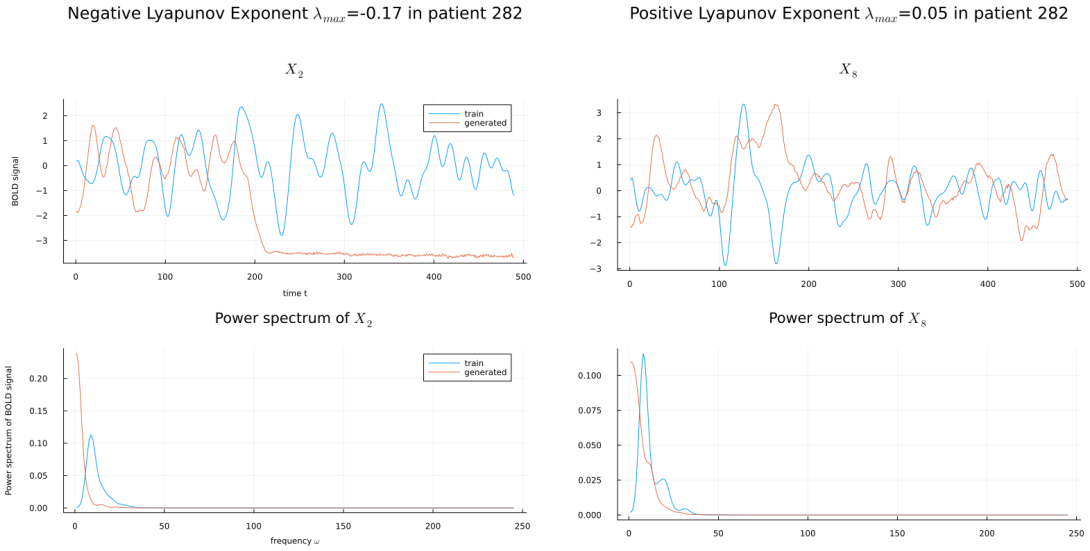


FIGURE 6.12: **Comparison between a model with positive and a model with negative λ_{max}:** The model with a negative λ_{max} simply decays towards a constant value after initially producing a trajectory similar to the participant data.

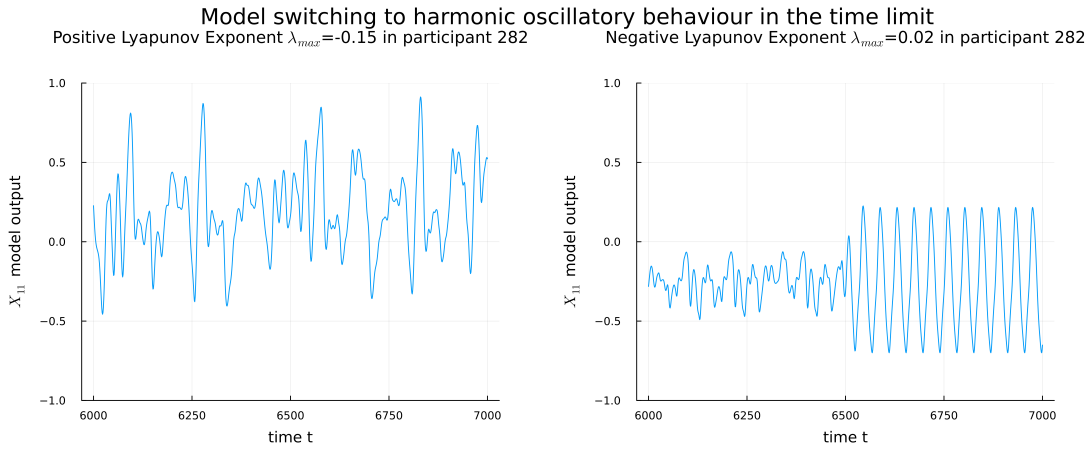


FIGURE 6.13: **Model switching to harmonic oscillatory behavior in the time limit:** The model with a negative λ_{max} simply switches to simple oscillations after evolving freely for enough time steps. Note that this time is not fixed to the model. If initialized with a random state it may almost immediately exhibit this behavior or only after many time steps as shown in the plots.

Similar to the maximum Lyapunov exponent I also examined the spectral norms of the W_1 and W_2 matrices of the trained models. The spectral norm of a matrix $\boldsymbol{A} \in \mathbb{R}^{M \times N}$ is induced by the L_2-norm and defined as

$$\|\boldsymbol{A}\|_2 = \max_{\|x\|_2=1} \|\boldsymbol{A}x\|_2 = \lambda_{max}(A^T A) \tag{6.1}$$

where $\lambda_{max}(A^T A)$ is the maximum eigenvalue of $A^T A$. The mean and standard deviation of the W matrices spectral norms is shown in figure 6.14. The matrix norms clearly vary much more between participants than the maximum Lyapunov exponent does.

