Decomposition of Deep Neural Networks into Correlation Functions

Author
Kirsten Fischer

First Reviewer
Prof. Dr. Moritz Helias

Second Reviewer
Prof. Dr. Carsten Honerkamp

Master Thesis in Physics

presented to

RWTH Aachen University
Faculty of Mathematics, Computer Science and Natural Sciences

performed at

Juelich Research Center
Institute of Computational and Systems Neuroscience (INM-6)
Institute for Advanced Simulation (IAS-6)

November, 2020

Abstract

Recent years have shown a great success of deep neural networks. One active field of research investigates the functioning mechanisms of such networks with respect to the network expressivity as well as information processing within the network.

In this thesis, we describe the input-output mapping implemented by deep neural networks in terms of correlation functions. To trace the transformation of correlation functions within neural networks, we make use of methods from statistical physics. Using a quadratic approximation for non-linear activation functions, we obtain recursive relations in a perturbative manner by means of Feynman diagrams. Our results yield a characterization of the network as a non-linear mapping of mean and covariance, which can be extended by including corrections from higher order correlations.

Furthermore, re-expressing the training objective in terms of data correlations allows us to study their role for solutions to a given task. First, we investigate an adaptation of the XOR problem, in which case the solutions implemented by neural networks can largely be described in terms of mean and covariance of each class. Furthermore, we study the MNIST database as an example of a non-synthetic dataset. For MNIST, solutions based on empirical estimates for mean and covariance of each class already capture a large amount of the variability within the dataset, but still exhibit a non-negligible performance gap in comparison to solutions based on the actual dataset. Lastly, we introduce an example task where higher order correlations exclusively encode class membership, which allows us to explore their role for solutions found by neural networks.

Finally, our framework also allows us to make predictions regarding the correlation functions that are inferable from data, yielding insights into the network expressivity. This work thereby creates a link between statistical physics and machine learning, aiming towards explainable AI.

Acknowledgements

There are many people to whom I want to extend my gratitude for supporting me throughout this thesis. First and foremost, I thank Prof. Dr. Moritz Helias for giving me the opportunity to join his research group at the INM-6 of the Juelich Research Center for this thesis. Above all, I am grateful for his calm support and encouraging guidance, the valuable ideas and thoughtful feedback as well as endless insightful discussions. He has been a great supervisor and essential for bringing this thesis to life.

My gratitude goes to Dr. David Dahmen for his valuable advice and inexhaustible creativeness. I also want to thank Alexandre René for numerous discussions as well as his detailed feedback.

I want to thank Prof. Dr. Carsten Honerkamp for hatching the idea for this project together with Prof. Dr. Moritz Helias and others. Furthermore, he kindly agreed to act as second reviewer for this thesis.

My deepest gratitude belongs to my best friend Debbora Leip, not only for carefully reviewing every last sentence of this thesis, but also for her constant support throughout the past years and in particular for grounding me whenever necessary. In addition, I thank my brother Jens Fischer for enduring his role as my linguistic reference as well as putting a final touch on my wording.

Finally, I thank all my colleagues at INM-6 for the welcoming and inspiring atmosphere which significantly shaped this entire experience.

Contents

Introduction

In recent years, deep neural networks have proven to be powerful tools for a wide range of applications, from image recognition [1] to playing Go [2]. However, our theoretical understanding regarding their empirical success is still limited at best. This includes questions as to what functions they can compute, how they process information or why they generalize, to name but a few [3]. Especially in fields such as medicine [4] or autonomous driving [5], finding answers to these questions is essential for developing reliable tools.

One major branch in machine learning is supervised learning, which addresses learning an input-output mapping from examples. If the possible output values are given by a finite set, this task is referred to as classification [6]. Many common problems are formulated in this way: In image classification, images are assigned to class labels such as 'cat' or 'dog' based on their pixel representation. In automated medical diagnosis, medical data such as EEG recordings are analyzed with respect to possible anomalies and indications for diseases [7, 8, 9]. To arrive at a transparent decision-making process, a more profound understanding regarding the functional principles of neural networks and their extraction of meaningful features from given data is required.

The objective of classification is to minimize an error measure between the correct class label and the prediction made by the neural network with respect to the joint probability distribution of data samples and class labels [3]. Thus, training dynamics and consequently the solution strategy implemented by the network depend on this probability distribution and more precisely the information encoded in it. In this context, processing of data samples by the network amounts to a transformation of the input distribution. The classification objective can then be reformulated as finding a transformation for which the distribution of the network output is strongly concentrated at the assigned target values for each class, reminiscent of a superposition of Dirac delta functions.

In this thesis, we study the transformation of the data distribution by the network to obtain insights into how the solution strategy found by the network employs information encoded in this distribution. Typically, a distribution is given in terms of its probability density function. However, the probability density function assigns a certain value to each input, thereby giving a priori a local description of the input data. In consequence, it can be difficult to discern global properties of a given class. Furthermore, the data distribution might not even have a mathematical representation in terms of a probability density function. Therefore, we exploit the fact that we can fully describe a distribution by its data statistics or cumulants, also referred to as connected correlation functions [10]. In this context, tracing how the input distribution is transformed by a deep neural network amounts to decomposing the network mapping into correlation functions.

A key element for the expressivity of neural networks is the application of a nonlinear activation function at intermediate network layers. Seminal works have shown

that multi-layer feed forward networks can approximate any continuous function as long as the activation function is not a polynomial [11, 12, 13]. To study the transformation of cumulants by a non-linear activation function, we express the corresponding cumulant-generating function using Feynman diagrams, which is a well-established method from statistical physics [14].

Regarding the information contained in the cumulants of the data distribution, the two lowest order cumulants are easily interpretable as they are given by mean and covariance. In the special case of a Gaussian distribution, these already define the distribution and, accordingly, all higher order cumulants beyond mean and covariance vanish. In the case of a general distribution, cumulants of arbitrary orders need to be taken into account for an exact description of the distribution [15]. However, we cannot take into account cumulants of arbitrary order, which raises questions regarding the information encoded in a cumulant of a certain order as well as their relative influence on the network mapping.

Furthermore, the exact probability distribution of data samples and class labels is generally not accessible. Instead, the error measure used during training is determined as an empirical estimate based on pairs of a data sample and corresponding class label [16]. Accordingly, we can determine empiric estimates for the cumulants of the data distribution. For classification tasks such as image recognition, the empirical mean of each class can be interpreted as a representative for this class and the corresponding covariance as an initial estimate of the variability within that class. Depending on the particular task, these quantities might already encode class membership to a large extent. Therefore, we will initially constrain our considerations to studying how mean and covariance are transformed by a neural network. This approach corresponds to approximating the distribution of the network input as a Gaussian mixture model with one component for each class.

