A Mathematical Analysis of Backpropagation and Predictive Coding

Ludwig-Maximilians-Universität München

Fakultät für Mathematik, Informatik und Statistik

Bachelor of Mathematics

Bachelor's Thesis

June 07, 2018

Author
Samuel Lippl

Supervisor
Prof. Dr. Markus Heydenreich

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

Samuel Lippl, 7. Juni 2018

Abstract

Deutsch

Neurale Netzwerke, zunächst als biologisches Modell des Gehirns eingeführt, haben sich als erfolgreiche Methode im Machine Learning etabliert, speziell wenn sie ihre Parameter mithilfe von Gradientenabstieg durch den Backpropagation-Algorithmus lernen. Dies hat die Frage aufgeworfen, ob unser Gehirn eine Art von Backpropagation implementiert. In dieser Bachelorarbeit stelle ich meinen Entwurf eines allgemeinen Modells neuraler Netzwerke vor und definiere in diesem Kontext den Backpropagation-Algorithmus und ein Modell neuronaler Komputation namens *Predictive Coding*. Ich zeige, dass die mithilfe von *Predictive Coding* angepassten Parameter unter gewissen Bedingungen gegen die Backpropagation-Parameter konvergieren und untersuche die biologische Plausibilität beider Methoden. Anschließend untersuche ich ein einfaches lineares *Predictive Coding*-Modell genauer, das sich mit oder ohne *a priori*-Verteilungen implementieren lässt. Ich zeige, dass *Predictive Coding* mit immer höheren Varianzen der *a priori*-Verteilungen gegen *Predictive Coding* ohne *a priori*-Verteilungen konvergiert.

English

Although first introduced as a biological model of the brain, neural networks have found an important application as predictive algorithms in Machine Learning, most notably if they learn their parameters by gradient descent using the backpropagation algorithm. Its success has led to the discussion whether some form of backpropagation may be implemented in the brain. In this Bachelor's Thesis, I introduce a general framework of neural networks, in which I define the backpropagation algorithm and a model of neuronal computation that is called predictive coding. I prove conditions under which the parameters that are found by predictive coding converge to the ones that the backpropagation algorithm yields and discuss biological plausibility of both algorithms. I go on by studying a simple linear predictive coding model in more depth. This may be implemented with or without priors. I prove that predictive coding with priors that have increasingly high variance is equivalent to implementing predictive coding without priors.

Contents

Chapter 1

An Introduction to Computation in Machines, Brains and Mathematics

> What kind of computer is the brain?
>
> Sejnowski et al. [1, p. 1300]

In "Science and Statistics" [2], George E. P. Box discusses the proper statistical methodology of tackling scientific problems. Although not at all concerned with this discipline, the paper gives a remarkably clear perspective on the field of computational neuroscience, not only with respect to its methodology but also with respect to its findings. The article therefore provides a proper compass, to position this Bachelor's Thesis at the intersection of statistics and Machine Learning, computational neuroscience and, evidently, mathematics.

Computational neuroscience is concerned with explaining "how electrical and chemical signals are used to represent and process information" [1, p. 1299]. These problems are tackled by constructing theoretical models that are then checked by empirical data. Like the methodology of many fields of science, this strategy corresponds to the general principle of "iteration between theory and practice" [2, p. 791] that Box discusses. It is important that this loop is closed: while omission of theory leads to the unjustified application of a few recipes to problems that are not suitable for these analyses – a problem which Box calls "cookbookery" [2, p. 797] –, "mathematistry", the "development of theory for theory's sake", has the tendency "to redefine the problem rather than solve it" [2, p. 797].

Besides ensuring a valid analysis of the data, theory plays a second important role in science: making certain that the obtained explanations actually contribute to our understanding. Sejnowski et al. [1] situate neural models on a spectrum from realistic to simplifying: The Hodgkin-Huxley model of signal transmission [3] in nerves is a perfect example of a realistic model. At the level of a single neuron, it describes the formation of an action potential, an electrical signal that is used for long-distance information transmission. Realistic models, i. e. those that attempt to describe the physiological properties of a neuronal network, are insufficient for two reasons:

Firstly, a huge amount of data is needed to fit these models, especially at the network level, and the computational power to simulate entire networks at this level is immense. Secondly, even if the data were available – which cannot be taken for granted, especially with concern to the human brain –, such a complex model may not enlighten our understanding of how the human brain works. A simplifying model may be better suited to illustrate important principles of information processing. These models are more concerned with how information is represented at a network level without considering the underlying physiological principles. Sejnowski et al. elaborate:

> The study of simplifying models of the brain can provide a conceptual framework for isolating the basic computational problems and understanding the computational constraints that govern the design of the nervous system. [1, p. 1300]

This Bachelor's Thesis is concerned with the mathematical treatment of a popular simplifying model: neural networks. Keeping in mind the concerns that the previous arguments have raised, there are two balancing acts such a treatment has to accomplish:

Theory and Application Neural networks have originally been inspired by a biological model [4]. Inherent to the concept of a simplifying model is, however, the possibility that such a network may become "an end in itself" [1, p. 1305]. This outcome may be framed as a mathematization in Box's terms. However, this development has been integrated into the field of Machine Learning that is concerned with the question "How may we recognize patterns?" instead of the neuroscientific question "How may our brain recognize patterns?". Neural networks have become a powerful tool in Machine Learning in a way that cannot be reframed as a model of our brain. In particular, the backpropagation algorithm has been a popular method to perform gradient descent in neural networks that is not likely to be implemented by our brain [5]. Other Machine Learning algorithms are even less related to a biological model of learning.
Does this mean that such a development is unsuitable for further neuroscientific discussion? No. The differences between how Machine Learning algorithms may be implemented in the brain and how learning in the brain occurs in reality provides fruitful ground for new ideas in both disciplines and discussion whether differences are due to constraints of evolution or optimality.
One example for such a mechanism is the development of the *predictive coding model* that was first proposed and successfully applied by Rao and Ballard [6] to explain certain observations in the visual cortex. Friston developed and generalized this model (see, for instance, [7]). Recently, Whittington and Bogacz presented predictive coding in its relation to the backpropagation algorithm [8].
It is helpful for these discussions to have a general framework for neural networks, whether they are biologically plausible or not. I propose such a definition in chapter 2. It lays the groundwork for a mathematically rigorous and general introduction to backpropagation and predictive coding as well as their relations. This treatment may lead to predictive coding becoming just another end in itself. While empirical data is not of concern to a mathematical Bachelor's Thesis, I therefore introduce mathematical axioms in neural networks that have been supported by empirical data in order to prevent the mathematistry of my discussion of neural networks.

Generality and Conceptualization In the spirit of a mathematical analysis, chapter 2 will introduce the concepts of prediction and learning in neural networks as well as backpropagation and predictive coding as generally as possible. This introduction shall be complemented by more restrictive conditions for which these networks will be simplified and easier to understand. An important goal of this Bachelor's Thesis is making definitions as general as possible without losing their usefulness. Chapter 3 will conduct first more concrete analyses and consider possible directions of future investigations.
The principles I have set forth will guide my attempt to present neural networks such that the analysis provides the rigorosity and generality of a mathematical treatment, is sufficiently grounded in reality to contribute to computational neuroscience and sufficiently concrete to provide clear results (or perspectives of clear results).

Chapter 2

Feedforward Neural Networks and Predictive Coding

This chapter will be concerned with introducing general neural networks. Section 2.1 will cover their structure. We will go on to states of neural networks in 2.2 before introducing learning in neural networks in 2.3. The specific theory of learning by backpropagation and predictive coding will be introduced alongside the more general groundwork.

2.1 Structure of Neural Networks

2.1 (Neural Networks as Graphs) Neural networks may be based on graphs where we refer to the nodes as *Processing Units* (PU) and to the edges as *Unit Connections* (UC). We can therefore use an adjacency matrix that specifies for each pair of PUs whether there is a connection between them (see 2.6) to specify the neural network.
Moreover, there is a set of parameters that specify the behaviour of a neural network. As most of these parameters are directly connected to a UC, we will define a neural network by its *weight matrix* which is a weighted adjacency matrix. However, we now face a difficulty concerning the relationship between the structure and the parameters. Generally, a connection between two PUs is indicated by a non-zero entry in the adjacency matrix while an entry of zero indicates that the two PUs are not connected. However, what if a connection between two PUs is weighted by zero? The difference between this case and no connection at all will become important in section 2.3: a connection with weight zero may change during learning whereas two unconnected PUs will always remain unconnected. Moreover, how may we define a neural network where we know the UCs but have no specification of their weights?
Subsection 2.1.1 presents a solution to these difficulties. Subsection 2.1.2 provides some vocabulary to talk about neural networks by introducing the standardized layerization. Subsequently, subsection 2.1.3 will first introduce concepts from predictive coding, namely the generative model. This section will conclude with 2.1.4 where we will embed the provided definition in the context of literature and will discuss the concept of *parallelity* and its connection to neural networks.

2.1.1 The Network Matrix

2.2 Definition (ς-χ-extension of a set). If A is a set, we define its ς-χ-*extension*

$$A_{\varsigma,\chi} := (A \times \{\varsigma, \chi\}) \cup \{\chi_A\}, \tag{2.1}$$

where we denote for any $a \in A$

$$a_\varsigma \equiv (a, \varsigma) \quad a_\chi \equiv (a, \chi), \tag{2.2}$$

and may omit the subscripts when they become clear from context.
We will treat 0_ς like an ordinary zero, i. e. a number in a matrix is omitted if and only if it is 0_ς and a diagonal matrix only has 0_ς entries that are not on its diagonal.
If $a \in A \times \{\varsigma\}$, then a is *constant* and we write $\mathrm{St}(a) = \varsigma$ (St is an abbreviation of *status*). If $a \in (A \times \{\chi\}) \cup \{\chi\}$, then a is *variable* and we write $\mathrm{St}(a) = \chi$. Finally, we denote the value of a by $\mathrm{Val}(a_\varsigma) = \mathrm{Val}(a_\chi) = a$ where $\mathrm{Val}(\chi)$ is not defined.

2.3 Definition (ς-χ-extension of a function). Consider sets I, A^i, where $i \in I$, B and C.

a) We define the *passive* ς-χ-extension of a function $f : \prod_{i\in I} A^i \to B$ by

$$\begin{aligned} f &: \prod_{i\in I} A^i_{\varsigma,\chi} \to B_{\varsigma,\chi}, \\ \mathrm{Val}\left(f(a)\right) &:= \begin{cases} f\left((\mathrm{Val}(a_i))_{i\in I}\right) & \text{if } \neg \underset{i\in I}{\exists} a_i = \chi, \\ c & \text{if } \left(\underset{i\in I}{\exists} a_i = \chi\right) \wedge f\left((\mathrm{Val}(a_i))_{i\in I: a_i \neq \chi}, \cdot\right) \equiv c, \\ \chi & \text{else}, \end{cases} \\ \mathrm{St}\left(f(a)\right) &:= \begin{cases} \varsigma & \text{if } \underset{i\in I}{\forall} \mathrm{St}(a_i) = \varsigma, \\ \chi & \text{else .} \end{cases} \end{aligned} \tag{2.3}$$

We will denote the passive extension by the same symbol as the original function.

b) We define the *active* ς-χ-extension of a function

$$f : \prod_{i\in I} A^i \times C \to \prod_{i\in I} A^i \times B, \begin{pmatrix} a \\ c \end{pmatrix} \mapsto \begin{pmatrix} (f_i(a,c))_{i\in I} \\ f_B(a,c) \end{pmatrix}$$

in $A := (A_i)_{i\in I}$ by

$$\begin{aligned} f_{\varsigma,\chi:A} &: \prod_{i\in I} A^i_{\varsigma,\chi} \times C \to \prod_{i\in I} A^i_{\varsigma,\chi} \times B, \\ f_{\varsigma,\chi:A}(a,c)_i &:= \begin{cases} f(a,c)_i & \mathrm{St}(a_i) = \chi, \\ a_i & \mathrm{St}(a_i) = \varsigma. \end{cases} \end{aligned} \tag{2.4}$$

If $C = \emptyset$, we simply write $f_{\varsigma,\chi}$.

2.4 (Passive Extension) The passive extension allows application of a function f to ς-χ-extended values. $f(a)$ is constant if and only if every argument is constant. Generally, the value of the extended f is simply the value of the original f when applied to the values of the arguments, for instance:

$$1_\varsigma + 2_\chi = (1+2)_\chi = 3_\chi.$$

This is covered by the first case of the definition. Things only get messy when the value of an argument is unknown. In this case, generally, $f(a)$ is unknown as well, for instance:

$$1_\varsigma + \chi = \chi.$$

The only exception is given by a function that is constant in the unknown argument, for instance:

$$0_\varsigma \cdot \chi = 0_\chi.$$

2.5 (Active Extension) The active extension ensures that no constant value will be changed. This will, for instance, be useful in the learning function, where we do not want 0_ς to change as this value encodes that there is no connection between two PUs, as we discussed in 2.1.

2.6 (Adjacency Matrix) An adjacency matrix has the following connection to a graph: Given a graph $\mathcal{G} := (\mathcal{V}, \mathcal{E}), \mathcal{E} \subseteq \mathcal{V}\times\mathcal{V}$, we may write $\mathcal{G}$ as a matrix $W \in \{0,1\}^{\mathcal{V}\times\mathcal{V}}$ where $W_{v,w} = 1 \Leftrightarrow (v,w) \in \mathcal{E}$. If we consider weighted edges, we may generalize the notion to a matrix $W \in A^{\mathcal{V}\times\mathcal{V}}$ on a ring A where $W_{v,w} \neq 0$ specifies the weight of edge $(v,w) \in \mathcal{E}$ and an edge with weight zero is synonymous with no edge at all. It is therefore important that the use of these weights is defined in an appropriate way. We can further extend the notion of an adjacency matrix to $W \in A^{\mathcal{V}\times\mathcal{V}}_{\varsigma,\chi}$ where

$$(v,w) \in \mathcal{E} \Leftrightarrow W_{v,w} \neq 0_\varsigma. \tag{2.5}$$

2.7 Definition (Graph Relations). If $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ is a graph and $v, w \in \mathcal{V}$, we introduce the following three relations:

$$v \to_{\mathcal{E}} w \equiv (v, w) \in \mathcal{E}, \tag{2.6}$$
$$v \leftarrow_{\mathcal{E}} w \equiv w \to_{\mathcal{E}} v, \tag{2.7}$$
$$v \smile_{\mathcal{E}} w \equiv v \to_{\mathcal{E}} w \vee v \leftarrow_{\mathcal{E}} w, \tag{2.8}$$

where we will often leave the subscript implicit.
We define their transitive closures $\to^*, \leftarrow^*, \smile^*$. If $v \to^* w$ (resp. $v \to w$) we say that v precedes (resp. directly precedes) w and w succeeds (resp. directly succeeds) v. v and w are connected (resp. directly connected) if and only if $v \smile^* w$ (resp. $v \smile w$).
We will denote the set of directly preceding nodes by $\to v$ and the set of directly succeeding nodes by $v \to$.

2.8 Definition (Connected Graph). A graph is called *connected* if and only if all its nodes are connected.

2.9 Definition (Neural Network I: Topological Structure). Consider a ring A and a finite set $\mathcal{V} \neq \emptyset$. A tupel $\mathfrak{N} = (\mathcal{V}, \theta), \theta \in A_{\varsigma,\chi}^{\mathcal{V}\times\mathcal{V}}$ is called a neural network if and only if the graph corresponding to θ is connected.

2.10 Remark. The condition that neural networks have to be connected is not a restriction because if there are two unconnected parts in the network, we can simply consider these parts as separate networks.

2.11 Notation (Adjacency Matrix). While generally using the common matrix notations, we will refer to the elements of θ in a slightly different way. The entry in row v and column w is denoted by $\theta_{v\to w}$, the entries in row v are denoted by $\theta_{v\to}$ and the entries in column w are denoted by $\theta_{\to w}$. For $V, W \subseteq \mathcal{V}$

$$\theta_{V\to W} \equiv (\theta_{v\to w})_{v\in V, w\in W} \in \mathbb{R}^{V\times W}.$$

Finally, we define the identity matrix I on an arbitrary domain, i. e. for sets M, N, we define

$$I_{M\to N} := (\delta_{mn})_{m\in M, n\in N},$$

where δ_{ij} is the Kronecker delta.

2.12 Notation. For any family $(a_i)_{i\in I}$, if $J \subseteq I$, we denote $a_J \equiv (a_j)_{j\in J}$. If there is a set of functions $f_i : A_i \to B_i$, $i \in I$, and $a \equiv (a_i)_{i\in I}, a_i \in A_i$, we denote $f(a) \equiv (f_i(a_i))_{i\in I}$.

2.13 Definition (States). Given a neural network $\mathfrak{N} = (\mathcal{V}, \theta)$, we will call

$$s \in S[\mathfrak{N}] := A^{\{i,o\}\times\mathcal{V}}$$

a *state* of $\mathfrak{N}$. If $(k, v) \in \{i, o\} \times \mathcal{V}$, we denote the (k, v)-entry in s by $s^{v:k}$. We refer to $s^{v:i}$ as the *input value* of v and $s^{v:o}$ as the *output value* of v.
Furthermore, we denote, for $v \in \mathcal{V}, V \subseteq \mathcal{V}$ and $k \in \{i, o\}$

$$s^{V:k} \equiv \left(s^{v:k}\right)_{v\in V}, \quad s^{:k} \equiv s^{\mathcal{V}:k}, \quad s^{v:} \equiv \left(s^{v:i}, s^{v:o}\right), \quad s^{V:} \equiv \left(s^{v:}\right)_{v\in V}.$$

We define the ς-χ-*extended state* $s \in S_{\varsigma,\chi}[\mathfrak{N}] := A_{\varsigma,\chi}^{\{i,o\}\times\mathcal{V}}$.
Using the analogous notation, we define the *state* of $\mathfrak{N}$ in $V \subseteq \mathcal{V}$ by

$$S^V[\mathfrak{N}] := A^{\{i,o\}\times V}, \quad S^{V:k}[\mathfrak{N}] := A^{\{k\}\times V},$$

and the ς-χ-*extended state* by

$$S_{\varsigma,\chi}^V[\mathfrak{N}] := A_{\varsigma,\chi}^{\{i,o\}\times V}, \quad S_{\varsigma,\chi}^{V:k}[\mathfrak{N}] := A_{\varsigma,\chi}^{\{k\}\times V}.$$

2.14 Definition (Neural Network II: Functional Structure). Let $\mathfrak{N} = (\mathcal{V}, \theta)$ be a neural network. Consider a set of *global parameters* Λ, *unit functions* $f \equiv (f^u)_{u\in\mathcal{V}}$ and *integrative functions* $g \equiv (g^u)_{u\in\mathcal{V}}$. The tupel (Λ, f, g) is called the *functional structure* of $\mathfrak{N}$ if and only if, for all $u \in \mathcal{V}$,

$$f^u : A \to A, \quad s^u \mapsto f^u(s^u), \tag{2.9}$$

and every g^u is defined as the active extension of some $\tilde{g}^u$ in $S^{\{u\}:i}[\mathfrak{N}]$ such that

$$\tilde{g}^u : S[\mathfrak{N}] \times A_{\varsigma,\chi}^{\mathcal{V}\times\mathcal{V}} \times \Lambda_{\varsigma,\chi} \to A, \quad (s^{:o}, \theta, \lambda) \mapsto \tilde{g}^u_{\theta,\lambda}(s^{:o}), \tag{2.10}$$

only depends on $\{s^v : \mathrm{Val}(\theta_{v\to u}) = 0\}$, $\theta_{\to u}$ and λ.
By convention, we define g by $\tilde{g}$ and leave its extension implicit.

2.15 Remark. We cannot simply demand that $\tilde{g^u}$ only depend on the state $s^{\to u}$ as we do not want it to depend on values where $\theta_{v\to u} = 0_\chi$.