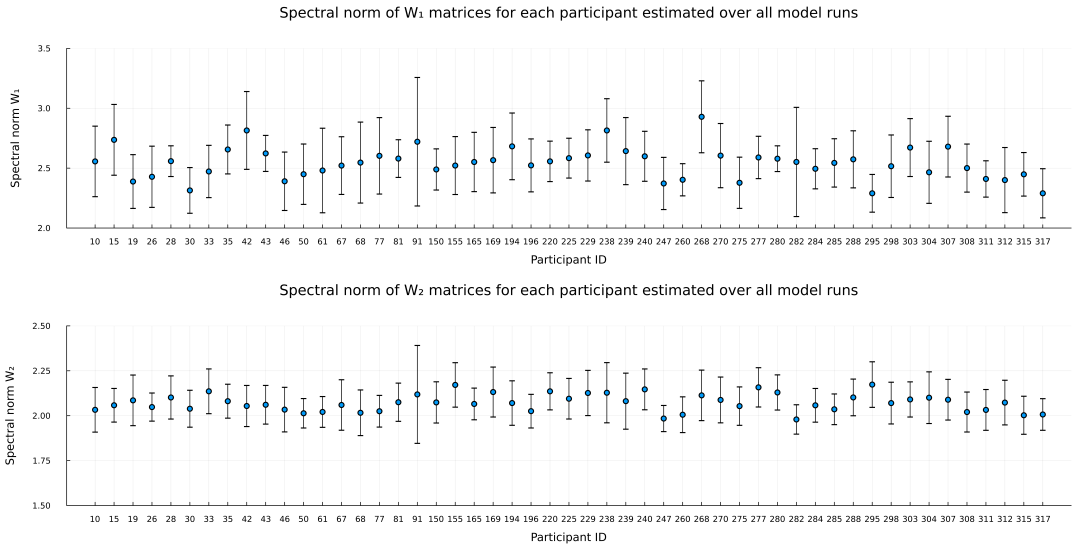


FIGURE 6.14: **Spectral norm of W matrices for each participant estimated over all model runs:** The spectral norm mean and standard deviation of the W_1 and W_2 matrices across model runs. Values of non-converged models were excluded.

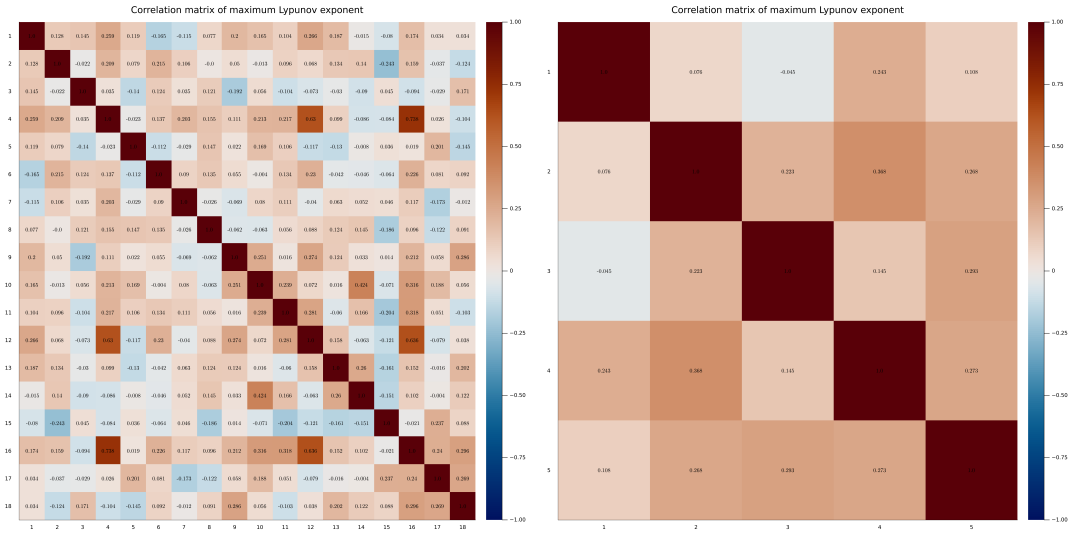
6.4 Investigating Correlations

Next, I will investigate the correlations between the maximal Lyapunov exponents and W matrix spectral norms. In the following I use the Pearson correlation coefficient, which is defined as follows given a paired data sample $\{(x_1, y_1), \cdots, (x_n, y_n)\}$:

$$r_{x,y} = \frac{\mathrm{CoV}(X,Y)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})}} \tag{6.2}$$

where $\bar{x}$ and $\bar{y}$ are the respective sample means ([43]). The elements of the correlation matrices shown in the following are the pairwise correlation coefficients $r_{ij} = \frac{\mathrm{CoV}(X_i,X_j)}{\sqrt{\mathrm{Var}(X_i)\mathrm{Var}(X_j)}}$ of the elements of the random vector $X = (X_1, \cdots, X_n)$.