With regard to the internal representation of deep neural networks, an established line of research studies the case of infinitely wide neural networks for which an exact equivalence to Gaussian processes exists [17, 18, 19]. By applying the central limit theorem at intermediate network layers, a prior over possible network mappings is derived based on the initial distribution of network parameters. This result allows Bayesian prediction with deep neural networks [19]. In contrast to this previous work, here we do not study the distribution with respect to the network parameters, but with respect to the data samples. Nevertheless, we can take a similar approach by assuming the activations at intermediate layers to be self-averaging and thus applying a disorder average for the cumulant-generating function with respect to the network parameters. As a result, higher order cumulants scale down with the network width. Therefore, we will initially restrict our considerations to mean and covariance.

Accordingly, this thesis is divided into two parts: In Part I, we study how feed forward networks with a quadratic activation function (see Chapter 1) transform Gaussian data statistics. We assess the accuracy of the resulting network description for networks with randomly sampled parameters (see Chapter 2) as well as for networks that have been trained to solve an adaptation of the XOR problem (see Chapter 3). By re-expressing the error measure used during training in terms of the data statistics of the network input (see Chapter 4), we can study how these quantities drive network training and are processed by the network (see Chapter 5). Furthermore, we apply this method to the MNIST database as an example of non-synthetic data (see Chapter 6) and explore limitations of such a Gaussian description of the network mapping (see Chapter 7).

In Part II, we investigate the influence of higher order correlations in the input distribution as well as the network mapping (see Chapter 8). Based on these considerations, we study both implications and limitations of an accordingly adapted network description (see Chapter 9). Lastly, we explore how the network expressivity relates to the information contained in data statistics of different orders (see Chapter 10).

Part I

Gaussian Statistics in Neural Networks

1 Feed forward network with quadratic activation function

In this thesis, we use feed forward networks which process data in a hierarchical manner as they are functionally structured into layers. Each layer performs an affine linear transformation which is defined by a weight matrix $W^l \in \mathbb{R}^{N_l \times N_{l-1}}$ and a bias vector $b^l \in \mathbb{R}^{N_l}$, followed by the pointwise application of a function $\phi : \mathbb{R} \to \mathbb{R}$ which is called *activation function*. Here, N_l defines the width of layer l and we use the convention $N_{-1} = d_{\text{in}}$ with d_{in} being the dimensionality of the network input $x \in \mathbb{R}^{d_{\text{in}}}$. The total number of such non-linear layers within a network is called the *network depth* L.

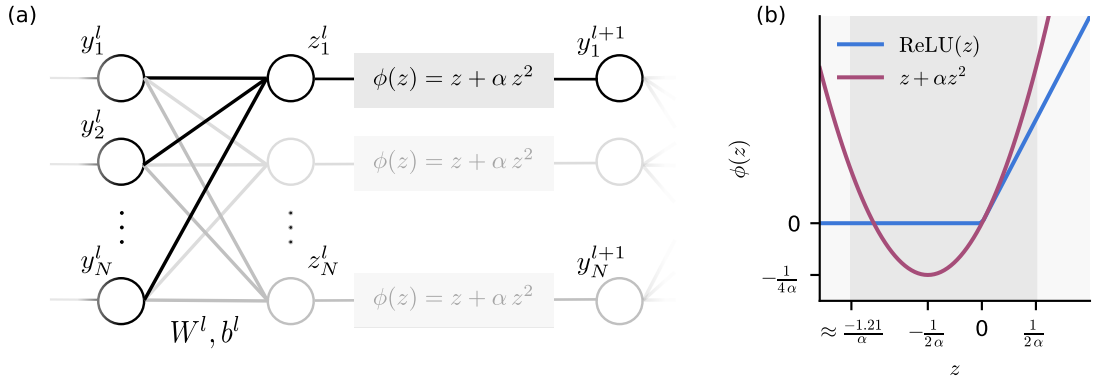


Figure 1: **(a)** Schematic representation of a network layer l. **(b)** The ReLU function (blue) can be approximated by the quadratic activation function (red) defined in (3). The range for which this approximation is applicable (dark gray) depends on the strength α of the quadratic term.

In addition, we use a purely linear readout layer as this choice allows for the application of the disorder average (see Appendix A.1) to the network output, which facilitates its theoretical description. Thus, the overall network mapping $g_\theta : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ is given by

$$\left.\begin{aligned} z_i^l &= (W^l\, y^l + b^l)_i, \\ y_i^{l+1} &= \phi(z_i^l), \end{aligned}\right\} \quad \text{for } l = 0, \dots, L-1 \tag{1}$$

$$z_i^L = (W^L\, y^L + b^L)_i, \tag{2}$$

where we set $y^0 = x$ and denote the set of network parameters as $\theta = \{W^l, b^l\}_{l=0,\dots,L}$. For each layer, z^l and y^l are referred to as *pre- and post-activations*, respectively (see Fig. 1(a)).

The network output is given by $g_\theta(x) = z^L$ and consequently, we have $N_L = d_{\text{out}}$ with d_{out} being the dimensionality of the network output. Since we mostly consider classification tasks with binary class labels, we set $d_{\text{out}} = 1$ unless specified otherwise. Additionally, we use a fixed width for all intermediate layers, so that $N_l = N$ for $l = 0, \ldots, L-1$, which we henceforth refer to as the *network width* N.

Common choices for the activation function are the hyperbolic tangent or the *rectified linear unit function* ReLU, which is defined as $\text{ReLU}(z) \coloneqq \max(0, z)$, due to their good performance and generalization capabilities [20]. Instead, in this work we use a quadratic activation function defined as

$$\phi(z) \coloneqq z + \alpha\, z^2, \tag{3}$$

where α determines the strength of the quadratic term.

This choice of activation function represents the most fundamental instance of a non-linear function and, unlike many common activation functions, is suitable for theoretical analysis. Under the assumption that α is small, a quadratic activation function can further be seen as an approximation of the ReLU function for values close to zero, as illustrated in Fig. 1(b). This relation can also be derived by performing a Taylor expansion of $\text{ReLU}(z)$ around zero (see Appendix A.2). In the following, the activation function $\phi(z)$ is always assumed to be a quadratic function as given in (3) unless specified otherwise.

2 Transformation of Gaussian statistics within networks

We first study how the distribution of the input data is transformed by a network in terms of the corresponding cumulants. Based on the assumption that for classification tasks, a large amount of the relevant information is contained in mean and covariance of each class, in this section we constrain our considerations to the case of Gaussian distributed input data.

We first analytically derive expressions that describe how mean and covariance of the data distribution are transformed by each network layer (see Section 2.1). These results can also be obtained using a graphical notation for the occurring algebraic terms, leading to Feynman diagrams (see Section 2.2). Based on the disorder average in Appendix A.1, we obtain a description of the network as a non-linear mapping of mean and covariance (see Section 2.3). We then compare our theoretical results with empirical estimates of the distribution obtained from data samples in the case of untrained networks (see Section 2.4).