2.16 (Integrative Functions) A unit's integrative function summarizes the output of a unit's directly preceding nodes. The first argument therefore provides the values of these nodes, the second argument provides the weight matrix θ and the third argument is a set of global parameters. How neural networks may adopt concrete states will be of concern in section 2.2.

2.17 (Unit Functions) The unit functions relate the summarized input to a node $s^{u:i} := g^u_{\theta,\lambda}(s)$ to its output value $s^{u:o} := f^u(s^{u:i})$.

2.18 Definition (Additive Neural Networks). A neural network $\mathfrak{N} = (\mathcal{V}, \theta)$ is called *straight* if and only if

$$\underset{v,w\in\mathcal{V}}{\forall} v \to w \Rightarrow \neg \underset{u\in\mathcal{V}}{\exists} v \to u \to^* w.$$

A straight neural network $\mathfrak{N} = (\mathcal{V}, \theta)$ with functional structure $(\Lambda, f, g) \equiv (f, g), \Lambda = \emptyset$ is called *additive* if and only if

$$g_\theta(s^{:o}) = \theta^T s^{:o}. \tag{2.11}$$

$\mathfrak{N}$ is called *general additive* neural network if and only if, for all $u \in \mathcal{V}$, there are sets $i_u, o_u \subseteq \mathcal{V} \setminus i_u$ such that

$$g^u_\theta(s^{i_u:o}, s^{o_u:o}) = \theta^T_{i_u\to u} s^{i_u:i} + \theta^T_{o_u\to u} s^{o_u:o}. \tag{2.12}$$

2.19 Straightness of neural networks ensures that there are no hidden dependencies. If v is not only directly connected to w but also indirectly, g^w does not specify the entire dependence on the value of v. It is therefore crucial for an additive network to be straight; if not, the influence of the preceding nodes would seem additive according to g but hidden dependencies may render it non-linear.

2.1.2 Layerization

2.20 (Layers of Neural Networks) Many neural networks are defined in layers where $\mathcal{V} := V^{(0)}\dot\cup\cdots\dot\cup V^{(L)}$ and $v \to w \Leftrightarrow \underset{l\le L}{\exists} \left(v \in V^{(l)} \wedge w \in V^{(l+1)}\right)$. These layers can be extended to a class of neural networks called *feedforward neural networks*. This subsection is concerned with finding and characterizing these layers, a process called *layerization*.

2.21 Notation. To simplify the notation of layers we define, for $l, l_1, \ldots, l_n \in \{0, \ldots, L\}$,

$$V^{(\square l)} := \bigcup_{k\in\{0,\ldots,L\}:k\square l} V^{(k)},$$

where $\square$ is some relation, e. g. $\le$, and

$$V^{(l_1,\ldots,l_n)} := \bigcup_{i=1}^{n} V^{(l_i)}.$$

2.22 Definition (Feedforward Neural Networks). A neural network is called feedforward if and only if its graph is acyclical. Feedforwardness is therefore a topological property.

2.23 Proposition (Characterization of feedforward neural networks). Let $\mathfrak{N} = (\mathcal{V}, \theta)$ be a neural network. Then, the following conditions are equivalent:

(i) $\mathfrak{N}$ is feedforward,

(ii) $\to$ is acyclical,

(iii) There are layers $V^{(0)}, \dots, V^{(L)}$ such that $\mathcal{V} = V^{(0)} \dot\cup \dots \dot\cup V^{(L)}$ and

$$V^{(l)} \ni v \to w \in V^{(k)} \Rightarrow l < k, \tag{2.13}$$

(iv) There is a function $\pi : \mathcal{V} \to \{1, \dots, |\mathcal{V}|\}$ such that

$$v \to w \Rightarrow \pi(v) < \pi(w), \tag{2.14}$$

(v) There is a function $\pi : \mathcal{V} \to \{1, \dots, |\mathcal{V}|\}$ such that

$$\theta_\pi := (\theta_{v\to w})_{\pi(v),\pi(w)} \tag{2.15}$$

is an upper triagonal matrix.

Proof. *(i)⇔(ii)* Follows by definition.
(i)⇒(iii) is a special case of Corollary 2.38[1].
(iii)⇒(iv) Assign numbers in $\{1, \dots, |\mathcal{V}^{(0)}|\}$ to the elements of $\mathcal{V}^{(0)}$. For any $0 < l \leq L$ assign numbers in $\left\{\sum_{j=1}^{l-1} |\mathcal{V}^{(j)}| + 1, \dots, \sum_{j=1}^{l} |\mathcal{V}^{(j)}|\right\}$ to the elements of $\mathcal{V}^{(l)}$. Then $v \to w$ where $v \in \mathcal{V}^{(l)}$ and $w \in \mathcal{V}^{(k)}$ implies $l < k$ and therefore

$$\pi(v) \leq \sum_{j=1}^{l} |\mathcal{V}^{(j)}| < \sum_{j=1}^{l} |\mathcal{V}^{(j)}| + 1 \leq \sum_{j=1}^{k} |\mathcal{V}^{(j)}| + 1 \leq \pi(w).$$

(iv)⇒(v) We simply prove that (2.14) implies (2.15). This becomes clear from the fact that

$$(\theta_\pi)_{\pi(v),\pi(w)} \neq 0_\varsigma \Rightarrow \theta_{v\to w} \neq 0_\varsigma \Rightarrow v \to w \Rightarrow \pi(v) < \pi(w),$$

which is the definition of an upper triagonal matrix.
(v)⇒(iv) We now prove that (2.15) implies (2.14). This follows from

$$v \to w \Rightarrow (\theta_\pi)_{\pi(v),\pi(w)} \neq 0_\varsigma \Rightarrow \pi(v) < \pi(w).$$

(iv)⇒(ii) Clearly, (2.14) could not hold if $\to$ contained a cycle. □

2.24 Condition (iii) has a more universal appeal which we will discover now. These layers can be specified either by a partition of $\mathcal{V}$ or a transitive total order $\preceq$. After proving their equivalence, we go on to provide the definition of different kinds of layerizations.

2.25 Definition (Layerization). A transitive total (but not necessarily antisymmetric) order $\preceq$ on $\mathcal{V}$ is called layerization. We denote

$$v \prec w \equiv v \preceq w \wedge w \not\preceq v, \quad v \simeq w \equiv v \preceq w \wedge w \preceq v.$$

A connection $v \to w$ is called

- *forward* if and only if $v \prec w$,
- *lateral* if and only if $v \simeq w$,

[1] Note that we do not use the proposition in any proof until then.

- *backward* if and only if $v \succ w$.

The corresponding equivalence classes $\mathcal{V}/\simeq$ are called *layers*. As $\preceq$ is transitive, we may define $\preceq$ on $\mathcal{V}/\simeq$ by

$$V \preceq W \equiv \underset{v\in V, w\in W}{\forall} v \preceq w \equiv \underset{v\in V, w\in W}{\exists} v \preceq w. \tag{2.16}$$

As $\preceq$ is antisymmetrical on $\mathcal{V}/\simeq$, we can now notate the layerization as a partition $\mathcal{V} = \dot{\bigcup}_{l=0}^{L} V^{(l)}$ where

$$\mathcal{V}/\simeq = \left\{V^{(l)} : 0 \leq l \leq L\right\}, \quad V^{(l)} \prec V^{(l+1)},$$

which yields the layerization form of Proposition 2.23.
We will call $\preceq$ the *ordinal* and the partition $\left(V^{(l)}\right)_{0\leq l\leq L}$ the *set notation* of the layerization. L is called the *depth* of the layerization.

2.26 Proposition. There is a one-to-one correspondence between ordinal and set layerization.

Proof. The function from ordinal to set layerization has already been given by Definition 2.25. On the other hand, if there is a set layerization, the order that is defined by

$$V^{(l)} \ni v \preceq w \in V^{(k)} \Leftrightarrow l \leq k,$$

is total and transitive.
We now only have to prove that the ordinal notation converted to the set notation converted to the ordinal notation provides indeed the same layerization. Let $v, w \in \mathcal{V}$. WLOG, suppose that $v \preceq w$. Then $v \prec w \dot{\vee} v \simeq w$ where $\dot{\vee}$ is the exclusive disjunction. If $v \prec w$, there are different equivalence classes $V \neq W$ such that $v \in V, w \in W$. As $v \prec w$, $V \prec W$ and therefore layer V comes before layer W. This, again, yields $v \prec w$. The other case, $v \simeq w$, works in an analogous way. □

2.27 By Proposition 2.26, we will use the relational and set form of layerization interchangeably from now on. The next definition provides an important stepping stone towards the function of layerizations by formalizing the notion of forward, lateral and backward connections by a relation: the flow[2]. The main motivation behind this is Definition 2.32.

2.28 Definition. Given a layerization $\preceq$ of a network $\mathfrak{N}$, we define the following relations which we will refer to as the *flow* of $\mathfrak{N}$.

$$\begin{aligned} v \trianglerighteq w &\equiv v \smile w \wedge v \preceq w, \qquad v \trianglelefteq w \equiv w \trianglerighteq v, \quad v \square w \equiv v \trianglerighteq w \wedge v \trianglelefteq w, \\ v \triangleright w &\equiv v \trianglerighteq w \wedge \neg(v \square w), \qquad\qquad\qquad\quad v \triangleleft w \equiv w \triangleright v. \end{aligned} \tag{2.17}$$

2.29 Lemma. $\underset{v,w\in\mathcal{V}}{\forall} v \smile w \Leftrightarrow v \square w \dot{\vee} v \triangleright w \dot{\vee} v \triangleleft w$ where $\dot{\vee}$ is an exclusive disjunction.

Proof. Suppose $\neg(v \smile w)$. It is evident from its definition that there is no flow between v and w in this case. In turn, if $v \smile w$, the totality of $\preceq$ implies the trichotomy and the definition of the strict inequalities implies that the trichotomy is exclusive. □

2.30 Definition. A layerization $\preceq$ on $\mathcal{V}$ is called

a) *minimal* if and only if there is a sequence $v_0, \ldots, v_L$ such that $v_l \in \mathcal{V}^{(l)}$ for all $l = 0, \ldots, L$ and $v_l \triangleright v_{l+1}$ for all $l = 0, \ldots, L-1$,

b) *spanned* if and only if the minimal and maximal values of $\preceq$ and $\triangleright^*$, the transitive closure of $\triangleright$, are identical,

c) *leftwards* (resp. *rightwards*) *anchored* if and only if for any $v_l \in \mathcal{V}^{(l)}$ there is a sequence $v_0, \ldots, v_{l-1}$ such that for all $k = 0, \ldots, l-1$, $v_k \triangleright v_{k+1}$ and $v_k \in V^{(k)}$ (resp. a sequence $v_{l+1}, \ldots, v_L$ such that for all $k = l, \ldots, L-1$, $v_k \triangleright v_{k+1}$ and $v_{k+1} \in V^{(k+1)}$).

[2] which is different from the standard flow in graph theory as defined, for instance, in [9, p. 149]

2.31 Lemma. An anchored layerization is minimal.

Proof. If the layerization is leftwards anchored, set $l = L$. If the layerization is rightwards anchored, set $l = 0$. □

2.32 Definition. Two layerizations $\preceq_1, \preceq_2$ of a neural network $\mathfrak{N}$ are called *equivalent* (Not.: $\preceq_1 \sim \preceq_2$) if and only if the flow remains identical.

2.33 Layerizations provide different perspectives on a neural network. Generally, however, we want our notion of forward, lateral or backward connections (especially whether a given network is feedforward) to remain invariant. Equivalent layerizations ensure such invariance.

2.34 Lemma. Given a layerization $\preceq$ and a chain $v_0 \triangleright \cdots \triangleright v_n$ where $v_0 \in V^{(l)}$ and $v_n \in V^{(k)}$, we obtain $n \leq k - l$.

Proof. Proof by induction on $k - l$. By definition, v_n has to lie in a higher layer than v_0 if $v_0 \neq v_n$. Therefore, if $k < l$, such a chain is not possible and if $k = l$, $n = 0$.
If $k - l \geq 1$, consider $v_{n-1} \in V^{(k')}$ and observe that $k' < k$ and therefore $k' \leq k - 1$. Then, by induction hypothesis, $n - 1 \leq k' - l \leq k - 1 - l$ and therefore $n \leq k - l$. □

2.35 Theorem (Standardized Layerization). For every flow $\triangleright$ of a neural network $\mathfrak{N}$, there is a unique minimal spanned and leftwards (resp. rightwards) anchored layerization $\preceq_l$ (resp. $\preceq_r$) such that the flow of $\preceq_l$ (resp. $\preceq_r$) is $\triangleright$. We call $\preceq_l$ (resp. $\preceq_r$) *leftwards* (resp. *rightwards*) *standardized layerization*.

Proof. If there are no $v, w \in \mathcal{V}$ such that $v \triangleright w$, a minimal layerization must necessarily have depth $L = 0$ and we therefore obtain $V^{(0)} = \mathcal{V}$ and uniqueness for both the leftwards and the rightwards standardized layerization. This is equivalent to the original layerization: if $v \square w$, then $v \smile w$ and therefore $v \square_l w$ and $v \square_r w$. This proves equivalence as there are no v, w such that $v \triangleright w$. The minimal and maximal values of $\preceq_l$, $\preceq_r$ and $\triangleright^*$ are given by $V^{(0)} = V^{(L)} = \mathcal{V}$ and therefore, $\preceq_l$ and $\preceq_r$ are spanned. Anchoredness is rendered trivial which proves the first case. We will therefore choose $v, w \in \mathcal{V}$ such that

$$v \triangleright w \tag{2.18}$$

and define $V^{(0)}$ as the minimal values of $\triangleright^*$ and $V^{(\max)}$ as the maximal values of $\triangleright^*$. These sets are uniquely defined as this is equivalent to the condition that the layerization is spanned. The two sets are also disjoint. In order to prove that, we prepare the following lemma:

2.36 Lemma. If $u \in V^{(0)} \cap V^{(L)}$,

$$\underset{v \in \mathcal{V}}{\forall} u \square^* v.$$

Proof. Let $v \in \mathcal{V}$. By definition of neural networks, $u \smile^* v$. As $\mathcal{V}$ is finite, the definition of the transitive closure directly implies that there is some chain $v_1, \ldots, v_{n-1} \in \mathcal{V}$ such that

$$u \smile v_1 \smile \cdots \smile v_{n-1} \smile v.$$

We prove $u \square^* v$ by induction on n. If $n = 1$, $u \smile v$ and therefore $u \triangleright v \vee u \square v \vee u \triangleleft v$. As u is both a maximal and a minimal element, this implies $u \square v$ and therefore $u \square^* v$.
Now suppose $n > 1$. By induction hypothesis, we know that $u \square^* v_{n-1}$. On the other hand, as $v_{n-1} \smile v$. As $\trianglerighteq^*$ is transitive, v_{n-1} is therefore also maximal. By the same argument as above, $v_{n-1} \square^* v$ and therefore $u \square^* v$. △

Now, suppose there is some $u \in V^{(0)} \cap V^{(L)}$. Using the PUs v, w from (2.18), we conclude by Lemma 2.36 $v \square^* u \square^* w$ and therefore $v \square^* w$ which is a contradiction to (2.18) and proves that $V^{(0)} \cap V^{(L)} = \emptyset$.
We apply the following algorithm to obtain the remaining layers if we want a leftwards anchored layerization:

1. Set $t = 0$ and $R^{(0)} := \mathcal{V} \setminus \left(V^{(0)} \cup V^{(L)}\right)$,
2. While $R^{(t)} \neq \emptyset$:
   (a) Define $V^{(t+1)}$ as the set of minimal values in $R^{(t)}$,
   (b) Set $R^{(t+1)} := \mathcal{R}^{(t)} \setminus V^{(t+1)}$,
   (c) Increment t by 1,
3. Set $L := t$.

Clearly, this yields a partition. As $R^{(L)}$ is empty, every $v \in \mathcal{V}$ is in some layer and the construction ensures that the sets $V^{(0)}, \dots, V^{(L-1)}$ as well as the sets $V^{(1)}, \dots, V^{(L)}$ are pairwise disjoint where we have already proven that $V^{(0)} \cap V^{(L)} = \emptyset$.
The layerization is leftwards anchored as we prove by induction on l in the definition: For $v_0 \in V^{(0)}$, the criterion is trivial. If $l = 1$, we obtain some $v_1 \in V^{(1)}$. As $v_1 \notin V^{(0)}$, it is not minimal and there is some $v_0 \in \mathcal{V}$ such that

$$v_0 \triangleright v_1.$$

As v_1 is minimal in $R^{(0)} = \mathcal{V} \setminus \left(V^{(0)} \cup V^{(L)}\right)$ we conclude $v_0 \in V^{(0)} \cup V^{(L)}$. As v_0 is not maximal, $v_0 \in V^{(0)}$ and we have found our chain. Finally, if $l > 1$ and $v_l \in V^{(l)}$, we know that v_l was minimal in $R^{(l-1)}$ but not in $R^{(l-2)}$. Therefore, there must be some $v_{l-1} \in V^{(l-1)}$ such that $v_{l-1} \triangleright v_l$ and the existence of the chain follows by induction hypothesis.
On the other hand, the layers $V^{(1)}, \dots, V^{(L-1)}$ are uniquely defined by the condition that the layerization must be leftwards anchored. Take any $v \in V^{(l)}$, $1 \leq l \leq L-1$, and suppose that there is an equivalent, leftwards anchored and spanned layerization $\preceq'$ with set notation $V'^{(0)}, \dots, V'^{(L')}$ such that $v \in V'^{(k)}, k \neq l$. If $k < l$, $\preceq'$ would be impossible by Lemma 2.34 because there is a chain $V'^{(0)} = V^{(0)} \ni v_0 \triangleright \dots \triangleright v_{l-1} \triangleright v \in V'^{(k)}$ as $\preceq_l$ is leftwards spanned. If $k > l$, the existence of $\preceq'$ would imply, by the same argument, that $\preceq_l$ is impossible. As we have already shown that $\preceq_l$ is well-defined, this is a contradiction. Therefore, $k = l$ and $\preceq_l$ is uniquely defined.
An analogous method shows that $\preceq_r$ is unique and well-defined where we use a top-down instead of a bottom-up approach with the algorithm and all arguments thereafter. □

2.37 Corollary. For every layerization $\preceq$ on a neural network $\mathfrak{N}$, there is a unique equivalent leftwards or rightwards standardized layerization.

Proof. The layerization is provided by the flow defined by $\preceq$ and Theorem 2.35. □

2.38 Corollary. If $\mathfrak{N}$ is a feedforward network and $v \trianglerighteq w \equiv v \to w$, there is a unique leftwards or rightwards standardized layerization. Canonically, we consider one of these layerizations and only specify which one, if it makes a difference.

2.39 Proposition (Depth of Layerizations). Let $\mathfrak{N}$ be a neural network. Given a layerization $\preceq$ of $\mathfrak{N}$, if $\preceq'$ and $\preceq''$ are equivalent minimal layerizations with depths L' and L'', we obtain

$$L' = L''.$$

We call L' the depth of $\mathfrak{N}$. Furthermore, any equivalent layerization that is not minimal has a depth of at least $L' + 1$.

Proof. It is sufficient to prove $L' \leq L''$. As $\preceq'$ is minimal, we can find $v_0, \dots, v_{L'}$ such that $V'^{(0)} \ni v_0 \triangleright \dots \triangleright v_{L'} \in V'^{(L')}$. Considering $\preceq''$, let $v_0 \in V''^{(k)}$ and $v_{L'} \in V''^{(l)}$. By the invariance of the flow under equivalent layerizations and Lemma 2.34, $L' \leq l - k \leq L''$. This also proves that any equivalent layerization (whether it is minimal or not) has a depth of at least L'. If we have proven that any layerization of depth L' is minimal, the second part of the proposition is proven. Consider, again, the chain $v_0, \dots, v_{L'}$. Let $v_0 \in V^{(k)}$ according to the equivalent layerization. If $k > 0$, it is easy to see that the remaining chain yields a contradiction. Therefore, $v_0 \in V^{(0)}$. Analogously, we can prove by induction that $v_l \in V^{(l)}$ and the layerization is therefore minimal. □

2.40 We have therefore found a way to layerize any feedforward network in a sensible way. On the other hand, when we extend the feedforward network to a more general network, we can still talk about the network's layers in a standardized manner. The next section lays the groundwork for this very necessity.