I plotted correlation matrices of the maximal Lyapunov exponents in figure 6.15a and in figure 6.15b. What is shown are the correlations between runs across all participants. In the figure 6.15a all runs are included except for not converged runs. Because there are up to 2 non-converged runs per participant, I reordered the runs such that non-converged runs are indexed as runs 19 and 20 and excluded these rows and columns from the plots as they included NaNs/missing entries to calculate the statistics. The remaining model runs are ordered by sorting from lowest to highest D_{PSE}, i.e. the run with the lowest D_{PSE} is run 1, second lowest is run 2 etc. Because the run number/index is arbitrary this reordering does not make any differences in the results, only for the position on the matrix.



(A) **Correlation matrix of maximum Lypunov exponent over all converged model runs:** Because there are up to 2 not converged model runs per participant the runs were reordered such that the not converged runs are at position 19 and 20 and left out of the correlation matrix. Runs arranged in ascending order by D_{PSE}

(B) **Correlation matrix of maximum Lypunov exponent over filtered model runs:** Model runs were filtered by first selecting the runs with the lowest 10 D_{stsp}. Then the runs with the lowest 5 D_{PSE} are selected of those 10. Runs arranged in ascending order by D_{PSE}

FIGURE 6.15: Correlation matrices of the maximum Lypunov exponents

In the figure 6.15b I filtered the runs to only include the best models based on the D_{stsp} and D_{PSE} metrics. To be precise, I filtered the runs by first selecting the models with the 10 best D_{stsp} scores and then further filtered those 10 by only selecting the 5 models with the best D_{PSE} results. This results in 5 runs per participants which are indexed by sorting from best to worst D_{PSE} as in the previous correlation matrix.

The correlations are mostly weak and positive, as indicated by the heatmap coloring of the matrix entries. The idea behind the filtering was that the image would be decisively different by only including good models and only the stronger correlations would remain in the filtered matrix. This is not the case and the few stronger correlations in figure 6.15a are no longer presence in figure 6.15b. The negative correlations are almost entirely eliminated up to one entry very close to zero (-0.045).

I repeated the identical process for the spectral norms of the W_1 and W_2 matrices. The unfiltered correlation matrices are shown in figure 6.16, the filtered correlation matrices are shown in figure 6.17. The results here are similar, mostly weak positive correlations in the unfiltered matrices. In the filtered matrices all negative correlations are eliminated.

To conclude this work, I lastly searched for correlations between the dynamical features of the trained models and measurable traits of the participants. The different rs-fMRI frequency bands have been linked to personality traits, see [45] and [46], which find

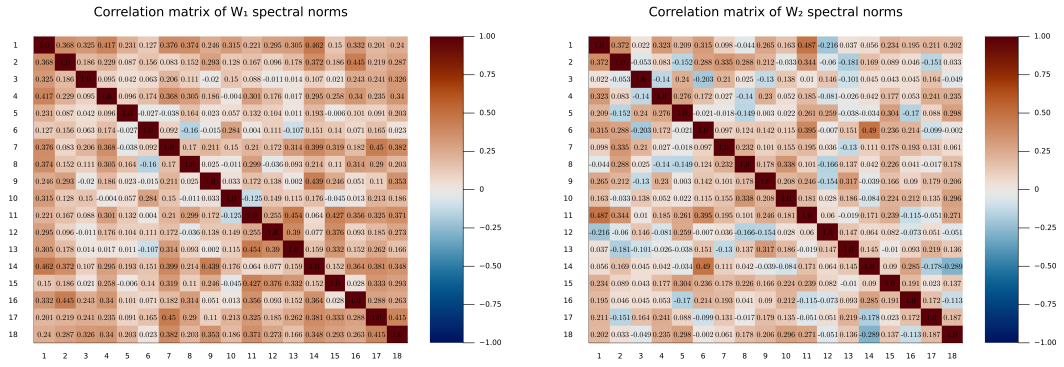


FIGURE 6.16: **Correlation matrices of W matrices spectral norms over all model runs:** Because there are up to 2 not converged model runs per participant the runs were reordered such that the not converged runs are at position 19 and 20 and left out of the correlation matrix. Runs arranged in ascending order by D_{PSE}