2.1 Cumulant transformation by a single network layer

Since our goal is to study the transformation of cumulants by a single network layer, we drop the layer index l in this section. We further denote the layer input as $x \in \mathbb{R}^{N_\text{in}}$ to avoid ambiguity and assume it to be Gaussian distributed with $x \sim \mathcal{N}(\mu_x, \Sigma_x)$.

The *cumulant-generating function* $\mathcal{W}_z$ for the pre-activations $z = W x + b$ is given by

$$\mathcal{W}_z(j) = \ln \left\langle \exp\left(j^\mathsf{T} z\right) \right\rangle_z \tag{4}$$

and follows directly from its counterpart $\mathcal{W}_x$ for the layer input as

$$\begin{aligned}
\mathcal{W}_z(j) &= \ln \left\langle \exp\left(j^\mathsf{T} z\right) \right\rangle_z \\
&= \ln \left\langle \exp\left(j^\mathsf{T} W x + j^\mathsf{T} b\right) \right\rangle_x \\
&= \ln \left\langle \exp\left(\left[W^\mathsf{T} j\right]^\mathsf{T} x\right) \right\rangle_x + j^\mathsf{T} b \\
&= \mathcal{W}_x\left(W^\mathsf{T} j\right) + j^\mathsf{T} b.
\end{aligned}$$

The above result imposes relations for the cumulant $G_z^{(n)}$ of order n, yielding

$$G^{(n)}_{z,(r_1,\ldots,r_n)} = \frac{\partial^n}{\partial j_{r_1} \ldots \partial j_{r_n}} \mathcal{W}_z(j)\bigg|_{j=0} \tag{5}$$

$$= \begin{cases} \sum_{s_1} w_{r_1 s_1} G^{(1)}_{x,(s_1)} + b_{r_1} & \text{for } n = 1, \\ \sum_{s_1,\ldots,s_n} w_{r_1 s_1} \ldots w_{r_n s_n} G^{(n)}_{x,(s_1,\ldots,s_n)} & \text{else.} \end{cases} \tag{6}$$

As expected, these relations take the form of tensor transformations, given that cumulants behave like symmetric tensors.

Since the layer input x is assumed to be Gaussian distributed, we have $G_x^{(n\geq 3)} = 0$ and the above expressions simplify to

$$\mu_z = G_z^{(1)} = W\,\mu_x + b, \tag{7}$$
$$\Sigma_z = G_z^{(2)} = W\,\Sigma_x\,W^\mathsf{T}, \tag{8}$$
$$G_z^{(n\geq 3)} = 0. \tag{9}$$

If the resulting covariance matrix $\Sigma_z = W\,\Sigma_x\,W^\mathsf{T}$ is non-degenerate for $W \in \mathbb{R}^{N_\text{out}\times N_\text{in}}$ with $N_\text{out} \leq N_\text{in}$, it follows directly that the pre-activations z are also Gaussian distributed with mean and covariance as above.

If for $N_\text{out} \leq N_\text{in}$, the covariance matrix Σ_z is degenerate, or if $N_\text{out} > N_\text{in}$, the affine linear transformation $z = W\,x + b$ maps to a subspace $V \subset \mathbb{R}^{N_\text{out}}$. This subspace is spanned by the eigenvectors of $W\,\Sigma_x\,W^\mathsf{T}$ with positive eigenvalues, while all remaining eigenvectors correspond to eigenvalue zero. In consequence, the probability of any point z that does not lie within this subspace is zero and the average $\langle\circ\rangle_z$ is effectively determined with respect to V as

$$\langle\circ\rangle_z = \int_{\mathbb{R}^{N_\text{out}}} \mathrm{d}z\ \circ\ p_z(z) \tag{10}$$
$$= \int_V \mathrm{d}z\ \circ\ \tilde{p}_z(z). \tag{11}$$

Here, $\tilde{p}_z(z)$ refers to the marginal distribution of $p_z(z)$ with respect to the orthogonal complement $V^\perp$ and is given by a Gaussian distribution. In the following, we write $z \sim \mathcal{N}(\mu_z, \Sigma_z)$ in all cases but keep this property in mind.

In the case of the quadratic activation function $\phi(z)$, the cumulant-generating function $\mathcal{W}_y$ of the post-activations y is given by

$$\mathcal{W}_y(j) = \ln \left\langle \exp\left(j^\mathsf{T} y\right)\right\rangle_y \tag{12}$$
$$= \ln \left\langle \exp\left(j^\mathsf{T} z + \alpha \sum_r j_r\, z_r^2\right)\right\rangle_{z\sim\mathcal{N}(\mu_z,\,\Sigma_z)}. \tag{13}$$

By defining $J_{rs}\,(j) \coloneqq 2\alpha\, j_r\, \delta_{rs}$, we can rewrite

$$\begin{aligned}
\exp\left(\mathcal{W}_y(j)\right) &= \left\langle \exp\left(j^\mathsf{T} z + \frac{1}{2}\, z^\mathsf{T} J z\right)\right\rangle_{z\sim\mathcal{N}(\mu_z,\,\Sigma_z)} \\
&= \left((2\pi)^N \det(\Sigma_z)\right)^{-\frac{1}{2}} \int \mathrm{d}z\ \exp\left(j^\mathsf{T} z + \frac{1}{2}\, z^\mathsf{T} J z\right) \exp\left(-\frac{1}{2}(z-\mu_z)^\mathsf{T}\Sigma_z^{-1}(z-\mu_z)\right) \\
&= \left((2\pi)^N \det(\Sigma_z)\right)^{-\frac{1}{2}} \int \mathrm{d}z\ \exp\left(-\frac{1}{2}\, z^\mathsf{T}(\Sigma_z^{-1} - J)z + \left(j + \Sigma_z^{-1}\mu_z\right)^\mathsf{T} z - \frac{1}{2}\mu_z^\mathsf{T}\Sigma_z^{-1}\mu_z\right) \\
&= \det\left(1 - \Sigma_z J\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\mu_z^\mathsf{T}\Sigma_z^{-1}\mu_z\right) \left\langle \exp\left(\left(j + \Sigma_z^{-1}\mu_z\right)^\mathsf{T} z\right)\right\rangle_{z\sim\mathcal{N}\left(0,\,(\Sigma_z^{-1} - J)^{-1}\right)}
\end{aligned}$$

which gives

$$\begin{aligned}\mathcal{W}_y(j) = &-\frac{1}{2}\ln\det\left(1-\Sigma_z J\right) - \frac{1}{2}\mu_z^\mathsf{T}\Sigma_z^{-1}\mu_z \\ &+ \ln\,\left\langle \exp\left(\left(j+\Sigma_z^{-1}\mu_z\right)^\mathsf{T} z\right)\right\rangle_{z\sim\mathcal{N}\left(0,\,(\Sigma_z^{-1}-J)^{-1}\right)}.\end{aligned} \tag{14}$$