2.1.3 The Generative Model

2.41 Definition (Stochastic Networks). Consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. We define a *stochastic network* $\mathfrak{S} := (\mathcal{V}, \theta)$ with functional structure (Λ, f, g), $\Lambda = \Omega \times \Lambda'$ such that, for all $u \in \mathcal{V}$,

$$g^u_{\theta,(\omega,\lambda')}(s) = g^u_{\theta,\lambda'}(\omega), \tag{2.19}$$

where

$$g^u_{\theta,\lambda'} : \Omega \to \mathbb{R}$$

is a random variable. We may then consider the entire network as a random vector

$$g_{\theta,\lambda'} : \Omega \to \mathbb{R}^{\mathcal{V}}.$$

2.42 (Predictive Coding I: The Generative Model) The notion of stochastic networks allows us to lay the groundwork for predictive coding. According to this theory, the neural model of reality is *hierarchical*: a feedforward network $\mathfrak{H} = (\mathcal{V}, \theta)$ where the distribution of the lower-level PUs is specified as dependent on the realization of their preceding, higher-level PUs. This is motivated in an applied context by characterizing the higher-level PUs as causes of the lower-level PUs. Generally, this means that $\to v$ is the set of causes of v. In turn, v is a cause for all $v \to$.
As an example, we will consider a small sector of a generative model that is concerned with vision. It is of interest to many organisms to estimate the speed of an object being thrown towards them. If we restrict ourselves to visual information, the only knowledge that is available to the organism is the input to the retinal receptors

$$\text{ret} \subset \mathcal{V}.$$

The retinal input is modeled as stochastically dependent on the properties of the object being thrown. For instance, there may be a PU

$$\text{ball} \in \mathcal{V}$$

encoding whether the object being thrown is a ball. As the information that an object is being thrown is not a part of the network, the value of the PU ball may not be dependent on any other PU. We thus have an integrative function representing the *a priori* distribution (or *prior*) of ball:

$$g^{\text{ball}}_{\theta,\lambda'} \sim \text{Bernoulli}(p_{\lambda'}), \quad 0 \leq p_{\lambda'} \leq 1.$$

(As $\text{ball} \in V^{(0)}$, p cannot depend on θ.)
It is possible that the model does not contain any information on the likelihood of a given object being a ball. In this case, ball would not be part of the actual stochastic network but rather a value that has to be specified beforehand and determines the distribution of the stochastic network itself. We will consider this case in 3.9.
The speed of an object may now depend on whether this object is a ball, i. e., if

$$\text{speed} \in \mathcal{V}$$

encodes the speed of the objects, we may observe that

$$\text{ball} \to \text{speed}.$$

The speed itself may be normally distributed[3] given its causes:

$$g^{\text{speed}}_{\theta,\lambda'}(s) \sim \mathcal{N}\left(\mu^{\text{speed}}_{\theta,\lambda'}(s), \Sigma^{\text{speed}}_{\theta,\lambda'}(s)\right), \quad \mu^{\text{speed}}_{\theta,\lambda'}, \Sigma^{\text{speed}}_{\theta,\lambda'} : \mathbb{R}^{\mathcal{V}} \to \mathbb{R}.$$

where μ^{speed} and Σ^{speed} are parameters that specify the normal distribution. More generally, we may have some (increasing) distribution

$$p^{\text{speed}}_{\theta,\lambda'}(s) : \mathbb{R} \to [0, 1]$$

[3] Negative speed would encode the object moving in the reverse direction.

of $g^{\text{speed}}_{\theta,\lambda'}(s)$.
Similarly, the speed of the objects may shape the retinal input:

$$\text{speed} \to \text{ret}.$$

Under comparable assumptions (but keeping in mind that there are multiple retinal inputs), we may conclude

$$g^{\text{ret}}_{\theta,\lambda'}(s) \sim \mathcal{N}\left(\mu^{\text{ret}}_{\theta,\lambda'}(s), \Sigma^{\text{ret}}_{\theta,\lambda'}(s)\right), \quad \mu^{\text{speed}}_{\theta,\lambda'} : \mathbb{R}^{\mathcal{V}} \to \mathbb{R}^{\text{ret}}, \quad \Sigma^{\text{speed}}_{\theta,\lambda'} : \mathbb{R}^{\mathcal{V}} \to \mathbb{R}^{\text{ret}\times\text{ret}}.$$

Again, we may replace these parameters with a general multivariate distribution

$$p^{\text{ret}}_{\theta,\lambda'}(s) : \mathbb{R}^{\text{ret}} \to [0,1].$$

We have thus provided a path from a higher-level cause towards the input onto the retina by

$$\text{ball} \to \text{speed} \to \text{ret}.$$

In general, however, the higher-level cause is not only not known, it is not even empirically measurable. How we infer higher-level causes from lower-level input and how we construct these causes will be the concern of the following section 2.2. For now, we will provide a general definition of a generative model.

2.43 Notation. We denote independence between random variables X and Y by

$$X \perp Y.$$

2.44 Definition. Consider a set X and a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Stochastic functions $f, g : X \times \Omega \to \mathbb{R}$ are called *independent* if and only if

$$\underset{x\in X}{\forall} f(x) \perp g(x).$$

2.45 Definition (Generative Models). Consider a feedforward stochastic network $\mathfrak{G} = (\mathcal{G}, \xi)$ with functional structure $(\Lambda' \times \Omega, f, g)$ and distribution

$$g_{\xi,\lambda'}(s) \sim \mathbb{P}_{\xi,\lambda'}(s).$$

$\mathbb{P}_{\xi,\lambda'}$ is called *generative distribution* if and only if f is constant in Ω and

$$\underset{v,w\in\mathcal{G}}{\forall} v \to^* w \Rightarrow g^v \perp g^w.$$

In this case $\mathfrak{G}$ is called *generative model*.

2.46 Definition (Normal Generative Model). A generative model $\mathfrak{G}$ is called *normal generative model* if and only if there is a functional structure (Λ', f, μ) and, for any $\lambda' \in \Lambda'$, some positive definite covariance matrix

$$\Sigma_{\lambda'} \in \mathbb{R}^{\mathcal{G}\times\mathcal{G}}$$

such that

$$g_{\xi,\lambda'}(s) = \mu_{\xi,\lambda'}(s) + \epsilon_{\lambda'}, \quad \epsilon_{\lambda'} \sim \mathcal{N}(0, \Sigma_{\lambda'}).$$

2.47 Remark. Because a normal generative model is also a generative model, we know that

$$\underset{v,w\in\mathcal{G}}{\forall} v \to^* w \Rightarrow \Sigma_{v\to w} = 0.$$

2.1.4 Alternative Definitions of Neural Networks

2.48 This subsection is dedicated to a brief embedding of the provided definition of neural networks into the literature. I have mostly based my definition on [4]. However, my definition is more explicit than the introduction and more general than the later sections, mostly because in the spirit of a mathematical treatment, I differentiated between applications and the general definitions.

2.49 (Parallelity) Parallelity is a crucial aspect of neural networks. Massively parallel pathways allow for the transformation of input space that is necessary for tasks like visual recognition. On the other hand, parallel pathways reduce the error rate of unreliable neural networks. These considerations have led Guresen and Kayakutlu [10] to claim parallelity as a necessary feature of a neural network. In their proposed definition, they require that "at least two of the multiple [Processing Units are] connected in parallel" [10, p. 428] because, they argue, "structures [...] with one or more [Processing Units] connected serially cannot be referred as [a neural network] because it will lose the power of parallel computing and starts to act more like existing computers than a brain" [10, p. 428].
In principle, I agree with their point. I would argue, however, that it is not helpful to include such a criterion in the definition. After all, if there is one parallel pathway in a massive serial network, this would correspond to a neural network according to their notion even though the difference to a uniquely serial network is insubstantial.
The definition therefore seems to raise an issue that cannot be resolved and is analogous to the definition of a "heap of sand", another instance of the Sorites paradox (see [11]): without doubt, one grain of sand is necessary to have a heap but it is not sufficient and we can probably not find a hard definition of such a vague statement. We would be better advised to simply define the underlying property a "heap" refers to, i. e. the *number* of grains of sand that allows us to use more rigorous notions.
Comparably, parallel processing such that it is advantageous cannot be covered by such a broad definition – not only does parallelity exist on a spectrum, it also strongly depends on the context. I would therefore suggest that an attempt to find a way to talk about parallelity, for instance by a *degree of parallelity*, may be more fruitful than Guresen and Kayakutlu's hard borders. It is, after all, rather unlikely that the exclusion of a few pathological cases would improve a mathematical definition.

2.2 States of Neural Networks

2.50 In the previous section, we have defined the topology of neural networks: their connective pattern and their functional structure. It remains an open question how this topology determines the values of the neural network. We will therefore introduce *state functions* relating states of input PUs to states of the entire network. The *canonical state functions* provide an intuitive connection between the functional structure of a network and its state attribution and are the ultimate goal of subsection 2.2.1.
The canonical functions are especially clear if we consider feedforward neural networks where the input is provided to the lowest layer $V^{(0)}$. On the other hand, the notation is particularly simple if the network is additive. We will therefore explore these cases in subsection 2.2.2.
Finally, the network functions provide an appropriate framework for inference in predictive coding. We will introduce the concept of *free energy* and then introduce inference in subsection 2.2.3.

2.2.1 Network Functions

2.51 Definition (State Functions). Consider a neural network $\mathfrak{N} = (\mathcal{V}, \theta)$. We define the *input* $\mathcal{I} \subseteq \mathcal{V}$ and the *output* $\mathcal{O} \subseteq \mathcal{V}$. Then,

$$T : S^{\mathcal{I}:i}_{\varsigma,\chi}[\mathfrak{N}] \times A^{\mathcal{V}\times\mathcal{V}}_{\varsigma,\chi} \times \Lambda \to S^{\mathcal{O}}_{\varsigma,\chi}[\mathfrak{N}], \qquad (i, \theta, \lambda) \mapsto T_{\theta,\lambda}(i), \tag{2.20}$$

a state function and consider the following special cases:

a) If $\mathcal{O} = \mathcal{V}$, T is called a *network function*.

b) If $\mathcal{I} = \mathcal{O} = \mathcal{V}$, T is called a *step*.

c) A network function Init_a such that $\text{Init}_a^{\mathcal{I}}(i) = i$ and $\text{Init}_a^{\mathcal{V}\setminus\mathcal{I}}(i) = (a)_{v\in\mathcal{V}\setminus\mathcal{I}}$ where $a \in A_{\varsigma,\chi}$ is called a-initialization. Where this is clear from context, we generally identify any input vector i with its χ-initialization $\text{Init} \equiv \text{Init}_\chi$.

2.52 Notation (Defining state functions). It is rather bothersome to define extended functions. We may therefore define the state function by

$$\tilde{T} : S^{\mathcal{I}:i}[\mathfrak{N}] \times A^{\mathcal{V}\times\mathcal{V}} \times \Lambda \to S^{\mathcal{O}}[\mathfrak{N}]$$

and then define T as the active extension in $S^{\mathcal{I}\cap\mathcal{O}:i}[\mathfrak{N}]$ (along with the implicit passive extensions), i. e.

$$T \equiv \tilde{T}_{\varsigma,\chi:S^{\mathcal{I}\cap\mathcal{O}:i}}[\mathfrak{N}]. \tag{2.21}$$

On the other hand, we may also define state functions by

$$\tilde{T} : S^{\mathcal{I}}[\mathfrak{N}] \times A^{\mathcal{V}\times\mathcal{V}} \times \Lambda \to S^{\mathcal{O}}[\mathfrak{N}],$$

where the output value of the state $s \in S^{\mathcal{I}}[\mathfrak{N}]$ has no influence on $\tilde{T}$ other than specifying which states are constant when defining $T \equiv \tilde{T}_{\varsigma,\chi:S^{\mathcal{I}\cap\mathcal{O}}}[\mathfrak{N}]$.

2.53 Remark. The previous paragraph points out the use of ς-χ-extension of states. In this case, we may specify that certain states (most often, the states of the input PUs) are not allowed to change. Without such a concept, the values of input PUs s^v would simply be overwritten by $g^v(s)$.

2.54 (Special State Functions) The meaning of the special state functions will become clearer in Definition 2.57 when we define their canonical form. As a brief introduction, a network function obviously determines the state of an entire network. On the other hand, a step of a neural network resembles a single step in the development of the network towards the final state of a network function. An important property of a sensible network function is *balancedness* which we will define below.

2.55 Definition (Balancedness). A state $s \in S[\mathfrak{N}]$, where $\mathfrak{N} = (\mathcal{V}, \theta)$ is a neural network with functional structure (Λ, f, g), is *balanced* if and only if

$$s^{:o} = f(s^{:i}), \quad s^{:i} = g_{\theta,\lambda}(s^{:o}). \tag{2.22}$$

We define

$$B[\mathfrak{N}] := \{s \in S[\mathfrak{N}] : s \text{ balanced}\}.$$

A network function T is balanced if and only if for all

$$T\left(S_{\varsigma,\chi}^{\mathcal{I}:i}[\mathfrak{N}]\right) \subseteq B[\mathfrak{N}].$$

A state function T is balanced if and only if there is a balanced network function $\tilde{T}$ such that

$$\tilde{T}^{\mathcal{I}} = T \circ \text{Init}.$$

2.56 Notation. If s is balanced and, for some $u \in \mathcal{V}$, $f^u = \text{Id}$, we may notate

$$s^u \equiv s^{u:i} = s^{u:o}.$$

2.57 Definition (Canonical State Functions). Consider a network $\mathfrak{N} = (\mathcal{V}, \theta)$. Consider a topology $\mathcal{T}$ on A.

a) The *canonical step* T for a state $s \in S_{\varsigma,\chi}^{\mathcal{V}:i}[\mathfrak{N}]$ is given by

$$T_{\theta,\lambda}^{:o}(s) := T_{\theta,\lambda}^{:o}[\mathfrak{N}](s) := f\left(s^{:i}\right), \quad T_{\theta,\lambda}^{v:i}(s) := T_{\theta,\lambda}^{v:i}[\mathfrak{N}](s) := \begin{cases} g_{\theta,\lambda}^v\left(T_{\theta,\lambda}^{:o}(s)\right) & \text{if } \text{St}(s^{v:i}) = \chi, \\ s^{v:i} & \text{if } \text{St}(s^{v:i}) = \varsigma. \end{cases} \tag{2.23}$$

We omit the specification of the neural network $\mathfrak{N}$ if it is clear from context.

b) The *canonical network function* N for a state $i \in S_{\varsigma,\chi}^{\leq \mathcal{V}:i}[\mathfrak{N}]$ where, for any $V \subseteq \mathcal{V}$,

$$S_{\varsigma,\chi}^{\leq V}[\mathfrak{N}] := \bigcup_{W \subseteq V} S_{\varsigma,\chi}^{W}[\mathfrak{N}],$$

is given by

$$N_{\theta,\lambda'}(i) := N_{\theta,\lambda}[\mathfrak{N}](i) := \lim_{n\to\infty} T_{\theta,\lambda}^{n}[\mathfrak{N}](\mathrm{Init}(i)), \tag{2.24}$$

and is only defined for values where this limit exists in $(S[\mathfrak{N}], \mathcal{T}^{S[\mathfrak{N}]})$ where $\mathcal{T}^{S[\mathfrak{N}]}$ is the product topology on $S[\mathfrak{N}]$. We again omit the specification of the neural network if it is clear.

2.58 (Convention: Real Neural Networks) From now on, we will by convention restrict ourselves to real neural networks, i. e. $A = \mathbb{R}$ together with the the norm topology.

2.59 Lemma (Fixed States). Let FP_T be the set of fixed points of T, i. e.

$$\mathrm{FP}_T := \{s \in S[\mathfrak{N}] : T_{\theta,\lambda}(s) = s\}.$$

In this case,

$$B[\mathfrak{N}] = \mathrm{FP}_T \subseteq N_{\theta,\lambda}\left(S_{\varsigma,\chi}[\mathfrak{N}]\right) = N_{\theta,\lambda}\left(S_{\varsigma,\chi}^{\leq \mathcal{V}}[\mathfrak{N}]\right)$$

and, if T is continuous,

$$\mathrm{FP}_T = N\left(S_{\varsigma,\chi}[\mathfrak{N}]\right)$$

Proof. $B[\mathfrak{N}] = \mathrm{FP}_T$: The fixed point equation corresponding to (2.23) is (2.22).
$\mathrm{FP}_T \subseteq N_{\theta,\lambda}\left(S_{\varsigma,\chi}[\mathfrak{N}]\right)$: If $s \in S[\mathfrak{N}]$ is a fixed point of T, then

$$N_{\theta,\lambda}(s) = \lim_{n\to\infty} T_{\theta,\lambda}^{n}(s) = \lim_{n\to\infty} s = s.$$

$N_{\theta,\lambda}\left(S_{\varsigma,\chi}[\mathfrak{N}]\right) \subseteq N_{\theta,\lambda}\left(S_{\varsigma,\chi}^{\leq \mathcal{V}}[\mathfrak{N}]\right)$ follows directly from $S_{\varsigma,\chi}[\mathfrak{N}] \subseteq S_{\varsigma,\chi}^{\leq \mathcal{V}}[\mathfrak{N}]$.
On the other hand, for any $i \in S_{\theta,\varsigma}^{\leq \mathcal{V}}[\mathfrak{N}]$, $N_{\theta,\lambda}(i) = N_{\theta,\lambda}\left(\mathrm{Init}(i)\right)$.
$\mathrm{FP}_T \supseteq N\left(S_{\varsigma,\chi}[\mathfrak{N}]\right)$: Suppose that S is continuous. Then, for any $i \in S_{\varsigma,\chi}[\mathfrak{N}]$:

$$N_{\theta,\lambda}(i) = \lim_{n\to\infty} T_{\theta,\lambda}^{n}(i) = T_{\theta,\lambda}\left(\lim_{n\to\infty} T_{\theta,\lambda}^{n}(i)\right) = T_{\theta,\lambda}\left(N_{\theta,\lambda}(i)\right).$$

Therefore, $N_{\theta,\lambda}(i)$ is a fixed point of T. □

2.60 (Constant Values and Network Functions) We often want the input to remain constant when applying a network function T. This is the reason why T is defined for extended values.

2.61 Definition (Implementations). Consider a neural network $\mathfrak{N} := (\mathcal{V}, \theta), \mathcal{I}, \mathcal{O} \subseteq \mathcal{V}$, and a state function

$$T_1 : S_{\varsigma,\chi}^{\mathcal{I}}[\mathfrak{N}] \to S_{\varsigma,\chi}^{\mathcal{O}}[\mathfrak{N}].$$

a) If $\mathcal{O}' \supseteq \mathcal{O}$,

$$T_2 : S_{\varsigma,\chi}^{\mathcal{O}'}[\mathfrak{N}] \to S_{\varsigma,\chi}^{\mathcal{O}'}[\mathfrak{N}]$$

describes T_1 if and only if the coordinates in $\mathcal{O}$ of the fixed points of T_2 are given by the image of T_1, i. e. if and only if

$$\{s^{\mathcal{O}} : s \in \mathrm{FP}_{T_2}\} = \mathrm{im}(T_1).$$

b) A neural network $\tilde{\mathfrak{N}} := (\tilde{\mathcal{V}}, \tilde{\theta})$ with functional structure $(\tilde{\Lambda}, \tilde{f}, \tilde{g})$ such that

$$\mathcal{V} \subseteq \tilde{\mathcal{V}}, \quad \tilde{\theta}_{\mathcal{V}\to\mathcal{V}} = \theta,$$

is called *implementation* of T_1 if and only if its canonical step $T_{\tilde{\theta},\tilde{\lambda}}[\tilde{\mathfrak{N}}]$ describes T_1.