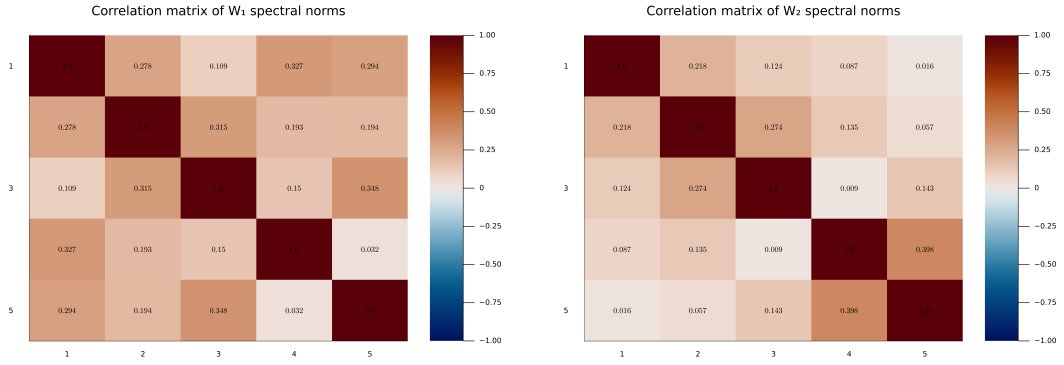


FIGURE 6.17: **Correlation matrices of W matrices spectral norms over filtered model runs:** Model runs were filtered by first selecting the runs with the lowest 10 D_{stsp}. Then the runs with the lowest 5 D_{PSE} are selected of those 10. Runs arranged in ascending order by D_{PSE}.

a connection between the traits Openness and Extraversion to analyzed rs-fMRI data. I calculated the mean maximum Lyapunov exponents for all participants, once over all converged models and then only over 5 best models (so same filtering as before) and correlated these with different personality measures. The personality measures are selection of those included in the LEMON dataset. I used the State-Trait Anxiety Inventory [47], the Hamilton rating scale for depression [48], and the big 5 personality traits [49]. Because not all participants are included in every personality measure I added the number of participants $N_{participants}$ for which data was available to the results. It should be noted that in this analysis, no correction for multiple testing was performed.

The results for the unfiltered exponents are shown in table 6.7, the results for the filtered exponents in table 6.8. There are no significant correlations.

TABLE 6.7: Correlation between personality measures and mean maximum Lyapunov exponents. Mean λ_{max} is calculated from all converged models. The results are statistically significant at a p-value$<$ 0.05. No correction for multiple testing was performed.

Personality Scale	$N_{Participants}$	Correlation	p-Value	Significant
STAI Anxiety	50	-0.093	0.519	False
Hamilton Depression scale	50	0.148	0.306	False
Neuroticism	50	-0.025	0.864	False
Extraversion	50	-0.053	0.716	False
Openness For Experiences	50	-0.275	0.053	False
Agreeableness	50	0.075	0.605	False
Conscientiousness	50	-0.016	0.911	False

TABLE 6.8: Correlation between personality measures and mean maximum Lyapunov exponents. Mean λ_{max} is only calculated from filtered models. The results are statistically significant at a p-value$<$ 0.05. No correction for multiple testing was performed.

Personality Scale	$N_{Participants}$	Correlation	p-Value	Significant
STAI Anxiety	50	0.053	0.716	False
Hamilton Depression scale	50	0.127	0.381	False
Neuroticism	50	0.026	0.860	False
Extraversion	50	0.057	0.695	False
Openness For Experiences	50	0.038	0.793	False
Agreeableness	50	0.039	0.789	False
Conscientiousness	50	-0.094	0.518	False

Chapter 7

Discussion and Outlook

7.1 Discussion

In this thesis, I investigated the use of the PLRNNs for training on fMRI time series. To this end I developed an observation model for BOLD signals found in fMRI and an inversion algorithm to incorporate this BOLD observation model into training using teacher forcing and backpropagation through time (BPTT). The BOLD observation model convolutes the latent states across time with the hemodynamic response function (hrf) before projecting them into observation space via a linear transformation. The inversion of the typically noisy observations to obtain teacher signals is achieved with a Wiener filter. The additional necessary inputs of the Wiener filter, the noise and denoised signal estimations, are obtained via Wavelet based noise estimation and denoising.

To benchmark this new method I created artificial benchmark datasets using the Lorenz63 system. The Lorenz system was convoluted with the hrf function and degraded with varying levels of noise to simulate the setting of fMRI time series. Overall, the BOLD observation model performed much better across all metrics than the standard linear observation model on these benchmark datasets in both noiseless and noisy settings. The BOLD observation model has the additional benefit of giving the user access to the unconvoluted latent space dynamics, which the linear observation model by design cannot.

I then tested this combination of PLRNN with a BOLD observation model on real fMRI time series from the publically available LEMON dataset. Only a selection of 51 participants was chosen from the LEMON dataset by filtering for low time dependence of the time series variance. Many participants showed increasing variance with the time of recording, which skews the results as the time series are not stationary. On 10 of these

selected participants I performed a parameter grid search for the teacher forcing α and the latent space dimension ℓ. The TF α has a small effect on the results, with very small (or zero) values improving the state space distance D_{stsp} while larger α-values improve the power spectrum error D_{PSE}. The latent space dimension ℓ had almost no effect on the results, only reducing the variance between runs when set equal to the observation dimension. The most likely explanation to me seems that the hidden dimension, which is another parameter of the shallow PLRNN (the PLRNN variant used in this work), was set too high, 50, and thus able to retain so much expressivity that the latent dimension did not influence the results.