The latter term corresponds to the cumulant-generating function $\mathcal{W}_z\left(\tilde{j}\right)$ of a Gaussian distribution for $\tilde{j} = j + \Sigma_z^{-1}\mu_z$, so that we obtain

$$\begin{aligned}\mathcal{W}_y(j) = &-\frac{1}{2}\ln\det\left(1-\Sigma_z J\right) - \frac{1}{2}\mu_z^\mathsf{T}\Sigma_z^{-1}\mu_z \\ &+ \frac{1}{2}\left(j+\Sigma_z^{-1}\mu_z\right)^\mathsf{T}\left(\Sigma_z^{-1}-J\right)^{-1}\left(j+\Sigma_z^{-1}\mu_z\right).\end{aligned} \tag{15}$$

From the series expansion of the logarithm $\ln(1-x) = -\sum\limits_{r=1}^{\infty}\frac{x^r}{r}$, it follows

$$-\frac{1}{2}\ln\det\left(1-\Sigma_z J\right) = -\frac{1}{2}\operatorname{tr}\ln\left(1-\Sigma_z J\right) \tag{16}$$

$$= \frac{1}{2}\operatorname{tr}\sum_{r=1}^{\infty}\frac{\left(\Sigma_z J\right)^r}{r}. \tag{17}$$

Furthermore, we can write

$$\left(\Sigma_z^{-1}-J\right)^{-1} = \Sigma_z\left(1-J\,\Sigma_z\right)^{-1} \tag{18}$$

$$= \Sigma_z\sum_{r=0}^{\infty}\left(J\,\Sigma_z\right)^r. \tag{19}$$

Combining the above equations and taking the derivative, we get

$$\begin{aligned}G^{(n)}_{y,\,(r_1,\ldots,r_n)} = \sum_{\mathcal{S}_{(r_1,\,\ldots,\,r_n)}} \Big\{&\frac{1}{2n}(2\alpha)^n\,\Sigma_{z,\,r_1r_2}\ldots\Sigma_{z,\,r_nr_1} \\ &+ \frac{1}{2}\,(2\alpha)^{n-2}\,\Sigma_{z,\,r_1r_2}\ldots\Sigma_{z,\,r_{n-1}r_n}\,\delta_{n\neq 1} \\ &+ \frac{1}{2}\,(2\alpha)^n\,\mu_{z,\,r_1}\,\Sigma_{z,\,r_1r_2}\ldots\Sigma_{z,\,r_{n-1}r_n}\,\mu_{z,\,r_n} \\ &+ (2\alpha)^{n-1}\,\mu_{z,\,r_1}\,\Sigma_{z,\,r_1r_2}\ldots\Sigma_{z,\,r_{n-1}r_n}\Big\},\end{aligned} \tag{20}$$

where the sum comprises all permutations of the indices $r_1, \ldots, r_n$. For mean and covariance of the post-activations y, we obtain

$$\mu_{y,\,r} = \mu_{z,\,r} + \alpha\,(\mu_{z,\,r})^2 + \alpha\,\Sigma_{z,\,rr}, \tag{21}$$

$$\begin{aligned}\Sigma_{y,\,rs} = &\Sigma_{z,\,rs} + 2\,\alpha\,\left(\mu_{z,\,r}\,\Sigma_{z,\,rs} + \mu_{z,\,s}\,\Sigma_{z,\,sr}\right) \\ &+ 2\,\alpha^2\,\Sigma_{z,\,rs}\,\Sigma_{z,\,sr} + 4\,\alpha^2\,\mu_{z,\,r}\,\Sigma_{z,\,rs}\,\mu_{z,\,s}.\end{aligned} \tag{22}$$

In particular, these expressions show that the quadratic activation function $\phi(z)$ couples mean and covariance in a non-linear manner.

2.2 Feynman rules for a quadratic activation function

The analytical derivation of expressions for the cumulants of the layer output in the case of a quadratic activation function relies entirely on the fact that we perform an average with respect to a Gaussian distribution. For arbitrary distributions, we need to apply a perturbative approach using Feynman diagrams. This method furthermore allows us to gain a different understanding regarding the functional structure of the obtained expressions. Feynman diagrams are composed of various graphical elements corresponding to the occurring algebraic terms, while their assembly is governed by several rules regarding the topological features of the diagrams [14].

The graphical elements that are used in this thesis are shown in Tab. 1. As an addition to commonly used notations, we introduce a two-point interaction vertex ('α-vertex') to account for the term $\alpha\, j_r\, z_r^2$ in (13). Accordingly, this two-point interaction vertex directly depends on the external source j.

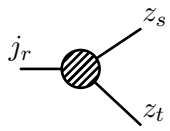
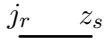
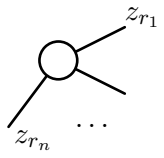
meaning	algebraic term	graphical representation
α-vertex two-point interaction vertex with fixed external line	$\alpha\, j_r\, \delta_{rs}\, \delta_{rt}$	
external line	$j_r\, \delta_{rs}$	
cumulant vertex with n internal lines	$G^{(n)}_{z,\,(r_1,\ldots,r_n)}$	

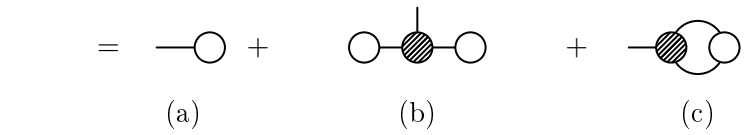
Table 1: Diagrammatic rules for the perturbative expansion of $\mathcal{W}_y\,(j)$ with $y = z + \alpha\, z^2$.

Feynman diagrams for Gaussian distributed layer input

To provide an insight into the use of Feynman diagrams, we first examine how the expressions in (21) and (22) from the previous section can be derived using this method. For a cumulant $G^{(n)}$ of order n, we need to determine all diagrams with n external lines. External lines occur either independently or originate from an α-vertex. Furthermore, they always need to be connected to a cumulant vertex, but cannot be connected to one another. According to the *linked cluster theorem*, only connected diagrams are contributing to cumulants, thus reducing the number of diagrams that need to be considered [14].