2.2.2 States in Feedforward and Additive Neural Networks

2.62 This subsection will clarify the rather general definitions of the previous paragraphs. More specifically, the canonical network function and step have a more concise form for feedforward and additive networks. The following subsection 2.2.3 will make the use of these definitions apparent as they allow us to talk more precisely about implementations.

2.63 Proposition. Consider a feedforward network $\mathfrak{N} = (\mathcal{V}, \theta)$ with functional structure (Λ, f, g) of depth L. For all $l = 0, \dots, L$, if $i \in S^{\leq \mathcal{V}:i}[\mathfrak{N}]$ is constant in the standardized layers $V^{(<l)}$,

$$N^{i}_{\theta,\lambda}(i) = \left(T^{L-l+1}_{\theta,\lambda}\right)^{:i}(\mathrm{Init}(i)), \quad N^{:o}_{\theta,\lambda}(i) = f\left(N^{i}_{\theta,\lambda}(i)\right). \tag{2.25}$$

In particular,

$$N^{i}_{\theta,\lambda} = \left(T^{L+1}_{\theta,\lambda}\right)^{:i} \circ \mathrm{Init}, \quad N^{:o}_{\theta,\lambda} = f \circ N^{i}_{\theta,\lambda}. \tag{2.26}$$

Proof. We prove the proposition by proving the statement

$$\left(T^{V^{(<l)}:i}_{\theta,\lambda}(i) = i^{V^{(<l)}}\right) \Rightarrow \left(\underbrace{\left(N^{i}_{\theta,\lambda}(i) = \left(T^{L-l+1}_{\theta,\lambda}(\mathrm{Init}(i))\right)^{:i}\right)}_{(1)} \wedge \underbrace{\left(N^{:o}_{\theta,\lambda}(i) = f\left(N^{i}_{\theta,\lambda}(i)\right)\right)}_{(2)} \right)$$

for any $i \in S^{\leq \mathcal{V}:i}[\mathfrak{N}]$ by induction on $L - l$, $l = L+1, \dots, 0$. Note that the statement is slightly more general than the proposition.
If $L - l + 1 = 0$, i. e. $l = L + 1$, (1) follows directly and (2) is obvious from the definition of N.
Next, suppose that the proposition is proven for $L - l = L - (l+1) + 1$ and consider some i such that

$$T^{V^{(<l)}}_{\theta,\lambda}(i) = i^{V^{(<l)}}.$$

In this case, define

$$s := T_{\theta,\lambda}(\mathrm{Init}(i)).$$

We observe that

$$s^{V^{(<l)}} = i^{V^{(<l)}}, \quad s^{V^{(l)}} = T^{V^{(l+1)}}_{\theta,\lambda}(\mathrm{Init}(i)) = T^{V^{(l+1)}}_{\theta,\lambda}(T_{\theta,\lambda}(\mathrm{Init}(i))) = T^{V^{(l+1)}}_{\theta,\lambda}(s),$$

as the network is feedforward and $T^{V^{(l)}}_{\theta,\lambda}$ therefore only depends on $i^{V^{(<l)}}$. This implies

$$T^{V^{(<l+1)}}_{\theta,\lambda}(s) = T^{V^{(\leq l)}}_{\theta,\lambda}(s) = s^{V^{(\leq l)}} = s^{V^{(<l+1)}},$$

and thus, by induction hypothesis,

$$N^{:i}_{\theta,\lambda}(i) = N^{:i}_{\theta,\lambda}(s) = \left(T^{L-(l+1)+1}_{\theta,\lambda}(\mathrm{Init}(s))\right)^{:i}.$$

As $s \in S[\mathfrak{N}]$, $\mathrm{Init}(s) = s$ and we therefore obtain

$$N^{i}_{\theta,\lambda}(i) = \left(T^{L-l}_{\theta,\lambda}(T_{\theta,\lambda}(\mathrm{Init}(i)))\right)^{:i} = \left(T^{L-l+1}_{\theta,\lambda}(\mathrm{Init}(i))\right)^{:i},$$

which concludes the induction and thus proves the proposition. □

2.64 In an additive neural network, the canonical step function is defined by

$$T_\theta = \theta^T f, \tag{2.27}$$

and therefore, the canonical network function is defined by

$$N_\theta = \lim_{n\to\infty} (\theta^T f)^n \circ \mathrm{Init}. \tag{2.28}$$

In particular, if L is the depth of the network, the canonical network function of an additive feedforward network is given by

$$N^{:i}_{\theta} = (\theta^T f)^{L+1} \circ \mathrm{Init}, \quad N^{:o}_{\theta} = f \circ N^{:i}_{\theta}.$$

2.2.3 States in Predictive Coding

2.65 (Predictive Coding II: The Free Energy Function) Recall the generative model from Definition 2.45. As it is impossible to infer the global parameter $\omega \in \Omega$, we may consider the alternative functional structure (Λ', f). However, we are now missing g and therefore the canonical step and the canonical network function. Furthermore, while we assume that the reality can be modeled by the generative model for some values of λ' and ξ, we do not know these values. Instead, our goal is to infer the states $s \in S[\mathfrak{G}]$ from some input $i \in S^{\mathcal{I}}[\mathfrak{G}]$ for given parameters, find a suitable implementation for this inference and improve estimation of λ' and ξ.
The inferred values should be as compatible as possible. We therefore posit a loss function

$$L_{\xi,\lambda'} : S[\mathfrak{G}] \to \mathbb{R}, \tag{2.29}$$

which we want to minimize. The knowledge we possess about how the states of the individual PUs should relate to each other is given by ξ and λ' which are treated as fixed parameters.
The input values $i \in S^{\mathcal{I}}[\mathfrak{G}]$ are already known and should therefore remain constant. The ideal network function would therefore return

$$\arg \min_{s \in S^{\mathcal{G}\setminus\mathcal{I}}[\mathfrak{G}]} L_{\xi,\lambda'}(s). \tag{2.30}$$

If we suppose a continuous random variable with density $p_{\xi,\lambda'}$ a sensible choice of L is the negative log-likelihood:

$$L_{\xi,\lambda'}(s) := -\ln p_{\xi,\lambda'}(s), \tag{2.31}$$

where, as we want $L_{\xi,\lambda'}$ to only depend on $s^{:i}$, we express $s^{:o}$ as $f(s^{:i})$.
The negative log-likelihood represents our surprise about the data. It is also called *free energy*.

2.66 (Convention: Differentiable model specifications) From now on, we will assume in the normal generative model that the mappings

$$\lambda' \mapsto \Sigma_{\lambda'}, \quad \mu, \quad f$$

are differentiable.

2.67 Proposition (Free Energy in the Normal Generative Model). If $L_{\xi,\lambda'}$ is the free energy function in the normal generative model,

$$F_{\xi,\lambda'}(s) = \frac{1}{2}\left(\ln\left(\det \Sigma_{\lambda'}\right) + \epsilon^T \Sigma_{\lambda'}^{-1} \epsilon\right), \tag{2.32}$$

where

$$\epsilon = s^{:i} - \mu_{\xi,\lambda'}\left(f(s^{:i})\right),$$

we obtain the following identities:

$$\arg\min_{s \in S[\mathfrak{G}]} F_{\xi,\lambda'}(s) = \arg\min_{s \in \mathbb{R}^{\mathcal{G}\setminus\mathcal{I}}} L_{\xi,\lambda'}(s), \qquad \frac{\partial}{\partial s} F_{\xi,\lambda'}(s) = \frac{\partial}{\partial s} L_{\xi,\lambda'}(s), \tag{2.33}$$

$$\arg\min_{\xi} F_{\xi,\lambda'}(s) = \arg\min_{\xi} L_{\xi,\lambda'}(s), \qquad \frac{\partial}{\partial \xi} F_{\xi,\lambda'}(s) = \frac{\partial}{\partial \xi} L_{\xi,\lambda'}(s), \tag{2.34}$$

$$\arg\min_{\lambda'} F_{\xi,\lambda'}(s) = \arg\min_{\lambda'} L_{\xi,\lambda'}(s), \qquad \frac{\partial}{\partial \lambda'} F_{\xi,\lambda'}(s) = \frac{\partial}{\partial \lambda'} L_{\xi,\lambda'}(s). \tag{2.35}$$

In other words: Minimizing L corresponds to minimizing F.

Proof. As we consider the normal generative model, ϵ is normally distributed:

$$\epsilon \sim \mathcal{N}(0, \Sigma_{\lambda'}).$$

Therefore,

$$p_{\xi,\lambda'}(s) = \frac{1}{\sqrt{(2\pi)^{|\mathcal{G}|} \det \Sigma_{\lambda'}}} \exp\left(-\frac{1}{2}\epsilon^T \Sigma^{-1} \epsilon\right).$$

Therefore,

$$L_{\xi,\lambda'}(s) = -\ln p_{\xi,\lambda'}(s) = \ln\left(\sqrt{(2\pi)^{|\mathcal{G}|}}\right) + \ln\left(\sqrt{\det \Sigma_{\lambda'}}\right) + \frac{1}{2}\epsilon^T \Sigma_{\lambda'}^{-1}\epsilon.$$

As the summand $\ln\left(\sqrt{(2\pi)^{|\mathcal{G}|}}\right)$ is constant in s, ξ and Σ, omitting it does not change the minimum or the derivative in these variables. Observing

$$\ln\left(\sqrt{\det \Sigma}\right) = \frac{1}{2}\left(\ln \det \Sigma\right)$$

yields F. As F and L only differ by a constant, their derivatives are identical. □

2.68 (Predictive Coding III: Inference) Generally, there is not a good method to find a closed form of the global minimum of L. Instead, we propose optimization by gradient descent, such that a single step is given by $\varphi_{\xi,\lambda'|\alpha} \equiv \left(\phi_{\xi,\lambda'|\alpha}\right)_{\varsigma,\chi}$ where

$$\phi_{\xi,\lambda'|\alpha} : S[\mathfrak{G}] \to S[\mathfrak{G}], \quad \phi^{i}_{\xi,\lambda'|\alpha}(s) := s - \alpha\left(\frac{\partial L_{\xi,\lambda'}(s)}{\partial s}\right)^T, \quad \phi^{o}_{\xi,\lambda'|\alpha}(s) := f\left(\phi^{i}_{\xi,\lambda'|\alpha}(s)\right). \tag{2.36}$$

The network function is therefore given by iterative gradient descent starting at a:

$$\Phi_{\xi,\lambda'|\alpha,a} := \lim_{n\to\infty} \phi^n_{\xi,\lambda'|\alpha} \circ \mathrm{Init}_a. \tag{2.37}$$

Choosing α and a as fixed parameters, we call an implementation of $\Phi_{\xi,\lambda'}$ a *predictive coding network*.

2.69 Remark. As optimization by gradient descent is an entire field on its own, we will mostly restrict ourselves to deriving formulas under the condition that gradient descent converges. It is a distinct – albeit important – task to find conditions for convergence of gradient descent.

2.70 Notation (Derivatives of and in Vectors and Matrices). Given functions

$$f : \mathbb{R}^m \to \mathbb{R}^n, \quad M : \mathbb{R} \to \mathbb{R}^{m\times n}, \quad y : \mathbb{R}^{m\times n} \to \mathbb{R},$$

we notate the derivatives by

$$\frac{\partial f}{\partial x} = \left(\frac{\partial f_i}{\partial x_j}\right)_{\substack{i=1,\dots,m\\ j=1,\dots,n}}, \quad \frac{\partial M}{\partial x} = \left(\frac{\partial M_{ij}}{\partial x}\right)_{\substack{i=1,\dots,m\\ j=1,\dots,n}}, \quad \frac{\partial y}{\partial A} = \left(\frac{\partial y}{\partial A_{ji}}\right)_{\substack{i=1,\dots,n\\ j=1,\dots,m}}.$$

Appendix A.1 derives a few important matrix derivatives.

2.71 Lemma. In a normal generative model, if μ is differentiable, the step is given by

$$\phi_{\xi,\Sigma|\alpha}(s) := s - \alpha\Sigma^{-1}\epsilon + \alpha\left(\frac{\partial}{\partial s}\mu_\xi(s)\right)^T \Sigma^{-1}\epsilon \in S[\mathfrak{G}]. \tag{2.38}$$

Proof. According to the chain rule,

$$\frac{\partial F}{\partial s} = \frac{\partial F}{\partial \epsilon}\frac{\partial \epsilon}{\partial s},$$

Lemma A.1 yields

$$\frac{\partial F}{\partial \epsilon} = \frac{\partial \frac{1}{2}\epsilon^T\Sigma^{-1}\epsilon}{\partial \epsilon} = \epsilon^T\Sigma^{-1}.$$

On the other hand,

$$\frac{\partial \epsilon}{\partial s} = \frac{\partial}{\partial s}(s - \mu_\xi(s)) = I_{\mathcal{G}\to\mathcal{G}} - \frac{\partial \mu_\xi(s)}{\partial s}.$$

Putting both together yields

$$\frac{\partial F}{\partial s} = \epsilon^T\Sigma^{-1} - \left(\left(\frac{\partial}{\partial s}\mu_\xi(s)\right)^T \Sigma^{-1}\epsilon\right)^T,$$

which, if filled into (2.36), yields (2.38), as Σ^{-1} is symmetric. □

2.72 From now on, we will specialize in additive networks as in the general case, many of the equations have a rather unintuitive implementation whose notation seems forced.

2.73 Corollary. In a normal additive generative model, the step is given by

$$\phi_{\xi,\Sigma|\alpha}(s) := s - \alpha\Sigma^{-1}\epsilon + \alpha f'(s)\xi\Sigma^{-1}\epsilon, \tag{2.39}$$

where

$$f' := \frac{\partial}{\partial s} f. \tag{2.40}$$

Proof. By the definition of additive networks,

$$\mu_\xi(s) = \xi^T f(s),$$

which implies

$$\frac{\partial}{\partial s}\mu_\xi(s) = \xi^T \frac{\partial}{\partial s} f(s) = \xi^T f'(s).$$

The corollary now follows directly from Lemma 2.71 and the fact that $f'(s)$ is diagonal and therefore symmetric. □

2.74 Theorem (Additive Normal Predictive Coding Network). Gradient descent in a normal additive generative model $\mathfrak{G} = (\mathcal{G}, \xi)$ with a functional structure $(\Lambda' \times \Omega, \tilde{f})$ can be implemented by a neural network $\mathfrak{P} = (\mathcal{P}, \theta)$ where

$$\begin{aligned} &\mathcal{P} = \mathcal{G} \cup \varepsilon, \quad \varepsilon = \{\varepsilon^v | v \in \mathcal{G}\} \\ &\theta_{v\to w} := \begin{cases} (\delta_{v,w}, \varsigma) & \text{if } v, w \in \mathcal{G}, \\ (\delta_{v,u}, \varsigma) - \xi_{v\to u} & \text{if } v \in \mathcal{G}, w = \varepsilon^u \in \varepsilon, \\ \alpha\left(-(\delta_{u,w}, \varsigma) + \xi^T_{u\to w}\right) & \text{if } v = \varepsilon^u \in \varepsilon, w \in \mathcal{G}, \\ (\delta_{u,x}, \varsigma) - \left(\Sigma^T_{\lambda'}\right)_{u\to x} & \text{if } v = \varepsilon^u, w = \varepsilon^x \in \varepsilon. \end{cases} \end{aligned} \tag{2.41}$$

Identifying the entries in $\mathcal{G}$ and ε, we may also use a shorthand matrix notation:

$$\theta = \begin{pmatrix} \theta_{\mathcal{G}\to\mathcal{G}} & \theta_{\mathcal{G}\to\varepsilon} \\ \theta_{\varepsilon\to\mathcal{G}} & \theta_{\varepsilon\to\varepsilon} \end{pmatrix} = \begin{pmatrix} I_\varsigma & I_\varsigma - \xi \\ -\alpha I_\varsigma + \alpha\xi^T & (I - \Sigma)^T \end{pmatrix}.$$

The functional structure of $\mathfrak{P}$ is given by (f, g), where

$$\begin{aligned} f := (f^v)_{v\in\mathcal{P}}, \quad f^v := \begin{cases} \tilde{f}^v \text{ if } v \in \mathcal{G}, \\ \mathrm{Id} \text{ else}, \end{cases}, \quad g_\theta : S[\mathfrak{P}] \to S^{:i}[\mathfrak{P}], \\ s \mapsto s^{v:i} := g^v_\theta(s) := \begin{cases} s^{v:i} + \theta_{\varepsilon^v\to v} s^{\varepsilon^v:o} + (f^v)'(s^{v:i})\theta^T_{\varepsilon\setminus\{\varepsilon^v\}\to v} s^{\varepsilon\setminus\{\varepsilon^v\}:o} & \text{if } v \in \mathcal{G}, \\ \theta^T_{\mathcal{P}\setminus\{u\}\to v} s^{\mathcal{P}\setminus\{u\}:o} + \theta_{u\to v} s^{v:i} & \text{if } v = \varepsilon^u \in \varepsilon. \end{cases} \end{aligned} \tag{2.42}$$

We refer to ε as the *error units* and $\mathcal{G}$ as the *generative units*. We call $\mathfrak{P}$ the predictive coding network on $\mathfrak{G}$.

2.75 Evidently, the additive normal predictive coding network is not a general additive network. However, the only reason for this is that the input from the other error units is weighted by the derivative of the generative unit function f. (Note that the neural network is well-defined as there is an auto-connection $v \to v$ that is constantly non-zero.) It is possible to implement gradient descent in a general additive network by adding derivative nodes and setting ln and exp as suitable unit functions. As there is no apparent use for such a specification, we will refrain from such a definition.