I then trained 20 models with the chosen hyperparameters on each participant and investigated the results. The main complication on fMRI time series seems to be correctly evaluating the models because the time series are so short. This results in the state space distance to have a high variance, even in the same model, which I showed in section 5.6. In the result section we also saw that the power spectrum error on the (shorter) test set is (on average) lower than on the training set. This is most likely the result of over-smoothing the power spectrum, as I picked the smoothing kernel for a series length of 500, which is close to the training set length of 489. For the shorter test set (length 163) tends to get over-smoothed, which in turn improves D_{PSE}. The problem is that picking a smaller smoothing σ for the test set makes it hard to compare the results on the different sets, as it is no longer obvious that they can be compared.

When comparing the effects of good D_{stsp} and D_{PSE} by looking at different example trajectories I showed that a low D_{PSE} is desirable because the resting-state fMRI time series mostly have one or two large peaks in their power spectrum representing dominant frequencies. Higher D_{PSE}-values mean that the models did not learn these dominant frequencies. Unless D_{stsp} is very high there it is not clear how a lower D_{stsp} effects the trajectories. It may just fall into the variance of the measure for short trajectories which I documented. A simple dimensionality reduction with principal component analysis (PCA) didn't give a good intuition on the high dimensional trajectories.

Investigating the maximum Lyapunov exponent λ_{max} delivered some interesting insights. Most model runs learned a positive λ_{max}, meaning they produce chaotic dynamics. The models that did not learn a positive λ_{max} showed clear performance issues when evolved in time such as converging to a fixed point. Otherwise, the distribution of λ_{max} values was very consistent across all participants. Investigating the correlations between runs delivered no meaningful results. The maximum Lyapunov exponent also did not correlate with any personality measures, which makes sense considering the mean λ_{max}-values were very similar across all participants. Investigating the spectral norm values of the W connectivity matrices did also not provide any further meaningful insights.

7.2 Outlook

Introduce more realistic benchmarks: In this work I used the Lorenz63 dataset and simply added the convolution with the hemodynamic response function and noise to the signal. There are simulation frameworks for whole-brain modelling such as neurolib, [50], which allow the user to generate artificial BOLD signals. It allows the user to choose the neural mass model, the functional connectivity and the noise assumptions for the simulation and outputs both the (in practice hidden) neuronal activity and the corresponding BOLD output signal. Such a framework can be used to create higher quality benchmark data which is closer to the real fMRI use-case. In these benchmarks should consist of both long simulations to act as proof of concept that the PLRNN with BOLD observation can learn these time series and short simulations to explore the issues of real world fMRI time series typically being very short.

Addressing the stationarity issue: In many fMRI time series the variance increases with time, which violates the common stationarity assumption and makes it difficult to compare train and test sets. The method of testing for stationarity in this work is rather crude. Both the window size of the simple moving variance (SMV) and the cutoff for the correlation between SMV and time were chosen arbitrarily upon visual inspection of the results. In further work on fMRI time series there should be a more formalized approach to this issue such as more formal statistical tests for stationarity.

Analyze further dynamical features: In this work I only included a few features into my analysis like the maximum Lyapunov exponent. However, the PLRNN framework allows us to compute further features such as fixed points and cycles of the trained models. These were not included in this thesis and might provide further insights into the quality of the reconstructions, much like a negative maximum Lyapunov exponent seems to be a good criterion for discarding a model. As shown in [23] I is also possible to estimate λ_{max} from the data itself, so comparing the estimated λ_{max} from data and the one calculated from the model is another possibility.

Improve visualization: An important issue I encountered when working with fMRI data is simply the lack of a good visual representation of both the data and the generated trajectories, which is why I opted to plot individual trajectories in the results section to illustrate a point. Instead of using simple dimensionality reduction like I attempted in this work, [51] apply more modern dimensionality reduction methods, such as Uniform Manifold Approximation and Projection (UMAP), with very promising results. In my opinion, a good dimensionality reduction method would allow for greater insight into the geometry of the data and the reconstructions, which is currently lacking. The state space measure D_{stsp}, which is meant to give a measure of the geometrical overlap between the

generated trajectories and the data, is only of limited use in the fMRI context as the short time series lengths results in either a large variance and very limited resolution of D_{stsp}.

Explore multiple data modalities in LEMON: The LEMON dataset doesn't just include fMRI data, but also multiple other physiological measurements which were acquired simultaneously to the fMRI scans. In [52] the PLRNN framework was expanded to be able to train on multimodal observations. Training on these additional data modalities would be the next logical step in exploring LEMON.