By applying these rules, we get the following diagrammatic contributions to mean and covariance of the post-activations y:

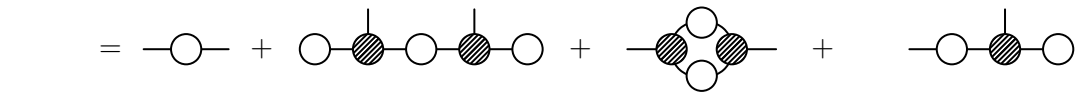
$$\mu_{y,r} = \mu_{z,r} + \alpha\,(\mu_{z,r})^2 + \alpha\,\Sigma_{z,rr}$$



(a) (b) (c)

(23)

$$\Sigma_{y,rs} = \Sigma_{z,rs} + 4\,\alpha^2\,\mu_{z,r}\,\Sigma_{z,rs}\,\mu_{z,s} + 2\,\alpha^2\,(\Sigma_{z,rs})^2 + 2\,\alpha\,\Sigma_{z,rs}\,(\mu_{z,r}+\mu_{z,s})$$



(a) (b) (c) (d)

(24)

The diagrams (23)(a) and (24)(a) result from the linear term in the activation function $\phi(z)$ while the others result from its quadratic part. Since the pre-activations z are Gaussian distributed, only cumulant vertices with one or two internal lines appear here. As expected, the corresponding algebraic expressions yield the same results for mean and covariance of the layer output as in the previous section.

From the fact that we are able to derive exact expressions for cumulants of arbitrary order in Section 2.1, it follows that we can perform a resummation of certain diagram components. This yields a compact diagrammatic formulation of the expressions in (20) for the cumulants $G_y^{(n)}$ of the post-activations y. For details, see Appendix A.3.

2.3 Gaussian description of network mapping

With the results derived in Section 2.1, we can describe the network as a non-linear mapping of mean and covariance of the input data to mean and covariance of the network output. This description builds on the disorder average (see Appendix A.1) according to which higher order cumulants beyond mean and covariance that are generated at intermediate layers scale down with the network width N. This result assumes the network parameters to be independent and identically distributed. In the case of untrained networks, we can thus approximate the distribution at intermediate layers as well as the distribution of the network output as Gaussian.

Based on these results, we determine theoretical values for mean and covariance of the network output z^L iteratively as

$$\left.\begin{aligned} \mu_{z^l,r} &= (W^l\,\mu_{y^l} + b^l)_r, \\ \mu_{y^{l+1},r} &= \mu_{z^l,r} + \alpha\,(\mu_{z^l,r})^2 + \alpha\,\Sigma_{z^l,rr} \end{aligned}\right\} \quad \text{for } l = 0,\dots,L-1 \tag{25}$$

$$\mu_{z^L,r} = (W^L\,\mu_{y^L} + b^L)_r, \tag{26}$$

$$\left.\begin{aligned} \Sigma_{z^l,rs} &= \left(W^l\,\Sigma_{y^l}\,(W^l)^\mathsf{T}\right)_{rs}, \\ \Sigma_{y^{l+1},rs} &= \Sigma_{z^l,rs} + 2\,\alpha\,\left(\mu_{z^l,r}\,\Sigma_{z^l,rs} + \mu_{z^l,s}\,\Sigma_{z^l,sr}\right) \\ &\quad + 2\,\alpha^2\,(\Sigma_{z^l,rs})^2 + 4\,\alpha^2\,\mu_{z^l,r}\,\Sigma_{z^l,rs}\,\mu_{z^l,s} \end{aligned}\right\} \quad \text{for } l = 0,\dots,L-1 \tag{27}$$

$$\Sigma_{z^L,rs} = \left(W^L \, \Sigma_{y^L} \, (W^L)^\mathsf{T}\right)_{rs}. \tag{28}$$

The resulting values $\mu_{z^L,\text{theo.}} = \hat{f}_\mu\big(\mu_x, \Sigma_x; \theta\big)$ and $\Sigma_{z^L,\text{theo.}} = \hat{f}_\Sigma\big(\mu_x, \Sigma_x; \theta\big)$ for mean and covariance of the network output are given as functions of mean μ_x and covariance Σ_x of the network input as well as the network parameters θ.

For a network of depth $L = 1$ with a single non-linear layer, the above expressions are exact whereas for $L > 1$, they yield approximations based on the disorder average. Therefore, their accuracy is expected to increase with the network width N.

2.4 Results for untrained networks

To assess the accuracy of the above approximations, we compare the empirical distribution of the network output with the resulting theoretical values for networks of different depth L and width N. In this section, we consider the case of untrained networks. As input we use data samples drawn from a two-dimensional Gaussian distribution with

$$\mu_x = \begin{pmatrix} 0.4 \\ 0.4 \end{pmatrix} \quad \text{and} \quad \Sigma_x = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.05 \end{pmatrix}, \tag{29}$$

so that $d_\text{in} = 2$. The distribution of data samples is visualized in Fig. 2(a).

We choose a strength of $\alpha = 0.5$ for the quadratic part of the activation function $\phi(z)$. The network parameters $\theta = \{W^l, b^l\}_{l=0,\ldots,L}$ are initialized according to the assumptions made for the disorder average (see Appendix A.1). Therefore, these are drawn according to $w^l_{rs} \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(0, \frac{\sigma_w^2}{N_{l-1}}\right)$ and $b^l_r \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(0, \sigma_b^2\right)$ for all layers l, where we set $\sigma_w = \sigma_b = 0.75$.

For a network of depth $L = 1$, the theoretical values for mean and covariance are $\mu_{z^L,\text{theo.}} = -1.379$ and $\Sigma_{z^L,\text{theo.}} = 0.264$ (see Eq. (25)-(28) in Section 2.3). We calculate empirical estimates for these quantities using the network output of $D = 10^4$ data samples, yielding $\mu_{z^L,\text{emp.}} = -1.383 \pm 0.005$ and $\Sigma_{z^L,\text{emp.}} = 0.268 \pm 0.004$ which lie in the range of one standard error from the theoretical values. Furthermore, the resulting probability density function shown in Fig. 2(b) agrees well with its empirical counterpart, which we obtain as a normalized histogram of the network output for this dataset. These results are in line with expectations, since for a single non-linear layer the expressions for mean and covariance are exact.

We want to measure the overall accuracy of the probability density functions obtained from our theory relative to the empirical counterparts. In this context, a comparison of the absolute values of mean and covariance can only serve as a check for the validity of the expressions in Section 2.3. However, such a comparison does not capture properties of the distribution that are contained in higher order cumulants. Furthermore, it is not clear how differences in higher order cumulants are to be interpreted or how such differences for different orders can be combined to yield a sensible comparison with respect to the overall distribution.