Proof (of Theorem 2.74). As mentioned, $\mathfrak{P}$ is indeed a neural network. The additive dependency structure makes that immediately visible if we note that $(f^v)'(s^{v:i})$ is known due to the connection $v \to v$.
In order to show that $\mathfrak{P}$ implements the gradient function, we have to characterize the fixed points of the canonical step function given by the equation

$$g_\theta(s) = s^{:i}, \quad f(s^{:i}) = s^{:o}.$$

We first consider the error units $\varepsilon^v \in \varepsilon$:

$$s^{\varepsilon^v} = s^{\varepsilon^v:i} = g_\theta(s^{\varepsilon^v}) = \theta^T_{\mathcal{P}\setminus\{v\}\to\varepsilon^v} s^{\mathcal{P}\setminus\{v\}:o} + \theta_{v\to\varepsilon^v} s^{\varepsilon^v:i} \overset{(*)}{=}$$
$$-\xi^T s^{\mathcal{G}\setminus\{v\}:o} + s^{v:i} + (I-\Sigma)s^{\varepsilon:o} = \epsilon^v + (I-\Sigma)s^{\varepsilon:o} = \epsilon^v + (I-\Sigma)s^{\varepsilon},$$

and therefore

$$s^\varepsilon = \Sigma^{-1}\epsilon.$$

Considering the generative units $v \in \mathcal{G}$ and using this equation, we can now conclude

$$s^{v:i} = s^{v:i} + \theta_{\varepsilon^v\to v} s^{\varepsilon^v:o} + (f^v)'(s^{v:i})\theta^T_{\varepsilon\setminus\{\varepsilon^v\}\to v} s^{\varepsilon\setminus\{\varepsilon^v\}:o} = s^{v:i} - \alpha s^{\varepsilon^v} + \alpha\,(f^v)'(s^{v:i})\xi_{v\to} s^\varepsilon.$$

where we have again used that the diagonal of ξ is empty. Vectorizing the equation, we observe

$$s^{:i} = s^{:i} - \alpha\Sigma^{-1}\epsilon + \alpha\left(\frac{\partial}{\partial s^{:i}} f(s^{:i})\right)\xi\Sigma^{-1}\epsilon.$$

This corresponds to the step function $\phi_{\xi,\Sigma|\alpha}$. As $\tilde{f}$ is continuous, ϕ is continuous. As $\mathbb{R}$ is closed, the value Φ converges to are the fixed point of ϕ which are the $\mathcal{G}$-coordinates of the fixed points of the canonical step functions. If we are given an element of the image of Φ, we may simply set $s^\varepsilon := \Sigma^{-1}\epsilon$ which yields a fixed point of the canonical step function $T[\mathfrak{P}]$ of $\mathfrak{P}$. We have thus proven the theorem.□

2.76 Lemma (Recursive Error Formula). Consider some balanced state $s := N(i)$ where $i \in S^{\mathcal{I}:i}[\mathfrak{P}], \mathcal{I} \subseteq \mathcal{G}$. For any $v \in \mathcal{G}$, if i^v is not constant,

$$s^{\varepsilon^v} = (f^v)'(s^{v:i})\xi_{v\to(v\to)} s^{\varepsilon^{v\to}}.$$

Proof. If i^v is not constant, s^v is not constant and therefore, the fixed point equation yields

$$s^v = s^v - \alpha s^{\varepsilon^v} + \alpha\,(f^v)'(s^{v:i})\xi_{v\to} s^\varepsilon.$$

For $w \notin v\to$, $\xi_{v\to w} = 0$ which implies the lemma. □

2.77 Lemma. Consider an additive normal predictive coding network $\mathfrak{P} = (\mathcal{G}\cup\varepsilon, \theta)$ with the functional structure (f, g). If the input is $i \in S^{V^{(0)}}_{\varsigma,\chi}[\mathfrak{P}]$ and $N(i)$ exists, the state of its canonical network function $s := N^{\mathcal{G}}(i)$ may be defined recursively in the standardized layerization $G^{(0)} \prec G^{(1)} \prec \cdots \prec G^{(L)}$ by

$$s^{v:i} := \begin{cases} \Sigma_{v\to G^{(0)}} s^{G^{(0)}:i} & \text{if } v \in G^{(0)} \wedge \mathrm{St}(i^v) = \chi, \\ i^v & \text{if } v \in G^{(0)} \wedge \mathrm{St}(i^v) = \varsigma, \\ \xi^T s^{:o} + \Sigma_{v\to G^{(0)}} s^{G^{(0)}:i} & \text{if } v \notin G^{(0)}, \end{cases} \tag{2.43}$$
$$s^{v:o} := f(s^{v:i}).$$

In particular, if

$$\bigvee_{v\in G^{(0)}} \bigvee_{w\in\mathcal{G}\setminus G^{(0)}} \Sigma_{v\to w} = 0,$$

i. e. if the input nodes are independent from all other nodes and i is constant,

$$s^{:i} := \left(\xi^T f\right)^L(\mathrm{Init}(i)), \quad s^{:o} := f(s^{:i}). \tag{2.44}$$

Proof. Define $\tilde{s} := N(i)$. Evidently, $\tilde{s}^{\mathcal{G}} = s$ and $\tilde{s} \in B[\mathfrak{P}]$. Consider $v \in G^{(0)}$. If $\mathrm{St}(i^v) = \varsigma$, $\tilde{s}^{v:i} = i^v$ as N is an active extension. If $\mathrm{St}(i^v) = \chi$ as well as if $v \in \mathcal{G} \setminus G^{(0)}$, Lemma 2.76 implies

$$\tilde{s}^{\varepsilon^v} = (f^v)'(s^{v:i})\xi_{v\to}\tilde{s}^{\varepsilon}.$$

We now show by induction on $\leftarrow$ that if $\mathrm{St}(i^v) = \chi$ or $v \in \mathcal{G} \setminus G^{(0)}$,

$$s^{\varepsilon^v} = 0.$$

Because ξ is feedforward and thus an upper triangular matrix where

$$\xi_{v\to G^{(L)}} = 0,$$

we obtain, for all $v \in G^{(L)}$,

$$\tilde{s}^{\varepsilon^v} = 0,$$

which proves the base case. On the other hand, if

$$\tilde{s}^{\varepsilon^{v\to}} = 0,$$

we can conclude that

$$\tilde{s}^{\varepsilon^v} = (f^v)'(s^{v:i})\xi_{v\to(v\to)}\tilde{s}^{\varepsilon^{v\to}} = 0,$$

which proves the induction step.
As $\tilde{s}$ is balanced, we can also infer that

$$\tilde{s}^{\varepsilon} = \Sigma^{-1}(s^{:i} - \xi^T s^{:o}),$$

and thus

$$s^{:i} = \Sigma\tilde{s}^{\varepsilon} + \xi^T s^{:o}.$$

As $\tilde{s}^{\varepsilon^{\mathcal{G}\setminus G^{(0)}}} = 0$ and $\xi^T_{\to G^{(0)}} = 0$, this implies (2.43). □

2.78 Lemma 2.77 demonstrates that a common use of the predictive coding network where $V^{(0)}$ is the input layer and the input layer does not possess any correlation structure with the succeeding nodes[4] can be characterized by a standard additive neural network $(\mathcal{G}, \xi)$ with functional structure f.

2.3 Learning in Neural Networks

2.79 A crucial property of neural networks is their ability to change their parameters according to training data; to learn. Learning is introduced in subsection 2.3.1. The backpropagation algorithm that implements gradient descent in feedforward neural networks is introduced in 2.3.2. Subsection 2.3.3 discusses biologically plausible learning and introduces the learning algorithm of a normal additive predictive coding network. The final subsection 2.3.4 provides some remarks on the relation between backpropagation and the predictive coding algorithm.

2.3.1 Learning Functions

2.80 Definition (Learning Structure). Consider a neural network $\mathfrak{N} = (\mathcal{V}, \theta)$ with functional structure (f, g, Λ). Let $\mathcal{I}, \mathcal{O} \subseteq \mathcal{V} \times \{i, o\}$ be disjoint. The PUs $\mathcal{I}$ correspond to the input and $\mathcal{O}$ corresponds to the output that shall be approximated. The function

$$\tilde{\mathcal{L}} : \left(S^{\mathcal{I}}[\mathfrak{N}] \times S^{\mathcal{O}}[\mathfrak{N}]\right)^{<\mathbb{N}} \times A^{\mathcal{V}\times\mathcal{V}}_{\varsigma,\chi} \times \Lambda_{\varsigma,\chi} \to A^{\mathcal{V}\times\mathcal{V}}_{\varsigma,\chi} \times \Lambda_{\varsigma,\chi}$$

defines by its active extension in $A^{\mathcal{V}\times\mathcal{V}}_{\varsigma,\chi} \times \Lambda_{\varsigma,\chi}$ a *learning function*

$$\mathcal{L} := \tilde{\mathcal{L}}_{\varsigma,\chi:A^{\mathcal{V}\times\mathcal{V}}_{\varsigma,\chi}\times\Lambda_{\varsigma,\chi}}, \quad (((i_1, o_1), \ldots, (i_n, o_n)), \theta, \lambda) \mapsto \mathcal{L}^{i;o}(\theta, \lambda) = (C^{i;o}(\theta, \lambda), G^{i;o}(\theta, \lambda)),$$

[4] Note that this is not a restriction as such a structure can be covered by a suitable ξ.

where

$$C^{i;o}(\theta,\lambda) \in A^{\mathcal{V}\times\mathcal{V}}$$

returns the actualized θ and

$$G^{i;o}(\theta,\lambda) \in \Lambda$$

returns the actualized global parameters.

2.81 Remark. $\left(S^{\mathcal{I}}[\mathfrak{N}] \times S^{\mathcal{O}}[\mathfrak{N}]\right)^{<\mathbb{N}}$ defines the set of finite training examples $((i_1,o_1),\dots,(i_n,o_n))$ for an arbitrary $n \in \mathbb{N}$.

2.82 (Learning by Gradient Descent) An important class of learning functions is given by gradient descent functions which are induced by a loss function. Loss functions are defined for some $\mathcal{O} \subseteq \mathcal{V} \times \{i,o\}$ by

$$L : \left(S^{\mathcal{O}}[\mathfrak{N}] \times S^{\mathcal{O}}[\mathfrak{N}]\right)^{<\mathbb{N}} \to \mathbb{R}, \quad ((\hat{o}_1,o_1),\dots,(\hat{o}_n,o_n)) \mapsto L\left(\hat{o},o\right). \tag{2.45}$$

The gradient descent is given by

$$\tilde{\mathcal{L}}^{i;o}_{\alpha}(\theta,\lambda) := \left(\theta - \alpha\left(\frac{\partial L\left(N_{\theta,\lambda}(i);o\right)}{\partial\theta}\right)^T, \lambda - \alpha\left(\frac{\partial L\left(N_{\theta,\lambda}(i);o\right)}{\partial\lambda}\right)^T\right)$$

such that the gradient descent learning function is defined by its active expansion in $\mathbb{R}^{\mathcal{V}\times\mathcal{V}}$:

$$\mathcal{L}_{\alpha} := \left(\tilde{\mathcal{L}}_{\alpha}\right)_{\varsigma,\chi:\mathbb{R}^{\mathcal{V}\times\mathcal{V}}} \quad \alpha > 0. \tag{2.46}$$

$\mathcal{L}_{\alpha}$ is *induced* by L.

2.83 We will mostly consider a loss function for a single training example. In many cases, an extension towards several training examples can be given by

$$L(\hat{o},o) := \sum_{i=1}^{n} L(\hat{o}_i,o_i).$$

Alternatively, we may extend a learning function $\mathcal{L}$ defined on a single training example by defining

$$\mathcal{L}^{i_1,\dots,i_n;o_1,\dots,o_n}_{\theta,\lambda} := \mathcal{L}^{i_n;o_n}\left(\mathcal{L}^{i_{n-1};o_{n-1}}\left(\cdots\mathcal{L}^{i_1;o_1}(\theta,\lambda)\cdots\right)\right) \tag{2.47}$$

although without additional precautions the former approach is more useful (see 3.5 and Example 3.6).

2.3.2 Backpropagation

2.84 Many feedforward neural networks are purely concerned with prediction: the lowest layer $V^{(0)}$ represents the input and the highest layer $V^{(L)}$ represents the output that is predicted. In this case, we notate

$$O := N^{V^{(L)};o}. \tag{2.48}$$

If i is the input and o is the actual result, we now try to optimize some loss function

$$L\left(O(i),o\right).$$

A popular choice for L is the squared error. The backpropagation algorithm can compute the gradient descent learning function and is especially suited for additive feedforward networks. We therefore present backpropagation in this framework below.

2.85 THEOREM (BACKPROPAGATION). Consider an additive feedforward network $\mathfrak{N} = (\mathcal{V}, \theta)$ with a differentiable functional structure f and the loss function

$$L : \mathbb{R}^{V^{(L)}} \times \mathbb{R}^{V^{(L)}} \to \mathbb{R}. \tag{2.49}$$

If $i \in V^{(0)}$ is constant, the gradient can then be computed by

$$\frac{\partial L(O_\theta(i), o)}{\partial \theta_{v\to w}} = \delta^w N^{v:o}(i), \tag{2.50}$$

where

$$\delta^T := \frac{\partial L}{\partial N_\theta^{v:i}(i)}$$

is recursively defined by

$$\begin{aligned} v \in V^{(L)} &\Rightarrow \delta^v = (f^v)' \left(N_\theta^{v:i}(i)\right) \frac{\partial L_\theta(O_\theta(i), o)}{\partial O_\theta^v(i)}, \\ v \notin V^{(L)} &\Rightarrow \delta^v = (f^v)' \left(N_\theta^{v:i}(i)\right) \delta^{v\to} \theta_{v\to(\to v)}. \end{aligned} \tag{2.51}$$

We define the learning function $\mathcal{B}_\alpha$ that is induced by L.

Proof. By

$$\frac{\partial L(O_\theta(i), o)}{\partial \theta_{v\to w}} = \frac{\partial L(O_\theta(i), o)}{\partial N_\theta^{w:i}(i)} \frac{\partial N_\theta^{w:i}(i)}{\partial \theta_{v\to w}},$$

we only have to note that

$$\frac{\partial}{\partial \theta_{v\to w}} N^{w:i}(i) = \frac{\partial}{\partial \theta_{v\to w}} \theta_{\to w}^T N^{:o}(i) = N^{v:o}(i)$$

to get (2.50).
We therefore prove that (2.51) defines δ by induction. If $v \in V^{(L)}$, i. e. $v \to = \emptyset$,

$$N_\theta^{v:i}(i) = O_\theta^v(i),$$

and (2.51) is therefore obvious. On the other hand, if we already know

$$\delta^{v\to} = \frac{\partial L(O_\theta(i), i)}{\partial N_\theta^{v\to:i}(i)},$$

we may write

$$\delta^v = \delta^{v\to} \frac{\partial N_\theta^{v\to:i}(i)}{\partial N_\theta^{v:o}(i)} \frac{\partial N_\theta^{v:o}(i)}{\partial N_\theta^{v:i}(i)}.$$

As

$$N_\theta^w(i) = \theta_{\to w}^T N_\theta(i),$$

and for any $u \in \to w$, N^u does not depend on N^v (as the network is straight),

$$\frac{\partial N_\theta^{v\to:i}(i)}{\partial N_\theta^{v:o}(i)} = \theta_{v\to(\to w)},$$

which, combined with the upper equation and the observation that

$$\frac{\partial N_\theta^{v:o}(i)}{\partial N_\theta^{v:i}(i)} = (f^v)' \left(N_\theta^{v:i}(i)\right),$$

yields (2.51). □

2.86 The method that the proof of the theorem sketches works for any feedforward network. However, in the case of non-additive networks, it is not very enlightening.

2.3.3 Learning in Predictive Coding

2.87 (Predictive Coding IV: Learning Perspectives) In the previous paragraphs on predictive coding, we have constructed the generative model $\mathfrak{G}$, its parameters ξ and λ' (resp. $\Sigma_{\lambda'}$) and inference in $\mathfrak{G}$. Then came the implementation of inference: the predictive coding model $\mathfrak{P}$ and its parameters θ. As θ depends on ξ and $\Sigma_{\lambda'}$, we have two options with respect to parameter learning: we may either learn ξ and $\Sigma_{\lambda'}$ and thereby θ or we may learn θ.
We will consider learning ξ and λ'. The approach we take is based on the expectation maximization algorithm presented by Dempster et al. [12] as implemented by Friston [7] and Whittington and Bogacz [8]: we want to minimize the free energy function in ξ and λ' for a balanced state $s \in B[\mathfrak{P}]$. We are therefore looking for the learning function $\mathcal{L}_\alpha[\mathfrak{P}]$ induced by L. The difference to backpropagation and most other learning algorithms is that we do not compare the real outcome o to the outcome according to $N[\mathfrak{P}](i)$. Rather, we obtain a balanced state for input and output together. We therefore often consider $i \in S^{\mathcal{V}^{(0,L)}:i}[\mathfrak{P}]$ (although any kind of input may be used) and then set

$$s := N_{\xi,\lambda'}[\mathfrak{P}](i).$$

Now, we try to reduce the free energy function $L_{\theta(\xi,\lambda')}(s)$[5] in ξ and λ'. We therefore do not try to change the parameters such that $N[\mathfrak{P}]_{\theta(\xi,\lambda')}(i)$ produces more compatible results. Rather, we obtain estimates of our values and then update ξ and λ' to make these estimates more compatible.
In order to do that, we need the gradient of $L_{\theta(\xi,\lambda')}(s)$ for balanced states in ξ and λ' which the following two theorems provide for additive normal generative models.

2.88 Theorem (Learning ξ). Consider an additive normal generative model $\mathfrak{G} = (\mathcal{G}, \xi)$ with functional structure (Λ, f). If $L_{\xi,\lambda'}$ is the free energy function in $\mathfrak{G}$ and s is balanced, we obtain

$$\frac{\partial L_{\xi,\lambda'}(s)}{\partial \xi} = -s^\varepsilon \left(s^{:o}\right)^T$$

Proof. As we have observed in the proof of Theorem 2.74,

$$s^\varepsilon = \Sigma^{-1}\epsilon.$$

According to Proposition 2.67,

$$\frac{\partial L_{\xi,\lambda'}(s)}{\partial \xi_{v\to w}} = \frac{\partial F_{\xi,\lambda'}(s)}{\partial \xi_{v\to w}}.$$

Lemma A.1 implies that

$$\frac{\partial F_{\xi,\lambda'}(s)}{\partial \epsilon} = \left(\Sigma^{-1}\epsilon\right)^T = \left(s^\varepsilon\right)^T.$$

On the other hand,

$$\frac{\partial \epsilon^u}{\partial \xi_{v\to w}} = \begin{cases} -s^{v:o} & \text{if } u = w, \\ 0 & \text{else,} \end{cases}$$

which implies that

$$\frac{\partial F_{\xi,\lambda'}(s)}{\partial \xi_{v\to w}} = \frac{\partial F_{\xi,\lambda'}(s)}{\partial \epsilon}\frac{\partial \epsilon}{\partial \xi_{v\to w}} = -s^{\varepsilon^w} s^{v:o},$$

and thus proves the theorem. □

2.89 Theorem (Learning λ'). Consider an additive normal generative model $\mathfrak{G} = (\mathcal{G}, \xi)$ with functional structure (Λ, f) where $\Lambda \subseteq \mathbb{R}^I$ is open (where I is some finite index set). If $L_{\xi,\lambda'}$ is the free energy function in $\mathfrak{G}$ and s is balanced, we obtain

$$\frac{\partial L_{\xi,\lambda'}(s)}{\partial \lambda'_i} = \frac{1}{2}\operatorname{tr}\left(\left(\Sigma_{\lambda'}^{-1} - \left(s^\varepsilon\right)\left(s^\varepsilon\right)^T\right)\frac{\partial \Sigma_{\lambda'}}{\partial \lambda'_i}\right). \tag{2.52}$$

In particular, if we are given Σ as a global parameter, i. e. $\lambda' = \Sigma$,

$$\frac{\partial L_{\xi,\Sigma}(s)}{\partial \Sigma} = \frac{1}{2}\left(\Sigma^{-1} - \left(s^\varepsilon\right)\left(s^\varepsilon\right)^T\right). \tag{2.53}$$

[5] I have written $\theta(\xi, \lambda')$ to make the dependence clearer.