Appendix A

Appendix

A.1 Proofs of Wavelet theorems

A.1.1 Proof for orthogonality relation for the wavelet transform, theorem 4.5

Proof. The proof is very straight forward using the frequency representation of the WT shown in Eq. 4.6, the Dirac delta δ and it's Fourier representation $2\pi\delta(x) = \int_{\mathbb{R}} e^{ikx}dk$.

$$\int_{\mathbb{R}}\int_{\mathbb{R}} \mathscr{W}_\psi\{f\}\mathscr{W}_\phi\{g\}^* \frac{dadb}{a^2} \tag{A.1}$$

$$= \int_{\mathbb{R}}\int_{\mathbb{R}} \left\{\sqrt{|a|}\int_{\mathbb{R}} \hat{f}(\omega)\hat{\psi}^*(a\omega)e^{ib\omega}d\omega\right\}\left\{\sqrt{|a|}\int_{\mathbb{R}} \hat{g}^*(\eta)\hat{\phi}(a\eta)e^{-ib\eta}d\eta\right\}\frac{dadb}{a^2} \tag{A.2}$$

$$= 2\pi\int_{\mathbb{R}}\int_{\mathbb{R}}\int_{\mathbb{R}} \hat{f}(\omega)\hat{g}^*(\eta)\hat{\psi}^*(a\omega)\hat{\phi}(a\eta)\left\{\frac{1}{2\pi}\int_{\mathbb{R}} e^{ib(\omega-\eta)}\right\}\frac{dad\omega d\eta}{|a|} \tag{A.3}$$

$$= 2\pi\int_{\mathbb{R}}\int_{\mathbb{R}}\int_{\mathbb{R}} \hat{f}(\omega)\hat{g}^*(\eta)\hat{\psi}^*(a\omega)\hat{\phi}(a\eta)\delta(\omega-\eta)\frac{dad\omega d\eta}{|a|} \tag{A.4}$$

$$= 2\pi\int_{\mathbb{R}} \hat{f}(\omega)\hat{g}^*(\omega)\left\{\int_{\mathbb{R}} \hat{\psi}^*(a\omega)\hat{\phi}(a\omega)\frac{da}{|a|}\right\}d\omega \tag{A.5}$$

Simplify the second integral on the R.H.S of A.5 by substituting $a\omega = \zeta, \omega = \xi \rightarrow \frac{1}{|a|}\,da\,d\omega = d\zeta\,d\xi$

$$\int_{\mathbb{R}} \hat{\psi}^*(a\omega)\hat{\phi}(a\omega)\frac{da}{|a|} = \int_{\mathbb{R}} \hat{\psi}^*(\zeta)\hat{\phi}(\zeta)d\zeta = C_{\psi,\phi} \tag{A.6}$$

Using A.6 in A.5 and applying the Plancherel theorem 3.6 we obtain the result.

$$\int_{\mathbb{R}}\int_{\mathbb{R}} \mathscr{W}_\psi\{f\}\mathscr{W}_\phi\{g\}^*\frac{dadb}{a^2} = 2\pi C_{\psi,\phi}\int_{\mathbb{R}} \hat{f}(\xi)\hat{g}^*(\xi)d\xi = C_{\psi,\phi}\int_{\mathbb{R}} f(t)g^*(t)dt \tag{A.7}$$

□

The proof of 4.11 works analogous, just the integral bounds over da must be adjusted.

A.1.2 Proof of Inverse continuous wavelet transform, definition 4.6

Proof. Using the orthogonality relation 4.5 the proof is again very straight-forward. Let $g \in L_2(\mathbb{R})$ be an arbitrary function.

$$\langle f, g\rangle_2 = \frac{1}{C_{\psi,\phi}} \int_{\mathbb{R}} \int_{\mathbb{R}} \mathscr{W}_\psi\{f\}(a,b)\mathscr{W}_\phi\{g\}^*(a,b)\frac{da\,db}{a^2} \tag{A.8}$$

$$= \frac{1}{C_{\psi,\phi}} \int_{\mathbb{R}} \int_{\mathbb{R}} \mathscr{W}_\psi\{f\}(a,b) \left\{ \int_{\mathbb{R}} g(t)^* \phi_{a,b}(t)\,dt \right\} \frac{da\,db}{a^2} \tag{A.9}$$

$$= \frac{1}{C_{\psi,\phi}} \int_{\mathbb{R}} g^*(t) \left\{ \int_{\mathbb{R}} \int_{\mathbb{R}} \mathscr{W}_\psi\{f\}(a,b)\phi_{a,b}(t)\frac{da\,db}{a^2} \right\} dt \tag{A.10}$$

$$= \left\langle \frac{1}{C_{\psi,\phi}} \int_{\mathbb{R}} \int_{\mathbb{R}} \mathscr{W}_\psi\{f\}(a,b)\phi_{a,b}(t)\frac{da\,db}{a^2}, g(t) \right\rangle_2 \tag{A.11}$$

Because g was chosen freely, we obtain the identity we wanted by the dominated convergence theorem on $L_2(\mathbb{R})$.

$$f(t) = \frac{1}{C_{\psi,\phi}} \int_{\mathbb{R}} \int_{\mathbb{R}^+} W_\psi\{f\}(a,b)\frac{1}{\sqrt{|a|}}\phi\left(\frac{t-b}{a}\right)\frac{da\,db}{a^2}$$

This identity only holds in the weak sense on $L_2(\mathbb{R})$, so the functions are identical almost everywhere but not pointwise. □

The proof of 4.11 works analogous, just the integral bounds over da must be adjusted.