Therefore, we use the *Kullback-Leibler divergence* as an accuracy measure, which was first introduced by Kullback and Leibler in 1951 [21]. For two probability density functions $\rho(z)$ and $\eta(z)$, it is given as

$$D_{KL}(\rho\|\eta) = \int \mathrm{d}z \, \rho(z) \left[\ln \rho(z) - \ln \eta(z)\right] \tag{30}$$

$$= -H(\rho) + H(\rho, \eta), \tag{31}$$

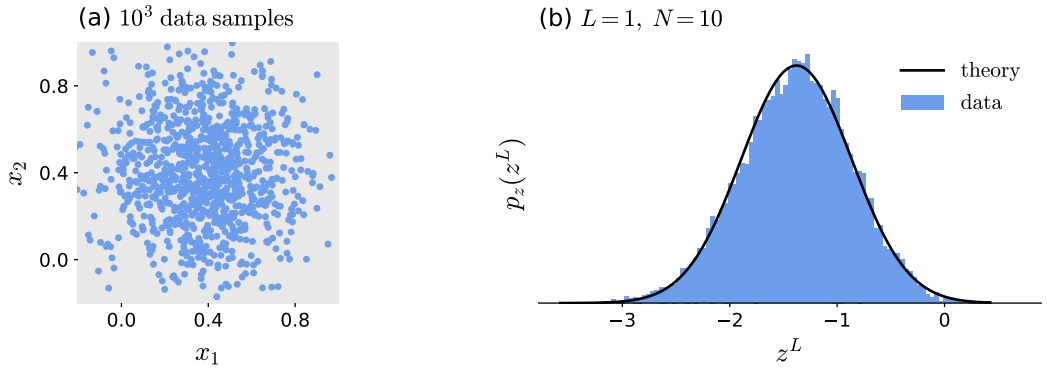


Figure 2: **(a)** Distribution of network input $x \in \mathbb{R}^2$. Data samples are drawn from the Gaussian distribution defined in (29). For illustrative purpose, we show a subset of 10^3 data samples of the full dataset of size $D = 10^4$ and constrain the depicted area in such a way that the majority of data samples is shown. **(b)** Distribution of the network output $z^L \in \mathbb{R}$ for a network of depth $L = 1$ and width $N = 10$. A normalized histogram (blue) of the network output for the full dataset of size $D = 10^4$ acts as an empirical estimate of the probability density function. Its theoretical counterpart is given as a Gaussian distribution (black) with mean and covariance calculated according to Section 2.3.

where $H(\rho)$ is the *entropy* of $\rho(z)$ and $H(\rho, \eta)$ is the *cross-entropy* between $\rho(z)$ and $\eta(z)$. The Kullback-Leibler divergence can be interpreted as a measure of the expected number of additional bits needed to encode samples drawn from $\rho(z)$ when using a code optimized for $\eta(z)$, in comparison to a code optimized for $\rho(z)$ [22]. In that context, $\eta(z)$ is considered to be an approximation of the true distribution $\rho(z)$.

Since we want to assess the accuracy with regard to our theoretical description, we choose $\rho(z) = p_{z^L,\,\text{emp.}}(z)$ and $\eta(z) = p_{z^L,\,\text{theo.}}(z)$. To obtain a dimensionless quantity, we further divide the Kullback-Leibler divergence by the entropy of $p_{z^L,\,\text{emp.}}$ and denote the resulting quantity as

$$\hat{D}_{KL}(p_{z^L,\,\text{emp.}} \| p_{z^L,\,\text{theo.}}) = \frac{D_{KL}(p_{z^L,\,\text{emp.}} \| p_{z^L,\,\text{theo.}})}{H(p_{z^L,\,\text{emp.}})}. \tag{32}$$

Accordingly, it describes the relative increase in the expected number of additional bits needed. In the above example for $L = 1$, we have $\hat{D}_{KL}(p_{z^L,\,\text{emp.}} \| p_{z^L,\,\text{theo.}}) = 0.2\%$, quantifying the good agreement between theoretical and empirical distribution in Fig. 2(b).

Since for deeper networks the expressions for mean and covariance of the network output represent approximations based on the disorder average, we expect larger deviations between theory and simulation. As an example, we consider a network of depth $L = 4$ and width $N = 10$. The obtained results are given in Tab. 2. While the mean values still lie in the range of one standard error of one another, the covariance values differ in the order of two standard errors. When comparing the probability density functions obtained from theory and simulation in Fig. 3(a), the histogram is noticeably skewed, which cannot be captured by a Gaussian description of the network output corresponding to theory. This effect results from higher order cumulants beyond mean and covariance becoming non-negligible and is also reflected in the relative Kullback-Leibler divergence $\hat{D}_{KL}(p_{z^L,\,\text{emp.}} \| p_{z^L,\,\text{theo.}}) = 1.0\%$.

According to the disorder average, we expect these deviations to decrease for broader network architectures. For a network of depth $L = 4$ and $N = 100$, the theoretical values

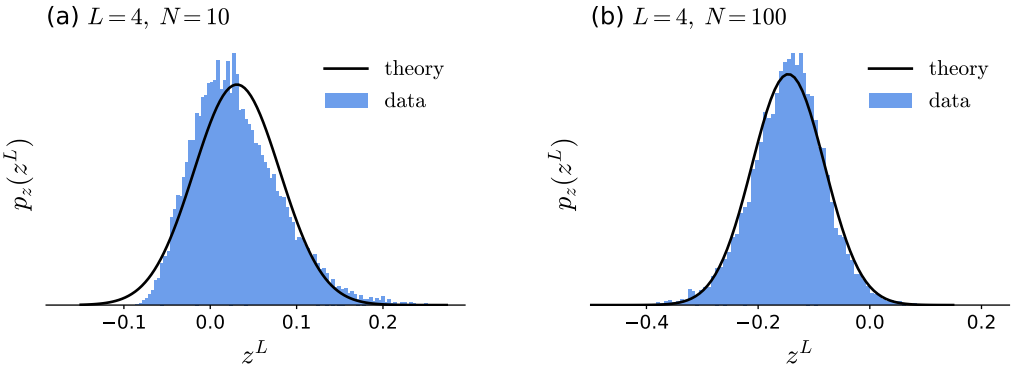


Figure 3: Distribution of the network output $z^L \in \mathbb{R}$ for networks of depth $L = 4$ and width $N = 10$ (a) or $N = 100$ (b). A normalized histogram (blue) of the network output for a dataset of size $D = 10^4$ acts as an empirical estimate of the probability density function. Its theoretical counterpart is given as a Gaussian distribution (black) with mean and covariance calculated according to Section 2.3.

Network architecture	$\mu_{z^L,\text{theo.}}$	$\mu_{z^L,\text{emp.}}$	$\Sigma_{z^L,\text{theo.}}$	$\Sigma_{z^L,\text{emp.}}$
$L = 1,\ N = 10 \quad [10^0]$	-1.379	-1.383 ± 0.005	0.264	0.268 ± 0.004
$L = 4,\ N = 10 \quad [10^{-3}]$	30.8	31.2 ± 0.5	2.48	2.54 ± 0.04
$L = 4,\ N = 100 \quad [10^{-3}]$	-146.0	-146.2 ± 0.6	4.23	4.31 ± 0.07

Table 2: Theoretical values and empirical estimates for mean and covariance of the network output z^L for different network architectures.

for mean and covariance of the network output (see Tab. 2) show a decrease in deviation between theory and simulation. In particular, the skewness of the histogram in Fig. 3(b) is significantly reduced. The theoretical and empirical curves for the probability density function show an improved agreement both qualitatively and quantitatively with $\hat{D}_{KL}(p_{z^L,\text{emp.}} \| p_{z^L,\text{theo.}}) = 0.5\%$.