Proof. We begin by evaluating

$$\frac{\partial}{\partial \lambda'_i} \ln \det \Sigma_{\lambda'}$$

for some $i \in I$. As $\Sigma_{\lambda'}$ is invertible for every λ', Lemma A.6 implies

$$\frac{\partial}{\partial \lambda'_i} \ln \det \Sigma_{\lambda'} = \frac{1}{\det \Sigma_{\lambda'}} \frac{\partial}{\partial \lambda'_i} \det \Sigma_{\lambda'} = \operatorname{tr}\left(\Sigma_{\lambda'}^{-1} \frac{\partial}{\partial \lambda'_i} \Sigma_{\lambda'}\right).$$

On the other hand,

$$\begin{aligned}
&\frac{\partial}{\partial \lambda'_i} \epsilon^T \Sigma_{\lambda'}^{-1} \epsilon \overset{\text{Lemma A.5}}{=} \operatorname{tr}\left(\frac{\partial \epsilon^T \Sigma_{\lambda'}^{-1} \epsilon}{\Sigma_{\lambda'}^{-1}} \frac{\partial \Sigma_{\lambda'}^{-1}}{\partial \lambda'_i}\right) \overset{\substack{\text{Lemma A.2}\\ \text{Lemma A.4}}}{=} \\
&\operatorname{tr}\left(\epsilon\epsilon^T \cdot (-1) \cdot \Sigma_{\lambda'}^{-1} \frac{\partial \Sigma_{\lambda'}}{\partial \lambda'_i} \Sigma_{\lambda'}^{-1}\right) \overset{s \in B[\mathfrak{P}]}{=} -\operatorname{tr}\left(\Sigma_{\lambda'} s^\varepsilon \left(s^\varepsilon\right)^T \frac{\partial \Sigma_{\lambda'}}{\partial \lambda'_i} \Sigma_{\lambda'}^{-1}\right) \overset{\operatorname{tr}(AB) = \operatorname{tr}(BA)}{=} \\
&-\operatorname{tr}\left(\Sigma_{\lambda'}^{-1} \Sigma_{\lambda'} s^\varepsilon \left(s^\varepsilon\right)^T \frac{\partial \Sigma_{\lambda'}}{\partial \lambda'_i}\right) = -\operatorname{tr}\left(s^\varepsilon \left(s^\varepsilon\right)^T \frac{\partial \Sigma_{\lambda'}}{\partial \lambda'_i}\right).
\end{aligned}$$

The fact that $\operatorname{tr}(A) + \operatorname{tr}(B) = \operatorname{tr}(A+B)$ and Proposition 2.67 proves (2.52).
On the other hand, if $\lambda' = \Sigma_{\lambda'}$, we have

$$\frac{\partial}{\partial \Sigma_{w \to v}} L_{\xi,\Sigma}(s) = \frac{1}{2} \operatorname{tr}\left(\underbrace{\left(\Sigma^{-1} - \left(s^\varepsilon\right)\left(s^\varepsilon\right)^T\right)}_{=:A} \underbrace{\frac{\partial \Sigma}{\partial \Sigma_{w \to v}}}_{=:B}\right),$$

where

$$B_{u \to u'} = \begin{cases} 1 & \text{if } u = w \wedge u' = v, \\ 0 & \text{else.} \end{cases}$$

Therefore,

$$\operatorname{tr}(AB) = \sum_{v \in \mathcal{G}} (AB)_{v \to v} = \sum_{u' \in \mathcal{G}} \sum_{u \in \mathcal{G}} A_{u' \to u} B_{u \to u'} = A_{v \to w},$$

which implies

$$\frac{\partial}{\partial \Sigma_{w \to v}} L_{\xi,\Sigma}(s) = \frac{1}{2} \left(\Sigma_{\lambda'}^{-1} - \left(s^\varepsilon\right)\left(s^\varepsilon\right)^T\right)_{v \to w},$$

and thus proves (2.53). □

2.90 (Biological Plausibility I: The Hebbian Postulate) Why do we consider the more complicated predictive coding model instead of relying on a simple feedforward model with backpropagation? On the one hand, a predictive coding model promises a more flexible model specification (see chapter 3), on the other hand, its definition is biologically motivated. In 1949, Donald Hebb, a Canadian psychologist, first presented his postulate on learning:

> When an axon of cell A is near enough to excite a cell B and repeatedly or presistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased. [13, p. 62]

This means that the correlated activity of two neurons strengthens their synaptic connection. The mathematical formulation of Hebbian learning (see [14]) includes several considerations. I have decided on the following two definitions. I will continue with some remarks on the relationship between backpropagation and predictive coding and their biological plausibility, i. e. whether these approaches are local resp. Hebbian.

2.91 Definition (Local and Hebbian Learning). A learning function $\mathcal{L}$ is *local* if and only if, $C^{i;o}_{v\to w}$ only depends on $N^v(i), N^w(i)$ and $\theta_{v\to w}$, i. e. for any $v, w \in \mathcal{V}$, we may write

$$C^{i;o}_{v\to w} = \tilde{C}_{v\to w}\left(N^v(i), N^w(i), \theta_{v\to w}\right)$$

for some function $\tilde{C}_{v\to w}$.
A local learning function is *Hebbian* if and only if, for every $v, w \in \mathcal{V}$, $C^i_{v\to w}$ is monotonically increasing in $N^{v:o}(i) \cdot N^{w:i}(i)$. This product is intended to capture the correlation between the activities of the two neurons.

2.92 Remark. Whereas a learning function is local as soon as it only uses the three locally available values $N^v(i), N^w(i), \theta_{v\to w}$, Hebbianity requires that higher covariance yields a higher connection weight. It is connected to the definition of Hebbianity in [14, p. 405].

2.3.4 Backpropagation and Predictive Coding

2.93 (Biological Plausibility II: A Comparison) Clearly, backpropagation is not local in all but the most simple examples. This is the reason why "for most of the past three decades since the invention of [backpropagation], it was generally believed that it could not be implemented by the brain" [5, p. 1837]. In particular, this is the reason why a simple feedforward network is most certainly not an implementation of backpropagation.
We will now be concerned with two things in this subsection: firstly, is the predictive coding model Hebbian? Secondly, is the predictive coding model an implementation of backpropagation? The first question will be answered in this paragraph and the second question will be answered by Proposition 2.95.
Indeed, the learning rule on ξ, as given by Theorem 2.88, is Hebbian. If we apply the learning rule to θ it is still local (but not Hebbian):

$$\theta_{v\to\varepsilon^w} \mapsto \theta_{v\to\varepsilon^w} - \alpha N^{\varepsilon^w:i}(i)N^{v:o}(i), \quad \theta_{\varepsilon^w\to v} \mapsto \theta_{\varepsilon^w\to v} + \alpha N^{\varepsilon^w:i}(i)N^{v:o}(i).$$

A challenge to its biological plausibility comes from the fact that the predictive coding model makes symmetric feedback and feedforward weights necessary and "there is no evidence for such a strong symmetry in cortex" [8, p. 1254]. However, Whittington and Bogacz remark that "preliminary simulations suggest that symmetric weights are not necessary for good performance of predictive coding network" [8, p. 1254] and refer to a forthcoming paper. Interestingly, an investigation by Liao et al. [5] provides similar results in the context of error backpropagation.
On the other hand, the locality of the learning rule on λ', as given by Theorem 2.89, strongly depends on how $\Sigma_{\lambda'}$ is specified. In the case of $\Sigma = \lambda'$, though, it is clearly not local unless we only allow diagonal covariance matrices as Σ needs to be inverted. Such a complicated computation poses a challenge for biological plausibility although Bogacz [15] presents an alternative implementation of predictive coding with a local learning rule.
In conclusion, while some problems still need to be reconciled, the predictive coding model is biologically more plausible than the backpropagation algorithm. This is not only supported by satisfying criteria like the Hebbian postulate but also by empirical evidence: the predictive coding model is based on a model presented by Rao and Ballard [6] which they successfully employed to explain observations in visual processing neurons.

2.94 For a given set of states, the recursive error formula for predictive coding that is given in Lemma 2.76 is clearly equivalent to error backpropagation in the corresponding feedforward network $\mathfrak{N}$ (Proposition 2.95 provides a proper definition of $\mathfrak{N}$). However the states $s^{\mathcal{G}}$ may be different which is why the predictive coding network does not necessarily implement backpropagation. Note that if $i \in S^{G^{(0)}:i}[\mathfrak{P}]$, we may conclude $s^{\varepsilon^{\mathcal{G}}\setminus G^{(0)}} = \delta^{\mathcal{G}\setminus G^{(0)}} = 0$ and the weight change is zero in both cases. In the standard case $i \in S^{G^{(0,L)}:i}[\mathfrak{P}]$, we do not necessarily obtain

$$N[\mathfrak{P}](i) = N[\mathfrak{N}](i)$$

and the weight change may be different. This is because the knowledge about $i^{G^{(L)}}$ provides further knowledge about the remaining PUs because of the specification of a stochastic model. It is thus

intuitive that if the variance of $G^{(L)}$ is high, the remaining states are more similar to the states in the feedforward network. This is supported by the following proposition.

2.95 Proposition. Consider a generative model $\mathfrak{G} = (\mathcal{G}, \xi)$ and a positive sequence $(a_n)_{n\in\mathbb{N}}$ such that $\lim_{n\to\infty} a_n = \infty$. Consider a sequence of functional structures on $\mathfrak{G}$ $(f, \Sigma^{(n)})_{n\in\mathbb{N}}$ such that all f^v are bounded and

$$\Sigma^{(n)} = \begin{pmatrix} \Sigma_{G^{(0)}\to G^{(0)}} & 0 & 0 \\ 0 & \Sigma_{G^{(1,\dots,L-1)}\to G^{(1,\dots,L-1)}} & 0 \\ 0 & 0 & a_n \cdot I_{G^{(L)}\to G^{(L)}} \end{pmatrix}$$

where

$$\Sigma_{G^{(0)}\to G^{(0)}} \in \mathbb{R}^{G^{(0)}\times G^{(0)}}, \quad \Sigma_{G^{(1,\dots,L-1)}\to G^{(1,\dots,L-1)}} \in \mathbb{R}^{G^{(1,\dots,L-1)}\times G^{(1,\dots,L-1)}}$$

are covariance matrices.
Let $\mathfrak{P}$ be the predictive coding network on $\mathfrak{G}$ and let $\mathfrak{N} = (\mathcal{G}, \xi)$ with functional structure f be the corresponding additive neural network. Let $i \in S^{G^{(0)}:i}[\mathfrak{P}], o \in S^{G^{(L)}:i}[\mathfrak{P}]$ be constant such that all $N_{\theta(\xi,\Sigma^{(n)})}[\mathfrak{P}](i,o)$ are well-defined. Then,

$$\lim_{n\to\infty} N^{\mathcal{G}}_{\theta(\xi,\Sigma^{(n)})}[\mathfrak{P}](i,o) = N_\xi[\mathfrak{N}](i) \tag{2.54}$$

and, if

$$\tilde{\mathcal{L}}^i_\alpha : \mathbb{R}^{\mathcal{G}\times\mathcal{G}} \times \mathbb{R}^{\mathcal{G}\times\mathcal{G}} \to \mathbb{R}^{\mathcal{G}\times\mathcal{G}}, \quad \tilde{\mathcal{L}}^i_\alpha(\xi,\Sigma) := \xi - \left(\frac{\partial L_{\xi,\Sigma}(N_{\xi,\Sigma}(i))}{\partial \xi}\right)^T,$$

as given by Theorem 2.88, yields

$$\mathcal{L}_\alpha := \left(\tilde{\mathcal{L}}_\alpha\right)_{\varsigma,\chi:\mathbb{R}^{\mathcal{G}\times\mathcal{G}}},$$

we obtain

$$\lim_{n\to\infty} \mathcal{L}^{(i,o)}_{\alpha a_n}(\xi, \Sigma^{(n)}) = \mathcal{B}^{i;o}_\alpha(\xi). \tag{2.55}$$

Proof. Clearly,

$$\left(\Sigma^{(n)}\right)^{-1} = \begin{pmatrix} \Sigma^{-1}_{G^{(0)}\to G^{(0)}} & 0 & 0 \\ 0 & \Sigma^{-1}_{G^{(1,\dots,L-1)}\to G^{(1,\dots,L-1)}} & 0 \\ 0 & 0 & \frac{1}{a_n} \cdot I_{G^{(L)}\to G^{(L)}} \end{pmatrix}.$$

Defining

$$s_n := N_{\theta(\xi,\Sigma^{(n)})}[\mathfrak{P}](i,o),$$

as $o \in S^{G^{(L):i}}[\mathfrak{P}]$ is constant,

$$s_n^{\varepsilon^{G^{(L)}}} = \left(\Sigma^{(n)}_{G^{(L)}\to G^{(L)}}\right)^{-1} (s_n^{G^{(L)}:i} - \xi^T_{\to G^{(L)}} s_n^{\mathcal{G}:o}) = \frac{1}{a_n}(o - \xi^T_{\to G^{(L)}} s_n^{\mathcal{G}:o}).$$

As f_n are bounded, $(o - \xi^T_{\to G^{(L)}} s_n^{\mathcal{G}:o})$ are bounded and, as $\lim_{n\to\infty} a_n = \infty$,

$$\lim_{n\to\infty} s_n^{\varepsilon^{G^{(L)}}} = 0.$$

By induction, Lemma 2.76 implies

$$\forall_{v\in\mathcal{G}\setminus G^{(0)}} \lim_{n\to\infty} s_n^{\varepsilon^v} = 0.$$

Analogously to Lemma 2.77 we can conclude (2.54) for the limit of s_n which also proves that it exists. It is straight-forward to prove by induction that

$$\lim_{n\to\infty} a_n s_n^{\varepsilon^{G^{(L)}}} = \lim_{n\to\infty} o - \xi^T s_n^{\mathcal{G}:o} = o - \xi^T_{\to G^{(L)}} N_\xi[\mathfrak{N}](i) = o - N^{\mathcal{G}}_\xi[\mathfrak{N}](i) = \delta^{G^{(L)}}$$

implies

$$\lim_{n\to\infty} a_n s_n^{\varepsilon} = \delta,$$

as both formulas share a recursive, linear formula. Therefore,

$$\lim_{n\to\infty}\left(\mathcal{L}^{(i,o)}_{\alpha a_n}(\xi,\Sigma^{(n)})\right)_{v\to w} = \xi_{v\to w} + \lim_{n\to\infty}\alpha a_n s_n^{\varepsilon^w} s_n^{v:o} =$$
$$\xi_{v\to w} + \alpha\lim_{n\to\infty}\delta^w s_n^{v:o} = \xi_{v\to w} + \alpha\delta^w N_\xi^{v:o}[\mathfrak{N}](i) = \mathcal{B}^{i;o}_\alpha(\xi). \qquad \square$$

2.96 (Biological Plausibility III: Enforcing Backpropagation) In this chapter, we have defined two kinds of networks: feedforward networks $\mathfrak{N}$ with backpropagation learning and predictive coding networks $\mathfrak{P}$. Among the reasons for the definition of a predictive coding network was a biologically plausible implementation of backpropagation. Proposition 2.95 yields such an implementation. The output variance simply has to be set high and be unlearnable and the resulting network satisfies locality of learning.
The relevance of such an implementation is, however, questionable. After all, predictive coding provides an entire model of reality, including hidden concepts whereas a feedforward network only implements a blunt prediction of y from x. The main reason why it is interesting to approximate backpropagation in a biologically plausible way is its proven effectiveness on the field of Machine Learning (see e. g. [5, 8]). However, a similar argument may be made vice versa: predictive coding promises a flexible and meaningful statistical model. While its power remains to be proven, we may, once again, be on the path of computational neuroscience inspiring statistics.
In summary, I would argue that biologically plausible implementations of backpropagation should not become an end in itself but rather launch a journey towards new models in neuroscience and statistics.[6] It would therefore be valuable to investigate the statistical properties of predictive coding as well as its (biologically plausible) capability to approximate other models. These questions motivate some initial investigations in the following chapter in which I hope to demonstrate that the mathematical potential of the framework I presented is far from exhausted.

[6] I would like to emphasize that I am also not suggesting that neuroscience faces such a risk of mathematization.

Chapter 3

Towards New Grounds

3.1 Linear Models

3.1 In order to get an intuition for the way predictive coding models work, it is sensible to study simple cases. After all, optimization by a closed form is very hard as soon as non-linearities are introduced by f. We will therefore provide some results that connect predictive coding models with linear models. We will consider ordinary linear models at first before going on to the simplest account of a linear model with a sensible interpretation in the biological context.

3.2 Let X and Y be finite index sets. In this section, we will be concerned with an underlying generative model $\mathfrak{G} = (X \cup Y, \xi)$ and functional structure (f, Σ), $f = \mathrm{Id}$, where

$$X \prec Y, \quad \xi := \begin{pmatrix} \xi_{X\to X} & \xi_{X\to Y} \\ \xi_{Y\to X} & \xi_{Y\to Y} \end{pmatrix} := \begin{pmatrix} 0 & \xi_{X\to Y} \\ 0 & 0 \end{pmatrix}, \quad \Sigma := \begin{pmatrix} \Sigma_{X\to X} & \Sigma_{X\to Y} \\ \Sigma_{Y\to X} & \Sigma_{Y\to Y} \end{pmatrix} := \begin{pmatrix} \Sigma_X & 0 \\ 0 & \Sigma_Y \end{pmatrix}$$

All lemmata and propositions are concerned with the resulting predictive coding network $\mathfrak{P}$.

3.3 Corollary. For a constant input $x \in \mathbb{R}^X$, if $N_{\xi,\Sigma}[\mathfrak{P}](x)$ exists,

$$N^Y_{\xi,\Sigma}[\mathfrak{P}](x) = \xi^T_{X\to Y} x.$$

Proof. Simple application of Lemma 2.77. □

3.4 Proposition. For a constant input $y \in \mathbb{R}^Y$, if $N_{\xi,\Sigma}[\mathfrak{P}](y)$ exists,

$$N^X_{\xi,\Sigma}[\mathfrak{P}](y) = \left(\Sigma_X^{-1} + \xi_{X\to Y}\Sigma_Y^{-1}\xi^T_{X\to Y}\right)^{-1} \xi_{X\to Y}\Sigma_Y^{-1} y. \tag{3.1}$$

Proof. Let

$$s := N_{\xi,\Sigma}[\mathfrak{P}](y).$$

Evidently,

$$s^{\varepsilon^Y} = \Sigma_Y^{-1}(y - \xi^T_{X\to Y} s^X), \quad s^{\varepsilon^X} = \Sigma_X^{-1} s^X.$$

On the other hand, Lemma 2.76 implies

$$s^{\varepsilon^X} = \xi_{X\to Y} s^{\varepsilon^Y}.$$

As Σ is positive and a block matrix, Σ_X and Σ_Y are positive definite by Lemma A.9. Proposition A.8 implies that $\xi_{X\to Y}\Sigma_Y^{-1}\xi^T_{X\to Y}$ is positive semidefinite and Σ_X^{-1} is positive definite. This, in turn, implies that $\Sigma_X^{-1} + \xi_{X\to Y}\Sigma_Y^{-1}\xi^T_{X\to Y}$ is positive definite and thus invertible. We now put these observations together to obtain

$$\Sigma_X^{-1} s^X = \xi_{X\to Y}\Sigma_Y^{-1}(y - \xi^T_{X\to Y} s^X),$$

and therefore

$$\Sigma_X^{-1} s^X + \xi_{X\to Y}\Sigma_Y^{-1}\xi^T_{X\to Y} s^X = \xi_{X\to Y}\Sigma_Y^{-1} y.$$

This, in turn, yields the formula from the proposition. □

3.5 We now consider how ξ, i. e. $\xi_{X\to Y}$, may be learned for a vector of examples

$$x = \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix} \in \mathbb{R}^{X\times n}, \quad y = \begin{pmatrix} y_1 & \dots & y_n \end{pmatrix} \in \mathbb{R}^{Y\times n}.$$

Identifying the fixed points of some learning function $C^{\binom{x}{y}}(\cdot, \Sigma)$ provides an intuition for good estimators of $\xi_{X\to Y}$ given the examples x, y. As discussed in 2.83, two viable alternatives in the context of gradient descent are given by the learning functions C and $\tilde{C}$ that are defined by

$$C^{\binom{x}{y}}_{X\to Y}(\xi, \Sigma) := \xi_{X\to Y} - \alpha \left(\frac{\partial \sum_{i=1}^n L_{\xi,\Sigma}\left(N_{\xi,\Sigma}\binom{x_i}{y_i}\right)}{\partial \xi_{X\to Y}} \right)^T,$$

$$\tilde{C}^{\binom{x}{y}}_{X\to Y}(\xi, \Sigma) := C^{\binom{x_n}{y_n}}\left(\cdots C^{\binom{x_1}{y_1}}(\xi, \Sigma)\right).$$

(We leave the notation of $\alpha > 0$ implicit as it is not particularly important.) C performs gradient descent (and therefore attempts optimization) on the loss function of all examples whereas $\tilde{C}$ iteratively applies gradient descent on every single example. $\tilde{C}$ seems to be more biologically applicable as we encounter stimuli in a serialized manner. However, Example 3.6 below demonstrates that $\tilde{C}$ has fixed points that do not account for all examples in an equal manner. In the remainder of the chapter, we will therefore consider the learning function C.

3.6 Example. Define $\mathfrak{N} = (\{v, w\}, \xi)$ with functional structure $f = \mathrm{Id}, \Sigma = I$ where $v \to w$ and suppose that the true $\xi_{v\to w} = 0$. Consider two training examples

$$\binom{x}{y} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

In this case, $\xi_{v\to w} = -\alpha$ is the only fixed point of $\tilde{C}$. $\tilde{C}$ is therefore clearly biased towards the latter example. On the other hand, C has the fixed point $\xi_{v\to w} = 0$. Note, however, that this does not imply that serial learning is nonsense but rather that it requires additional care and the parallel learning function C provides a better intuition at this point.