A.2 Hyperparameter Settings

TABLE A.1: Hyperparameter settings for Lorenz benchmark runs.

latent dim	3
scalar saving interval	25
gaussian noise level	0.05000
optimizer	RADAM
start lr	0.00100
batch size	16
model	shallowPLRNN
batches per epoch	50
hidden layers	20
D stsp bins	30
PE n	20
observation model	HRF Identity
lat model regularization	0.00010
end lr	0.00000
device	cpu
gradient clipping norm	10
D stsp scaling	0.30000
image saving interval	25
num bases	30
hidden dim	50
sequence length	499
MAR ratio	0.00000
obs model regularization	0.00000
epochs	1000
MAR lambda	0.00500
weak tf alpha	0.10000
PSE smoothing	20
train test split	50000
min conv noise	0.00001
TR	Subject of experiment

TABLE A.2: Hyperparameter settings for training on the LEMON dataset.

latent dim	Subject of experiment
scalar saving interval	25
gaussian noise level	0.05000
optimizer	RADAM
batch size	16
start lr	0.00100
batches per epoch	50
model	clippedShallowPLRNN
hidden layers	20
D stsp bins	30
observation model	Regressor
lat model regularization	0.00000
PE n	20
end lr	0.00000
device	cpu
gradient clipping norm	0.00000
D stsp scaling	0.30000
image saving interval	25
num bases	50
hidden dim	50
sequence length	200
MAR ratio	0.00000
obs model regularization	0.00000
epochs	1000
MAR lambda	0.01000
weak tf alpha	Subject of experiment
min conv noise	0.00001
PSE smoothing	20.00000
train test split	0.75000
TR	1.40000

Bibliography

[1] Rui Dilão. *Dynamical System and Chaos: An Introduction with Applications*. Springer Nature, 2023.

[2] Steven H Strogatz. *Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[3] Giancarlo Benettin, Luigi Galgani, Antonio Giorgilli, and Jean-Marie Strelcyn. Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. part 1: Theory. *Meccanica*, 15:9–20, 1980.

[4] Ryan Vogt, Maximilian Puelma Touzel, Eli Shlizerman, and Guillaume Lajoie. On lyapunov exponents for rnns: Understanding information propagation using dynamical systems tools. *Frontiers in Applied Mathematics and Statistics*, 8:818799, 2022.

[5] Stephen Chung and Hava Siegelmann. Turing completeness of bounded-precision recurrent neural networks. *Advances in Neural Information Processing Systems*, 34: 28431–28441, 2021.

[6] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In *Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, September 10-14, 2006. Proceedings, Part I 16*, pages 632–640. Springer, 2006.

[7] Avrim Blum and Ronald Rivest. Training a 3-node neural network is np-complete. *Advances in neural information processing systems*, 1, 1988.

[8] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, pages 1183–1188. IEEE, 1993.

[9] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2): 157–166, 1994.

[10] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[13] Sachin S Talathi and Aniket Vartak. Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771*, 2015.

[14] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[15] Michael Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Eighth Annual Conference of the Cognitive Science Society, 1986*, pages 513–546, 1986.

[16] Fernando J Pineda. Dynamics and architecture for neural computation. *Journal of Complexity*, 4(3):216–245, 1988.

[17] Jonas Mikhaeil, Zahra Monfared, and Daniel Durstewitz. On the difficulty of learning chaotic dynamics with rnns. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 11297–11312. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/495e55f361708bedbab5d81f92048dcd-Paper-Conference.pdf.

[18] Kenji Doya et al. Bifurcations in the learning of recurrent neural networks 3. *learning (RTRL)*, 3:17, 1992.

[19] Daniel Durstewitz. A state space approach for piecewise-linear recurrent neural networks for identifying computational dynamics from neural measurements. *PLoS computational biology*, 13(6):e1005542, 2017.

[20] Georgia Koppe, Hazem Toutounji, Peter Kirsch, Stefanie Lis, and Daniel Durstewitz. Identifying nonlinear dynamical systems via generative recurrent neural networks with applications to fmri. *PLoS computational biology*, 15(8):e1007263, 2019.

[21] Dominik Schmidt, Georgia Koppe, Zahra Monfared, Max Beutelspacher, and Daniel Durstewitz. Identifying nonlinear dynamical systems with multiple time scales and long-range dependencies. *arXiv preprint arXiv:1910.03471*, 2019.

[22] Manuel Brenner, Florian Hess, Jonas M Mikhaeil, Leonard F Bereska, Zahra Monfared, Po-Chen Kuo, and Daniel Durstewitz. Tractable dendritic rnns for reconstructing nonlinear dynamical systems. In *International Conference on Machine Learning*, pages 2292–2320. PMLR, 2022.

[23] Jonas Mikhaeil, Zahra Monfared, and Daniel Durstewitz. On the difficulty of learning chaotic dynamics with rnns. *Advances in Neural Information Processing Systems*, 35:11297–11312, 2022.