The above results apply to instances of different network architectures for a particular set of network parameters θ. To check whether the discussed effects appear consistently, we determine the relative Kullback-Leibler divergence $\hat{D}_{KL}(p_{z^L,\text{emp.}} \| p_{z^L,\text{theo.}})$ averaged over networks initialized on $n_{\text{seeds}} = 10^3$ different seeds. For consistency, in all cases the same dataset as above is used as network input. In line with expectations, the deviations increase with the network depth L while decreasing with the network width N (see Fig. 4(a)). This effect is consistent across different network realizations as can be seen from the magnitude of the obtained standard errors shown in Fig. 4(b) and (c). The results for depth $L = 4$ suggest that the relative Kullback-Leibler divergence approaches a plateau for larger network widths N. This would translate to the accuracy of the expressions for mean and covariance of the network output being limited for deeper networks, presumably due to the approximations made.

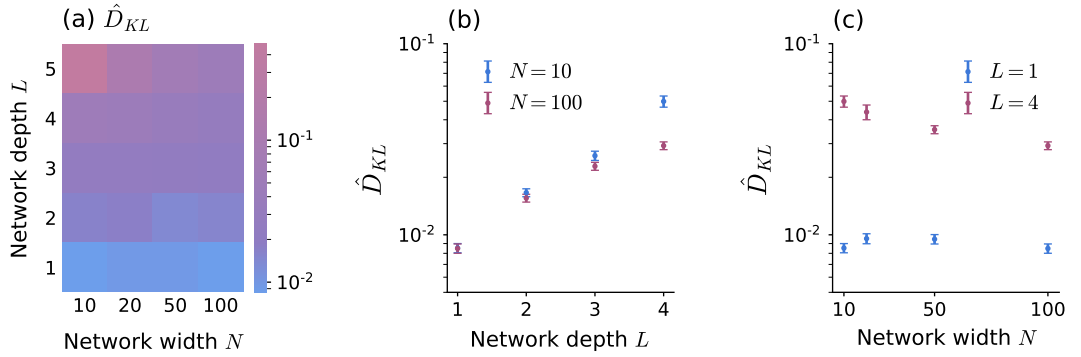


Figure 4: Relative Kullback-Leibler divergence for different network architectures, averaged over networks that are initialized on $n_{\text{seeds}} = 10^3$ different seeds. In (a), the resulting mean values are depicted. In (b) and (c), mean and corresponding standard error are shown for varying network depth L and width N, respectively.

3 Binary classification on XOR

In the previous chapter, we investigated how mean and covariance of the input distribution are transformed by neural networks and assessed the accuracy of our theoretical description for networks with randomly sampled parameters. As a next step, we apply our method to networks that have been trained to solve a given task, with the network parameters having been adjusted accordingly.

In this chapter, we use an adaptation of the *exclusive or* (XOR) problem, a classical problem in machine learning. In its original formulation, the objective is to predict the output of the XOR logic gate given two binary inputs [23]. Since this problem is not linearly separable, it requires the use of a non-linear activation function. Therefore, it allows us to study the influence of a non-linear term on the solution strategy chosen by the network, as opposed to using purely linear networks.

To stay in the context of continuous probability distributions, we use real-valued instead of binary inputs and formulate the XOR problem as a classification task (see Section 3.1). We then study the accuracy of the Gaussian network description in Section 2.3 for networks trained on this task (see Section 3.2).

3.1 Representation of the XOR problem by a Gaussian mixture model

We construct a dataset for the XOR problem using samples drawn from a *Gaussian mixture model*. In general, the probability density function of a Gaussian mixture model for $x \in \mathbb{R}^{d_{\text{in}}}$ is given by a superposition of M Gaussian distributions

$$p_{\text{GMM}}(x) = \sum_{m=1}^{M} \pi_m \mathcal{N}(x|\mu_x^m, \Sigma_x^m), \tag{33}$$

where $\mathcal{N}(x|\mu_x^m, \Sigma_x^m)$ denotes a Gaussian probability density function with mean μ_x^m and covariance Σ_x^m for $m = 1, \ldots, M$. Each *mixture component* m is scaled by a *mixture weight* π_m. The mixture weights obey

$$\sum_{m=1}^{M} \pi_m = 1, \tag{34}$$

to ensure that the resulting probability density function is properly normalized [6].

The XOR problem is formulated as a binary classification task, meaning the data samples belong to two classes. We assign each mixture component m a *class label* $t^k \in \{0, 1\}$ belonging to class k. Each data sample $x^{(d)}$ is then assigned a *target label* $t^{(d)} \in \{0, 1\}$ corresponding to the mixture component it is drawn from. Here, the superscript d refers to the sample index.

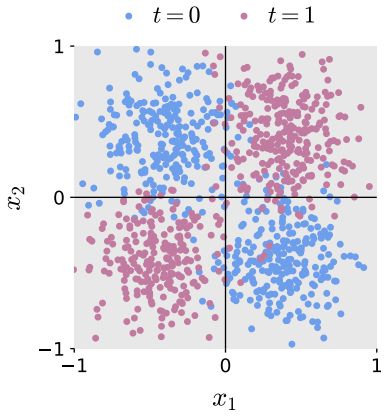


Figure 5: Distribution of data samples $x \in \mathbb{R}^2$ for the XOR problem. Data samples are drawn from the Gaussian mixture model defined in (35)-(37) with class labels $t^k \in \{0, 1\}$ (blue, red) assigned accordingly. For illustrative purpose, we show a subset of 10^3 data samples of the full dataset of size $D = 10^4$ and constrain the depicted area in such a way that the majority of data samples is shown. The optimal decision boundaries for solving the task (black lines) coincide with the axes.

For our adaptation of the XOR problem, we choose $M = 4$ components and assign

$$t = 0: \qquad \mu_x^{m=1,2} = \pm \begin{pmatrix} -0.4 \\ 0.4 \end{pmatrix} \qquad \Sigma_x^{m=1,2} = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.05 \end{pmatrix}, \tag{35}$$

$$t = 1: \qquad \mu_x^{m=3,4} = \pm \begin{pmatrix} 0.4 \\ 0.4 \end{pmatrix} \qquad \Sigma_x^{m=3,4} = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.05 \end{pmatrix}, \tag{36}$$

with all components being equally weighted

$$\pi_m = \frac{1}{M} = 0.25 \quad \text{for } m = 1, \ldots, 4. \tag{37}$$

Each classwise probability density function $p_{t,x}(x)$ is then given as a superposition of the respective mixture components m'

$$p_{t,x}(x) = \sum_{m'} \pi_{m'} \mathcal{N}(x|\mu_x^{m'}, \Sigma_x^{m'}). \tag{38}$$

The resulting data distribution is visualized in Fig. 5.