Proof. Suppose that $\xi_{v\to w}$ is a fixed point of $\tilde{C}$. Then,

$$\xi_{v\to w} = C^{\binom{x_2}{y_2}}\left(C^{\binom{x_1}{y_1}}(\xi_{v\to w})\right) = C^{\binom{1}{-1}}(\xi_{v\to w} + \alpha(1 - \xi_{v\to w}) \cdot 1) =$$
$$(\xi_{v\to w} + \alpha(1 - \xi_{v\to w}) \cdot 1) + \alpha(-1 - (\xi_{v\to w} + \alpha(1 - \xi_{v\to w}) \cdot 1) = -\alpha.$$

On the other hand, if $\xi_{v\to w}$ is a fixed point of C, we obtain

$$\xi_{v\to w} = C^{\binom{1\quad 1}{1\quad -1}}(\xi_{v\to w}) = \xi_{v\to w} + \alpha(1 - \xi_{v\to w}) + \alpha(-1 - \xi_{v\to w}) = -\xi_{v\to w}$$

and thus $\xi_{v\to w} = 0$. □

3.7 Proposition. If $x \in \mathbb{R}^{X\times n}, y \in \mathbb{R}^{Y\times n}$ are constant, x has full row rank and $N_{\xi,\Sigma}[\mathfrak{P}](x_i, y_i)$ converges for every $1 \le i \le n$, then the fixed points of C are given by

$$\xi_{X\to Y} = (xx^T)^{-1}xy^T.$$

Proof. We define

$$s_i := N_{\xi,\Sigma}[\mathfrak{P}](x_i, y_i),$$

and

$$s := \begin{pmatrix} s_1 & \cdots s_n \end{pmatrix} \in \mathbb{R}^{\mathcal{P}\times n},$$

as every $f^v = \mathrm{Id}$. As x and y are constant,

$$s^{\varepsilon^Y} = \Sigma_Y^{-1}(y - \xi_{X\to Y}^T x).$$

As $\xi_{X\to Y}$ is a fixed point of C the gradient of the loss functions has to be zero:

$$s^{\varepsilon^Y} x^T = \sum_{i=1}^{n} s_i^{\varepsilon^Y} \left(s_i^X\right)^T = 0.$$

Therefore,

$$\Sigma_Y^{-1}(y - \xi_{X\to Y}^T x)x^T = 0,$$

which implies

$$yx^T = \xi_{X\to Y}^T xx^T.$$

As x has full row rank, xx^T is invertible and we may conclude the proposition. □

3.8 Proposition 3.7 demonstrates that the optimal estimator of ξ in $\mathfrak{P}$ corresponds to the least squares estimator of the linear model

$$y^T = x^T \xi_{X\to Y}.$$

3.9 (Predictive Coding V: Priors) We will now discuss an important concept that has already been mentioned in 2.42: priors on the PUs of $G^{(0)}$. Whereas a normal distribution together with an expected value makes sense for a PU that has preceding units, the error of the value x of a highest-level PU v is given by

$$\epsilon^v = x.$$

The loss function therefore prefers values close to zero. We may prefer a flat prior that does not prioritize any value as more likely to a shrinkage prior that prefers values that are closer to zero. (Note that the present model could easily implement a PU v that returns values around $\mu \neq 0$ by defining $f^v(x) := x + \mu$.) Such a prior could be defined by changing the loss function $F_{\xi,\Sigma}(s)$. We would want to make sure that $\Sigma_{G^{(0)}\to\mathcal{G}} = 0$ such that we could write

$$\begin{aligned} F_{\xi,\Sigma}(s) =& \frac{1}{2}\left(\ln(\det\Sigma) + \left(\epsilon^{G^{(0)}}\right)^T \Sigma^{-1}_{G^{(0)}\to G^{(0)}} \left(\epsilon^{G^{(0)}}\right) + \left(\epsilon^{G^{(\geq 1)}}\right)^T \Sigma^{-1}_{G^{(\geq 1)}\to G^{(\geq 1)}} \left(\epsilon^{G^{(\geq 1)}}\right)\right) = \\ & \frac{1}{2}\left(\ln(\det\Sigma) + \left(s^{G^{(0)}}\right)^T \Sigma^{-1}_{G^{(0)}\to G^{(0)}} \left(s^{G^{(0)}}\right) + \left(\epsilon^{G^{(\geq 1)}}\right)^T \Sigma^{-1}_{G^{(\geq 1)}\to G^{(\geq 1)}} \left(\epsilon^{G^{(\geq 1)}}\right)\right). \end{aligned}$$

Setting $\Sigma^{-1}_{G^{(0)}\to G^{(0)}} = 0$ now yields the flat prior. However, Σ is no longer well-defined in this case.
As an alternative, we may choose Σ so that $\Sigma^{-1}_{G^{(0)}\to G^{(0)}}$ is very low. More precisely, given a strictly positive sequence $(a_n)_{n\in\mathbb{N}}$ such that $\lim_{n\to\infty} a_n = \infty$, we define

$$\Sigma_n := \begin{pmatrix} a_n \Sigma_{G^{(0)}\to G^{(0)}} & 0 \\ 0 & \Sigma_{G^{(\geq 1)}\to G^{(\geq 1)}} \end{pmatrix}.$$

Now, $\lim_{n\to\infty} (\Sigma_n)^{-1}_{G^{(0)}\to G^{(0)}} = 0$ and we may define (given that all values exist and converge)

$$s := \lim_{n\to\infty} s_n := \lim_{n\to\infty} N_{\xi,\Sigma_n}[\mathfrak{P}](i).$$

For a sufficiently large covariance matrix Σ_n, s_n would in turn be a good approximation of s with a well-defined generativ model.
This is clearly a rather complicated method to get rid of priors. It would be easier to define a predictive coding network without the error nodes in $G^{(0)}$ – corresponding to setting $\Sigma^{-1}_{G^{(0)}\to G^{(0)}} = 0$ within the loss function. The underlying generative model would then be no longer well-defined – however, we could define a generative model that is conditional on the values of $G^{(0)}$.
Whatever approach one may choose, it is still important to understand how the approximation by the limit works. Does the choice of $\Sigma_{G^{(0)}\to G^{(0)}}$ have an impact on s and does s differ from the result of the modified predictive coding network? We will investigate this question within the simple linear model. A short digression into the concept of weighted pseudoinverses will prove useful.

3.10 Definition (Moore-Penrose pseudoinverse). Given an arbitrary matrix $A \in \mathbb{R}^{m\times n}$, the *Moore-Penrose pseudoinverse* is defined as the unique matrix A^+ satisfying the following four conditions:

(i) $AA^+A = A$,

(ii) $A^+AA^+ = A^+$,

(iii) $(A^+A)^T = A^+A$,

(iv) $(AA^+)^T = AA^+$.

A proof of the unique existence of A^+ can for example be found in [16].

3.11 The Moore-Penrose pseudoinverse is a generalization of the inverse matrix that is useful in many different applications. Every student of statistics will have come upon the concept as the least squares estimator is given by

$$\hat{\beta} = X^+Y.$$

Specifically, if X has full column rank, we obtain the well-known formula

$$X^+ = (X^TX)^{-1}X^T.$$

3.12 Theorem. Let $A \in \mathbb{R}^{m\times n}, y \in \mathbb{R}^m$. Then,

$$\arg\min_{x\in\mathbb{R}^n} \|Ax - y\| = A^+y + \ker(A). \tag{3.2}$$

Proof. Theorem 6.1 in [16]. □

3.13 Theorem (Tikhonov's Regularization). For any $A \in \mathbb{R}^{m\times n}$, we may define the pseudoinverse by

$$A^+ = \lim_{a\to 0} A^T(AA^T + aI)^{-1} = \lim_{a\to 0}(A^TA + aI)^{-1}A. \tag{3.3}$$

Proof. Theorem 4.3 in [16]. □

3.14 The observant reader may already have noted the similarity between (3.1) if $\Sigma_X^{-1} \to 0$ and (3.3): if we define

$$x := \lim_{n\to\infty} N^X_{\xi,\Sigma_n}[\mathfrak{P}](y), \quad \Sigma_n := \begin{pmatrix} a_n\Sigma_X & 0 \\ 0 & \Sigma_Y \end{pmatrix}, \tag{3.4}$$

we may write

$$x = \hat{\beta}^T y, \quad \hat{\beta}^T = \lim_{n\to\infty}\left(\frac{1}{a_n}\Sigma_X^{-1} + \xi_{X\to Y}\Sigma_Y^{-1}\xi_{X\to Y}^T\right)^{-1}\xi_{X\to Y}\Sigma_Y^{-1}. \tag{3.5}$$

Specifically, if $\Sigma_X^{-1} = I_{X\to X}$ and $\Sigma_Y^{-1} = I_{Y\to Y}$, we know that

$$\hat{\beta} = \xi^+_{X\to Y}$$

is well-defined ($(A^+)^T = \left(A^T\right)^+$). Theorem 3.12 implies that

$$\arg\min_{\tilde{x}\in\mathbb{R}^X} \|\xi^T_{X\to Y}\tilde{x} - y\| = x + \ker\left(\xi_{X\to Y}\right).$$

As

$$\begin{aligned}\arg\min_{\tilde{x}\in\mathbb{R}^X}\lim_{n\to\infty} L_{\xi,\Sigma_n}(\tilde{x}, y) = &\arg\min_{\tilde{x}\in\mathbb{R}^X}\lim_{n\to\infty}\epsilon^T\Sigma_n^{-1}\epsilon = \\ &\arg\min_{\tilde{x}\in\mathbb{R}^X}\lim_{n\to\infty}\left(\frac{1}{a_n}\tilde{x}^T\Sigma_X^{-1}\tilde{x} + (y - \xi^T_{X\to Y}\tilde{x})^T\Sigma_Y^{-1}(y - \xi^T_{X\to Y}\tilde{x})\right) = \\ &\arg\min_{\tilde{x}\in\mathbb{R}^X}\|\xi^T_{X\to Y}\tilde{x} - y\| \in x + \ker(\xi_{X\to Y}),\end{aligned}$$

we have demonstrated that in this case, flattening the prior by a limit or by modifying the predictive coding network is, in a certain sense, equivalent. What if $\Sigma_Y^{-1} \neq I_{Y\to Y}$? How does changing Σ_X^{-1} affect the estimate? The answer to these questions is given by a generalization of the pseudoinverse that has been introduced by Chipman [17].

3.15 Definition (Weighted Pseudoinverse). If $A \in \mathbb{R}^{m\times n}$ is a matrix and $U \in \mathbb{R}^{n\times n}, V \in \mathbb{R}^{m\times m}$ are positive definite, the *U-V-weighted pseudoinverse* $A^{+(U,V)} \in \mathbb{R}^{n\times m}$ is defined as the unique matrix satisfying the following four conditions:

(i) $AA^{+(U,V)}A = A$,

(ii) $A^{+(U,V)}AA^{+(U,V)} = A^{+(U,V)}$,

(iii) $\left(A^{+(U,V)}A\right)^T = U^{-1}A^{+(U,V)}AU$,

(iv) $\left(AA^{+(U,V)}\right)^T = V^{-1}AA^{+(U,V)}V$.

Chipman also proved the existence and uniqueness of $A^{+(U,V)}$ [17, 1084f.].

3.16 Chipman introduced the weighted pseudoinverse as a transformation of the Moore-Penrose pseudoinverse by which one may define $A^{+(V,W)}$. For this transformation, we need the square root of a positive definite matrix.

3.17 Definition (Square Root of a Matrix). Let $A \in \mathbb{R}^{n\times n}$ be a positive semidefinite matrix with spectral decomposition

$$A = Q\Lambda Q^T$$

where Q is orthogonal and Λ a diagonal matrix. We define the *square root* of A by

$$A^{\frac{1}{2}} := Q\Lambda^{\frac{1}{2}}Q^T, \tag{3.6}$$

where

$$\Lambda^{\frac{1}{2}}_{ij} := \begin{cases} \sqrt{\Lambda_{ii}} & \text{if } i = j, \\ 0 & \text{else.} \end{cases} \tag{3.7}$$

Note that $\Lambda^{\frac{1}{2}}$ is well-defined as the diagonal entries are non-negative. Furthermore, we obtain

$$A^{\frac{1}{2}}A^{\frac{1}{2}} = Q\Lambda^{\frac{1}{2}}Q^TQ\Lambda^{\frac{1}{2}}Q^T = Q\Lambda Q^T = A,$$

and define

$$A^{-\frac{1}{2}} := \left(A^{\frac{1}{2}}\right)^{-1}.$$

3.18 Lemma. If $A \in \mathbb{R}^{m\times n}$ is a matrix and $U \in \mathbb{R}^{n\times n}, V \in \mathbb{R}^{m\times m}$ are positive definite and we define

$$\tilde{A} := V^{-\frac{1}{2}}AU^{\frac{1}{2}}, \tag{3.8}$$

the U-V-weighted pseudoinverse is given by

$$A^{+(U,V)} = U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}. \tag{3.9}$$

Proof. We simply have to prove that the proposed matrix satisfies the conditions of Definition 3.15 where we note that $A = V^{\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}}$.

(i) $AU^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}A = V^{\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}}U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}V^{\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}} = V^{\frac{1}{2}}\tilde{A}\tilde{A}^+\tilde{A}U^{-\frac{1}{2}} = V^{\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}} = A$,

(ii) $U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}AU^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}} = U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}V^{\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}}U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}} = U^{\frac{1}{2}}\tilde{A}^+\tilde{A}\tilde{A}^+V^{-\frac{1}{2}} = U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}$,

(iii) As the square root of a positive definite matrix is symmetrical, we observe

$$\left(U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}A\right)^T = \left(U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}V^{\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}}\right)^T = U^{-\frac{1}{2}}\left(\tilde{A}^+\tilde{A}\right)^TU^{\frac{1}{2}} =$$
$$U^{-\frac{1}{2}}\tilde{A}^+\tilde{A}U^{\frac{1}{2}} = U^{-\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}V^{\frac{1}{2}}\tilde{A}U^{\frac{1}{2}} = U^{-1}U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}V^{\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}}U = U^{-1}U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}AU,$$

(iv)

$$\left(AU^{\frac{1}{2}}\tilde{A}^{+}V^{-\frac{1}{2}}\right)^{T} = \left(V^{\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}}U^{\frac{1}{2}}\tilde{A}^{+}V^{-\frac{1}{2}}\right)^{T} = V^{-\frac{1}{2}}\left(\tilde{A}\tilde{A}^{+}\right)^{T}V^{\frac{1}{2}} =$$
$$V^{-\frac{1}{2}}\tilde{A}\tilde{A}^{+}V^{\frac{1}{2}} = V^{-\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}}U^{\frac{1}{2}}\tilde{A}^{+}V^{\frac{1}{2}} = V^{-1}V^{\frac{1}{2}}\tilde{A}U^{-\frac{1}{2}}U^{\frac{1}{2}}\tilde{A}^{+}V^{-\frac{1}{2}}V = V^{-1}AU^{\frac{1}{2}}\tilde{A}^{+}V^{-\frac{1}{2}}V.\square$$

3.19 Theorem. Let $A \in \mathbb{R}^{m\times n}$ and consider positive definite matrices $U \in \mathbb{R}^{n\times n}, V \in \mathbb{R}^{m\times m}$. If $y \in \mathbb{R}^{m}$,

$$\arg\min_{x\in\mathbb{R}^{n}} (Ax-y)^{T}V^{-1}(Ax-y) = A^{+(U,V)}y + \ker(A). \tag{3.10}$$

Proof. We prove the theorem by a reduction of the problem to Theorem 3.12 with the help of Lemma 3.18. We will need an additional lemma for the proof:

3.20 Lemma. Let X, Y be sets and $f : Y \to \mathbb{R}$ a function and $g : X \to Y$ a surjective function. Then, if f has a minimum on Y,

$$\arg\min_{y\in Y} f(y) = g\left(\arg\min_{x\in X} f\left(g(x)\right)\right).$$

Proof. $\subseteq$: suppose that

$$y \in \arg\min_{y\in Y} f(y).$$

In this case, for all $y' \in Y$, $f(y) \leq f(y')$. As g is surjective, there is some x such that $g(x) = y$. For all $x' \in X$, $g(x') \in Y$ and thus $f(g(x)) \leq f(g(x'))$ which implies

$$x \in \arg\min_{x\in X} f\left(g(x)\right),$$

and therefore

$$y \in g\left(\arg\min_{x\in X} f\left(g(x)\right)\right).$$

$\supseteq$: suppose that

$$y \in g\left(\arg\min_{x\in X} f\left(g(x)\right)\right).$$

In this case, there is some $x \in X$ such that $y = g(x)$ and

$$x \in \arg\min_{x\in X} f\left(g(x)\right).$$

Therefore, for all $x' \in X$, $f(g(x)) \leq f(g(x'))$. Let $y' \in Y$ be arbitrary. As g is surjective, there is some $x' \in X$ such that $y' = g(x')$ and thus $f(y) \leq f(y')$. This implies

$$y \in \arg\min_{y\in Y} f(y),$$

and concludes the proof. $\triangle$

Defining

$$\tilde{A} := V^{-\frac{1}{2}}AU^{\frac{1}{2}}, \quad \tilde{y} := V^{-\frac{1}{2}}y,$$

we know, by Theorem 3.12,

$$\tilde{A}^{+}\tilde{y} + \ker(\tilde{A}) = \arg\min_{\tilde{x}\in\mathbb{R}^{n}} \left(\tilde{A}\tilde{x}-\tilde{y}\right)^{T}\left(\tilde{A}\tilde{x}-\tilde{y}\right) =$$
$$\arg\min_{\tilde{x}\in\mathbb{R}^{n}} \left(V^{-\frac{1}{2}}AU^{\frac{1}{2}}\tilde{x}-V^{-\frac{1}{2}}y\right)^{T}\left(V^{-\frac{1}{2}}AU^{\frac{1}{2}}\tilde{x}-V^{-\frac{1}{2}}y\right) =$$
$$\arg\min_{\tilde{x}\in\mathbb{R}^{n}} \left(AU^{\frac{1}{2}}\tilde{x}-y\right)^{T}V^{-1}\left(AU^{\frac{1}{2}}\tilde{x}-y\right) \overset{\text{Lemma 3.20}}{=} U^{-\frac{1}{2}}\arg\min_{x\in\mathbb{R}^{n}} (Ax-y)^{T}V^{-1}(Ax-y),$$

where we have used the transformation $x = U^{-\frac{1}{2}}\tilde{x}$.
We observe that

$$x \in U^{\frac{1}{2}}\ker(\tilde{A}) \Leftrightarrow U^{-\frac{1}{2}}x \in \ker(\tilde{A}) \Leftrightarrow$$
$$V^{-\frac{1}{2}}Ax = \left(V^{-\frac{1}{2}}AU^{\frac{1}{2}}\right)U^{-\frac{1}{2}}x = \tilde{A}U^{-\frac{1}{2}}x = 0 \Leftrightarrow Ax = 0 \Leftrightarrow x \in \ker(A),$$

and thus

$$U^{\frac{1}{2}}\ker(\tilde{A}) = \ker(A).$$

This implies

$$\arg\min_{x\in\mathbb{R}^n}(Ax-y)^T V^{-1}(Ax-y) = U^{\frac{1}{2}}\left(\tilde{A}^+\tilde{y} + \ker(\tilde{A})\right) =$$
$$U^{\frac{1}{2}}\tilde{A}^+V^{-\frac{1}{2}}y + \ker(A) \overset{\text{Lemma 3.18}}{=} A^{+(U,V)}y + \ker(A). \qquad \square$$