[24] Florian Hess, Zahra Monfared, Manuel Brenner, and Daniel Durstewitz. Generalized teacher forcing for learning chaotic dynamics. *arXiv preprint arXiv:2306.04406*, 2023.

[25] Peter T Fox. The coupling controversy. *Neuroimage*, 62(2):594–601, 2012.

[26] William D Penny, Karl J Friston, John T Ashburner, Stefan J Kiebel, and Thomas E Nichols. *Statistical parametric mapping: the analysis of functional brain images*. Elsevier, 2011.

[27] Valery Serov et al. *Fourier series, Fourier transform and their applications to mathematical physics*, volume 197. Springer, 2017.

[28] Steven W Smith et al. The scientist and engineer's guide to digital signal processing, 1997.

[29] S. Puthusserypady. *Applied Signal Processing*. NowOpen Series. Now Publishers, 2021. ISBN 9781680839784. URL https://books.google.de/books?id=fwwfzgEACAAJ.

[30] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. doi: 10.1109/JPROC.2004.840301. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[31] AF Jones and DL Misell. The problem of error in deconvolution. *Journal of Physics A: General Physics*, 3(5):462, 1970.

[32] N. Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series: With Engineering Applications*. Massachusetts Institute of Technology : Paperback series. M.I.T. Press, 1964. ISBN 9780262730051.

[33] Thomas schafer2006recurrent. Wavelets vorlesung, 2012. https://www.fim.uni-passau.de/fileadmin/dokumente/fakultaeten/fim/lehrstuhl/sauer/geyer/Wavelets.pdf, Last visited on 2023/09/15".

[34] Stéphane Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 2008. ISBN 9780080922027.

[35] Firdous A Shah and Azhar Y Tantary. *Wavelet Transforms: Kith and Kin*. CRC Press, 2022.

[36] Ingrid Daubechies. *Ten lectures on wavelets*. SIAM, 1992.

[37] Gilbert Strang and Truong Nguyen. *Wavelets and filter banks*. SIAM, 1996.

[38] David L Donoho and Iain M Johnstone. Ideal spatial adaptation by wavelet shrinkage. *biometrika*, 81(3):425–455, 1994.

[39] David L Donoho. De-noising by soft-thresholding. *IEEE transactions on information theory*, 41(3):613–627, 1995.

[40] John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–317. IEEE, 2007.

[41] Anahit Babayan, Miray Erbey, Deniz Kumral, Janis D Reinelt, Andrea MF Reiter, Josefin Röbbig, H Lina Schaare, Marie Uhlig, Alfred Anwander, Pierre-Louis Bazin, et al. A mind-brain-body dataset of mri, eeg, cognition, emotion, and peripheral physiology in young and old adults. *Scientific data*, 6(1):1–21, 2019.

[42] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2):130–141, 1963.

[43] John H McDonald. *Handbook of biolological statistics*. New York•, 2014.

[44] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[45] Shigeyuki Ikeda, Hikaru Takeuchi, Yasuyuki Taki, Rui Nouchi, Ryoichi Yokoyama, Yuka Kotozaki, Seishu Nakagawa, Atsushi Sekiguchi, Kunio Iizuka, Yuki Yamamoto, et al. A comprehensive analysis of the correlations between resting-state

oscillations in multiple-frequency bands and big five traits. *Frontiers in Human Neuroscience*, 11:321, 2017.

[46] Julien Dubois, Paola Galdi, Yanting Han, Lynn K Paul, and Ralph Adolphs. Resting-state functional brain connectivity best predicts the personality dimension of openness to experience. *Personality neuroscience*, 1:e6, 2018.

[47] Charles Donald Spielberger. Manual for the state-trait anxiety inventory (self-evaluation questionnaire). *(No Title)*, 1970.

[48] Max Hamilton. A rating scale for depression. *Journal of neurology, neurosurgery, and psychiatry*, 23(1):56, 1960.

[49] Paul T Costa and Robert R McCrae. *NEO PI/FFI manual supplement for use with the NEO Personality Inventory and the NEO Five-Factor Inventory*. Psychological Assessment Resources, 1989.

[50] Caglar Cakan, Nikola Jajcay, and Klaus Obermayer. neurolib: A simulation framework for whole-brain neural mass modeling. *Cognitive Computation*, Oct 2021. ISSN 1866-9964. doi: 10.1007/s12559-021-09931-9. URL https://doi.org/10.1007/s12559-021-09931-9.

[51] Javier Gonzalez-Castillo, Isabel S Fernandez, Ka Chun Lam, Daniel A Handwerker, Francisco Pereira, and Peter A Bandettini. Manifold learning for fmri time-varying functional connectivity. *Frontiers in Human Neuroscience*, 17, 2023.

[52] Manuel Brenner, Georgia Koppe, and Daniel Durstewitz. Multimodal teacher forcing for reconstructing nonlinear dynamical systems. *arXiv preprint arXiv:2212.07892*, 2022.

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 29.09.2023

Eric Volkmann