3.21 Theorem. Let $A \in \mathbb{R}^{m\times n}$ and consider positive definite matrices $W \in \mathbb{R}^{p\times p}, V \in \mathbb{R}^{m\times m}$. Furthermore, let $B \in \mathbb{R}^{p\times n}$ be such that, for any $z \in \mathbb{R}^n$, there are vectors $x \in \mathbb{R}^m, y \in \mathbb{R}^p$ such that

$$A^T x + B^T y = z. \tag{3.11}$$

If we define

$$U := \left(A^T V^{-1} A + B^T W^{-1} B\right)^{-1}, \tag{3.12}$$

the U-V-weighted pseudoinverse of A is defined by the limit

$$A^{+(U,V)} = \lim_{a\to 0}\left(A^T V^{-1}A + aB^T W^{-1}B\right)^{-1} A^T V^{-1}. \tag{3.13}$$

Proof. Theorem 1 in [18]. (Note that the notation is slightly different for reasons of consistency.) $\square$

3.22 Corollary. Revisiting the considerations of 3.14, specifically

$$x := \lim_{n\to\infty} N^X_{\xi,\Sigma_n}[\mathfrak{P}](y), \qquad \Sigma_n := \begin{pmatrix} a_n\Sigma_X & 0 \\ 0 & \Sigma_Y \end{pmatrix}, \tag{3.4 revisited}$$
$$x = \hat{\beta}^T y, \qquad \hat{\beta}^T = \lim_{n\to\infty}\left(\frac{1}{a_n}\Sigma_X^{-1} + \xi_{X\to Y}\Sigma_Y^{-1}\xi^T_{X\to Y}\right)^{-1}\xi_{X\to Y}\Sigma_Y^{-1}, \tag{3.5 revisited}$$

we have now proven that x is well-defined as long as $N^X_{\xi,\Sigma_n}[\mathfrak{P}](y)$ is well-defined for all n. More specifically,

$$\hat{\beta} = \xi^{+(U,V)}_{X\to Y}, \tag{3.14}$$

where

$$U := \left(\Sigma_X^{-1} + \xi_{X\to Y}\Sigma_Y^{-1}\xi^T_{X\to Y}\right)^{-1}, \quad V := \Sigma_Y.$$

Furthermore, x minimizes the loss function with flat prior on X that is given by

$$\left(y - \xi^T_{X\to Y}x\right)^T \Sigma_Y^{-1}\left(y - \xi^T_{X\to Y}x\right). \tag{3.15}$$

Proof. (3.14) follows from Theorem 3.21 by setting $B := I \in \mathbb{R}^{X\times X}$. This already implies condition (3.11). Furthermore, $A = \xi^T_{X\to Y} \in \mathbb{R}^{Y\to X}$ and $W = \Sigma_X$. (3.15) follows from Theorem 3.19. $\square$

3.23 The presented model does not satisfy one important condition in a biological context that has been mentioned in 2.42: the causes are scarcely empirically measurable. The linear model we have considered in the last section therefore does not paint the full picture as X, as a cause would normally be latent. Instead, all inputs from the environment reach the level $G^{(L)}$. The preceding PUs are constructed as helpful concepts to make sense of these inputs. The simplest model that relates some

empirical input to another empirical input is the predictive coding network built upon the additive normal generative model $\mathfrak{G} := (H \cup X \cup Y, \xi)$ with functional structure (Id, Σ) where

$$H \prec X \approx Y,$$

and

$$\Sigma := \begin{pmatrix} \Sigma_{H\to H} & 0 & 0 \\ 0 & \Sigma_{X\to X} & 0 \\ 0 & 0 & \Sigma_{Y\to Y} \end{pmatrix} =: \begin{pmatrix} \Sigma_H & 0 & 0 \\ 0 & \Sigma_X & 0 \\ 0 & 0 & \Sigma_Y \end{pmatrix}.$$

In this framework, we may predict the empirical input X by the empirical input Y.

3.24 Proposition. If $x \in \mathbb{R}^X$ is constant and $s := N_{\xi,\Sigma}[\mathfrak{P}](x)$ well-defined,

$$s^H = (\Sigma_H^{-1} + \xi_{H\to X}\Sigma_X^{-1}\Sigma_{H\to X}^T)^{-1}\xi_{H\to X}\Sigma_X^{-1}x, \quad s^Y = \xi_{H\to Y}^T s^H.$$

Proof. As s^Y is not constant, $\Sigma^{-1}(s^Y - \xi_{H\to Y}^T s^H) = s^{\varepsilon^Y} = 0$ and therefore $s^Y = \xi_{H\to Y}^T s^H$. The formula for s^H can be obtained by an argument similar to Proposition 3.4 but keeping in mind that $s^{\varepsilon^Y} = 0$.□

3.2 Results and Outlook

In this Bachelor's thesis, I have introduced a general framework for neural networks that distinguishes their structure from the process by which they take certain values and is able to learn its parameters. In particular, I have defined the backpropagation algorithm on feedforward additive neural networks and introduced predictive coding, a model in computational neuroscience that is also able to learn its parameters. I have demonstrated that under specific conditions, learning in predictive coding converges against the backpropagation algorithm. Furthermore, I have studied the simple linear predictive coding network in more depth.

Future investigations may include a more detailed description of how the covariance matrix of the predictive coding network is learned. Connecting back to the thoughts of chapter 1, an implementation of general predictive coding networks in a programming language, for instance the statistical programming language R [19], is crucial to apply the methodology to the fields of computational neuroscience and statistics. In its early days, development of predictive coding was inspired by the statistical methodology of expectation maximization [7, 12]. Today, applications of predictive coding to statistics may provide fruitful prespectives, for example as a statistical system that integrates prediction and imputation of its covariates. Such an interdisciplinarity, grounded by rigorous mathematical analysis, promises exciting new developments in all concerned fields.

Bibliography

1. Sejnowski, T. J., Koch, C. & Churchland, P. S. Computational Neuroscience. *Science* **241,** 1299–1306 (1988).
2. Box, G. E. P. Science and Statistics. *Journal of the American Statistical Association* **71,** 791–799. ISSN: 00034819 (1976).
3. Hodgkin, A. L. & Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology* **117,** 500–544. ISSN: 00928240 (1952).
4. Rojas, R. *Neural Networks* (Springer-Verlag, Berlin, 1996).
5. Liao, Q., Leibo, J. Z. & Poggio, T. How Important is Weight Symmetry in Backpropagation?, 1837–1844 (2015).
6. Rao, R. P. N. & Ballard, D. H. Predictive Coding in the Visual Cortex: a Functional Interpretation of Some Extra- classical Receptive-field Effects. *Nature Neuroscience* **2.** doi:10.1038/4580 (1999).
7. Friston, K. A theory of cortical responses. *Philosophical Transactions of the Royal Society B* **360,** 815–836 (2005).
8. Whittington, J. C. R. & Bogacz, R. An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity. *Neural Computation* **29,** 1229–1262. ISSN: 1530888X (2017).
9. Diestel, R. *Graph Theory* 5th ed., 429. ISBN: 978-3-662-53622-3. doi:10.1007/978-3-662-53622-3 (Springer-Verlag Berlin Heidelberg, 2017).
10. Guresen, E. & Kayakutlu, G. Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science* **3,** 426–433 (2011).
11. Hyde, D. & Raffman, D. *Sorites Paradox* 2014. https://plato.stanford.edu/archives/win2014/entries/sorites-paradox/.
12. Dempster, A. P., Laird, N. M. & Rubin, D. B. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* **39,** 1–38 (1977).
13. Hebb, D. O. *The Organization of Behavior* (John Wiley & Sons, New York, 1949).
14. Gerstner, W. & Kistler, W. M. Mathematical formulations of Hebbian learning. *Biological Cybernetics* **87,** 404–415 (2002).
15. Bogacz, R. A tutorial on the free-energy framework for modelling perception and learning. *Journal of Mathematical Psychology* **76,** 198–211 (2017).
16. Barata, J. C. A. & Hussein, M. S. The Moore-Penrose Pseudoinverse: A Tutorial Review of the Theory. *Brazilian Journal of Physics* **42,** 146–165. ISSN: 01039733 (2012).
17. Chipman, J. S. On Least Squares with Insufficient Observations. *Journal of the American Statistical Association* **59,** 1078–1111 (1964).
18. Ward, J. F. On a Limit Formula for Weighted Pseudoinverses. *SIAM Journal on Applied Mathematics* **33,** 34–38 (1977).
19. R Core Team. *R: A Language and Environment for Statistical Computing* R Foundation for Statistical Computing (Vienna, Austria). https://www.r-project.org.

20. Golberg, M. A. The Derivative of a Determinant. *The American Mathematical Monthly* **79,** 1124–1126 (1972).

Appendix A

Matrix Rules

This appendix is concerned with some lemmata that are helpful to prove the results of the Bachelor's thesis.

A.1 Matrix Derivation

A.1 Lemma. If $A \in \mathbb{R}^{n\times n}, x \in \mathbb{R}^n$, $\frac{\partial x^T Ax}{\partial x} = x^T(A + A^T)$.

Proof.

$$\frac{\partial}{\partial x_i} x^T Ax = \frac{\partial}{\partial x_i} \sum_{j=1}^{n}\sum_{k=1}^{n} A_{jk}x_j x_k = \sum_{j\neq i} A_{ji}x_i + \sum_{k\neq i} A_{ik}x_k + 2A_{ii}x_i =$$
$$\sum_{j=1}^{n} A_{ji}x_i + \sum_{k=1}^{n} A_{ik}x_k = x^T(A_{\cdot i}^T + A_{\cdot i}). \qquad \square$$

A.2 Lemma. If $A \in \mathbb{R}^{n\times n}, x \in \mathbb{R}^n$,

$$\frac{\partial x^T Ax}{\partial A} = xx^T.$$

Proof.

$$\frac{\partial}{\partial A_{ij}} x^T Ax = \frac{\partial}{\partial A_{ij}} \sum_{k=1}^{n}\sum_{l=1}^{n} A_k lkx_k x_l = x_i x_j \qquad \square$$

A.3 Lemma (Product Rule). If $X_M, X_N \subseteq \mathbb{R}$ are open subsets such that

$$M : X_M \to \mathbb{R}^{l\times m}, \quad N : X_N \to \mathbb{R}^{m\times n},$$

and $M(X_M) \subseteq X_N$, we obtain

$$\frac{\partial}{\partial x}(MN)(x) = M(x)\left(\frac{\partial}{\partial x}N(x)\right) + \left(\frac{\partial}{\partial x}M(x)\right)N(x).$$

Proof.

$$\frac{\partial}{\partial x}(MN)_{ij}(x) = \frac{\partial}{\partial x}M_{\cdot i}(x)N_{\cdot j}(x) = \frac{\partial}{\partial x}\sum_{k=1}^{m} M_{ik}(x)N_{kj}(x) =$$
$$\sum_{k=1}^{m} M_{ik}(x)\left(\frac{\partial}{\partial x}N_{kj}(x)\right) + \left(\frac{\partial}{\partial x}M_{ik}(x)\right)N_{kj}(x) = \left(M(x)\left(\frac{\partial}{\partial x}N(x)\right)\right)_{ij} + \left(\left(\frac{\partial}{\partial x}M(x)\right)N(x)\right)_{ij} \square$$

A.4 Lemma. If $X \subseteq \mathbb{R}$ is an open subset,

$$A : X \to \mathbb{R}^{n \times n}$$

is a differentiable function and $A(x)$ is invertible for all $x \in X$,

$$\frac{\partial A(x)^{-1}}{\partial x} = -A(x)^{-1} \frac{\partial A(x)}{\partial x} A(x)^{-1}.$$

Proof.

$$0 = \frac{\partial}{\partial x} I = \frac{\partial}{\partial x} A(x) A(x)^{-1} \overset{\text{Lemma A.3}}{=} A(x) \left(\frac{\partial}{\partial x} A(x)^{-1} \right) + \left(\frac{\partial}{\partial x} A(x) \right) A(x)^{-1},$$

which implies the lemma. □

A.5 Lemma (Chain Rule for matrix-valued functions). If $X_M \subseteq \mathbb{R}$ is open,

$$M : X_M \to \mathbb{R}^{m \times n}$$

is differentiable and $X_f \subseteq \mathbb{R}^{m \times n}$ is open such that $M(X_M) \subseteq X_f$, and

$$f : X_f \to \mathbb{R}$$

is differentiable, then

$$f \circ M : \mathbb{R} \to \mathbb{R}, \quad x \mapsto (f \circ M)(x)$$

is differentiable and

$$\frac{\partial f\left(M(x)\right)}{\partial x} = \operatorname{tr}\left(\frac{\partial f\left(M(x)\right)}{\partial M(x)} \frac{\partial M(x)}{\partial x} \right).$$

Proof. We prove the lemma by transforming the matrix $M \in \mathbb{R}^{m \times n}$ into a vector $\tilde{M} \in \mathbb{R}^{mn}$ using the bijective mapping

$$\Psi : \mathbb{R}^{m \times n} \to \mathbb{R}^{mn}, \underset{\substack{i=1,\dots,m \\ j=1,\dots,n}}{\forall} \Psi_{m(j-1)+i}(M) := M_{ij}.$$

We define

$$\tilde{M} := \Psi \circ M, \quad \tilde{f} := f \circ \Psi^{-1},$$

which implies

$$f \circ M = \tilde{f} \circ \tilde{M}.$$

The matrix chain rule now follows directly from the chain rule for vector-valued functions although the notation is a bit cumbersome:

$$\frac{\partial f\left(M(x)\right)}{\partial x} = \frac{\partial \tilde{f}\left(\tilde{M}(x)\right)}{\partial x} = \frac{\partial \tilde{f}\left(\tilde{M}(x)\right)}{\partial \tilde{M}(x)} \frac{\partial \tilde{M}(x)}{\partial x} =$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \frac{\partial \tilde{f}\left(\tilde{M}(x)\right)}{\partial \tilde{M}_{m(j-1)+i}(x)} \cdot \frac{\partial \tilde{M}_{m(j-1)+i}(x)}{\partial x} = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{\partial f\left(M(x)\right)}{\partial M_{ij}(x)} \cdot \frac{\partial M_{ij}(x)}{\partial x} =$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \left(\frac{\partial f\left(M(x)\right)}{\partial M(x)} \right)_{ji} \cdot \left(\frac{\partial M_{ij}(x)}{\partial x} \right)_{ij} = \sum_{j=1}^{n} \left(\frac{\partial f\left(M(x)\right)}{\partial M(x)} \frac{\partial M_{ij}(x)}{\partial x} \right)_{jj} = \operatorname{tr}\left(\frac{\partial f\left(M(x)\right)}{\partial M(x)} \frac{\partial M(x)}{\partial x} \right) \square$$

A.6 Lemma (Jacobi's Formula). If $\Lambda_i \subseteq \mathbb{R}$ is open and

$$A : \Lambda_i \to \mathbb{R}^{n\times n}$$

is a matrix-valued function,

$$\frac{\partial \det A(\lambda)}{\partial \lambda} = \operatorname{tr}\left(\operatorname{Adj} A(\lambda) \frac{\partial}{\partial \lambda} A(\lambda)\right),$$

where $\operatorname{Adj} A(\lambda)$ is the classical adjoint matrix of $A(\lambda)$.
In particular, if A is invertible,

$$\frac{\partial \det A(\lambda)}{\partial \lambda} = \det A \operatorname{tr}\left(A(\lambda)^{-1} \frac{\partial}{\partial \lambda} A(\lambda)\right).$$

Proof. The proof can, for example, be found in [20]. The second equation is clear as the adjoint matrix suffices the formula

$$A \operatorname{Adj}(\mathrm{A}) = \det A \, I. \qquad \square$$

A.2 Positive definite and semidefinite matrices

A.7 Definition (Positive (semi-)definite matrices). A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is called *positive definite* (resp. *semidefinite*) if and only if for all $0 \neq a \in \mathbb{R}^n$,

$$a^T A a > 0 \quad (\text{resp. } a^T A a \geq 0).$$

Clearly, positive definite matrices are also positive semidefinite.

A.8 Proposition. Let $A, B \in \mathbb{R}^{n\times n}$ be such that A is positive definite and B is positive semidefinite.

a) $A + B$ is also positive definite,

b) For some $\lambda > 0$, λA is also positive definite and λB is also positive semidefinite,

c) The following statements are equivalent:

   (i) B is positive definite,
   (ii) B is invertible,
   (iii) B only has strictly positive eigenvalues,

d) A is invertible and A^{-1} is also positive definite,

e) If $C \in \mathbb{R}^{n\times m}$, $C^T B C$ is also positive semidefinite. If $\operatorname{rk}(C) = m$, $C^T A C$ is positive definite.

Proof. Let $0 \neq a \in \mathbb{R}^n$.

a) $a^T(A+B)a = \underbrace{a^T A a}_{>0} + \underbrace{a^T B a}_{\geq 0} > 0$,

b) $a^T(\lambda A)a = \lambda a^T A a > 0$, $a^T(\lambda B)a = \lambda a^T B a \geq 0$,

c) As B is positive semidefinite, positive definiteness of B is equivalent to

$$\underset{a\in\mathbb{R}^n}{\forall} \left(a^T B a = 0 \Rightarrow a = 0\right).$$

If B is not invertible, it does not have full rank and its columns are linearly dependent: there is $0 \neq a \in \mathbb{R}^n$ such that $Ba = 0$. In this case, positive definiteness is clearly not satisfied and therefore $(i) \Rightarrow (ii)$. As B is symmetric, it has the spectral decomposition

$$B = Q^T \Lambda Q,$$

where Q is an orthogonal matrix, i. e. $Q^TQ = I$ and Λ is the diagonal matrix of eigenvalues. If we suppose (ii), if B is invertible, so is

$$\Lambda = QBQ^T.$$

Therefore every entry on the diagonal is non-zero. Suppose that the entry $\lambda_{ii} < 0$. In this case, if

$$Q = \begin{pmatrix} Q_1 \\ \vdots \\ Q_n \end{pmatrix},$$

we obtain

$$Q_i B Q_i^T = \lambda_{ii} < 0,$$

which is a contradiction as B is positive semidefinite. Therefore all entries on the diagonal (and therefore the eigenvalues of B) are strictly positive. Finally, considering the same spectral decomposition and supposing that the diagonal entries are strictly positive, we may define $\Lambda^{\frac{1}{2}}$ by

$$\Lambda^{\frac{1}{2}}_{ij} := \begin{cases} \sqrt{\lambda_{ii}} & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

Clearly, $\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}} = \Lambda$. We obtain, for any $0 \neq a \in \mathbb{R}^n$,

$$a^TBa = (Qa)^T\Lambda(Qa) = (\Lambda^{\frac{1}{2}}Qa)^T(\Lambda^{\frac{1}{2}}Qa) > 0$$

as Q is orthogonal and therefore invertible and $\Lambda^{\frac{1}{2}}$ is a diagonal matrix with only nonzero entries and therefore invertible.

d) A is invertible by c) as it is positive definite. If $0 \neq a \in \mathbb{R}^n$,

$$a^TA^{-1}a = a^TA^{-1}AA^{-1}a = (A^{-1}a)^TA(A^{-1}a) > 0,$$

as A^{-1} obviously has full rank. This implies that $A^{-1}a \neq 0$.

e) If $0 \neq a \in \mathbb{R}^m$, $a^TC^TBCa = (Ca)^TBCa \geq 0$. On the other hand, if C has rank m, its columns are independent which implies that for any $0 \neq a \in \mathbb{R}^m$, $Ca \neq 0$ and therefore $a^TC^TACa = (Ca)^TB(Ca) > 0$. □

A.9 Lemma. If a block matrix

$$A = \begin{pmatrix} B & 0 \\ 0 & C \end{pmatrix}, \quad B \in \mathbb{R}^{m\times m}, \quad C \in \mathbb{R}^{n\times n},$$

is positive definite, so are B and C.

Proof. If B were not positive definite, take $0 \neq a \in \mathbb{R}^m$ such that

$$a^TBa \leq 0,$$

and define $\tilde{a} \in \mathbb{R}^{m+n}$ where

$$\tilde{a}^i := \begin{cases} a^i & \text{if } i \leq m, \\ 0 & \text{if } i > m. \end{cases}$$

This implies

$$\tilde{a}^TA\tilde{a} = a^TBa + 0 \leq 0,$$

which contradicts positive definiteness of A. A similar argument shows positive definiteness of C. □