Theoretical performance limit

For a network with parameters θ, we aim at maximizing the *classification accuracy* or *performance* P_θ. This quantity is defined as the amount of correctly classified data samples relative to the total number of data samples

$$P_\theta = \frac{\#\text{ correctly classified data samples}}{\#\text{ data samples}}. \tag{39}$$

For our adaptation of the XOR problem, the overlap between the probability density functions of different mixture components implies a theoretical upper limit of the

achievable performance. This limit serves as a baseline when training any network on a dataset that is drawn from the distribution defined in (35)-(37). Given the classwise probability density functions $p_{t,\,x}(x)$, the classification accuracy is maximized by assigning any point $x \in \mathbb{R}^{d_\text{in}}$ to the class label corresponding to the higher probability. Consequently, the *optimal decision boundaries* for a binary classification task are defined by the condition $p_{t=0,\,x}(x) = p_{t=1,\,x}(x)$. Due to the symmetry of the problem, the optimal decision boundaries for our adaptation of the XOR problem coincide with the axes, corresponding to $x_i = 0$ for $i = 1, 2$, as illustrated in Fig. 5.

We determine the expected value of the theoretical performance limit using the cumulative distribution function $\Phi_{0,\,1}(x)$ of the standard normal distribution and start by considering each component separately. Any point drawn from the component $m = 1$ that lies either in the lower left or upper right quadrant will be misclassified. The probability mass of component $m = 1$ located in the other two areas yields the minimal expected classification error $\varepsilon_{m=1}$ for this component. In each dimension, the probability mass located beyond a single axis is given by $1 - \Phi_{0,\,1}\left(\frac{0 \pm 0.4}{\sqrt{0.05}}\right) = 3.68\%$. Taking into account the geometry of the problem, we thus get

$$\varepsilon_{m=1} = 2 \cdot \left[1 - \Phi_{0,\,1}\left(\frac{0 \pm 0.4}{\sqrt{0.05}}\right)\right] \cdot \Phi_{0,\,1}\left(\frac{0 \pm 0.4}{\sqrt{0.05}}\right) \approx 7.1\%.$$

Since the distribution is symmetric and all components are equally weighted, this result directly corresponds to the *minimal expected cumulative classification error* $\mathcal{E}_\text{min}$, yielding a maximum of $P_\text{max} = 92.9\%$ for the achievable performance.

3.2 Results for trained networks

During training, we minimize an estimator of the classification error $\mathcal{E}_\theta$. For network input x, we use the mean squared error between the network output $g_\theta(x)$ and the corresponding target label t as *loss function* so that $\ell(x,\, t;\, \theta) = \|g_\theta(x) - t\|^2$. To improve training dynamics, we randomly subdivide the full training dataset into disjoint subsets or *batches* $\{\mathcal{B}_\beta\}_{\beta=1,\ldots,D_\text{train}/B}$ of B data samples so that $\mathcal{B}_\beta = \{(x^{(b)}, t^{(b)})\}_{b=1,\ldots,B} \subset \{(x^{(d)}, t^{(d)})\}_{d=1,\ldots,D_\text{train}}$. Here, B and D_train denote the *batch size* and the size of the training dataset, respectively. In each optimization step, the *network loss* $\mathcal{L}$ is determined as the empirical average of the loss function $\ell(x,\, t;\, \theta)$ with respect to a batch $\mathcal{B}_\beta$, which gives

$$\mathcal{L}\big(\{(x^{(b)}, t^{(b)})\}_{b=1,\ldots,B};\, \theta\big) = \frac{1}{B}\sum_{b=1}^{B} \ell(x^{(b)},\, t^{(b)};\, \theta). \tag{40}$$

Within one training epoch, the full training dataset subdivided into batches $\mathcal{B}_\beta$ is presented once to the network.

To assess the performance P_θ of a network, we need a transformation $g_\theta(x) \mapsto \{0, 1\}$ for $x \in \mathbb{R}^{d_\text{in}}$ because the network output $g_\theta(x) \in \mathbb{R}$ is continuous while the class labels are discrete. We use a step function to assign a predicted class label $\hat{t} \in \{0, 1\}$ according to

$$\hat{t} = \begin{cases} 0 & g_\theta(x) < 0.5, \\ 1 & \text{else.} \end{cases} \tag{41}$$

The performance P_θ of a particular network with parameters θ is then evaluated using a test dataset of size D_eval containing data samples previously unknown to the network.

The performance is determined as

$$P_\theta = \frac{1}{D_\text{eval}} \sum_{d=1}^{D_\text{eval}} \delta_{t^{(d)}\,\hat{t}^{(d)}}, \tag{42}$$

with $t^{(d)}$ and $\hat{t}^{(d)}$ being the target label and the predicted class label of a data sample, respectively.

We train all networks on a training dataset of $D_\text{train} = 10^4$ data samples using the common optimization algorithm ADAM with default parameters from the Python package PYTORCH (version 1.4.0) [24]. Furthermore, we choose a batch size of $B = 10$ and train for $n_\text{epochs} = 2$ epochs. Performance evaluation is based on a test dataset of size $D_\text{eval} = 10^4$. Throughout this thesis, we use the same training parameters unless specified otherwise.

To describe the distribution of the network output $g_\theta(x) = z^L$, we can use our knowledge regarding the input distribution. The probability π_m of a data sample $x^{(d)}$ belonging to component m remains unchanged when processing this data sample by the network. Therefore, it is

$$p_{z^L}(z^L) = \sum_{m=1}^{M} \pi_m \, p_{m,\,z^L}(z^L), \tag{43}$$

where $p_{m,\,z^L}(z^L)$ refers to the output distribution of component m. Thus, the output distribution is given by a general, non-Gaussian mixture model.

Since the input distribution of each component is Gaussian, we can use the Gaussian description of the network mapping in Section 2.3 to obtain

$$\begin{aligned} p_{z^L}(z^L) &= \sum_{m=1}^{M} \pi_m \, p_{m,\,z^L}(z^L) && (44)\\ &\approx \sum_{m=1}^{M} \pi_m \, \mathcal{N}_m(z^L | \mu_{z^L}^m,\, \Sigma_{z^L}^m) && (45)\\ &=: p_{\text{GMM},\,z^L}(z^L), && (46) \end{aligned}$$

where we trace the transformation of mean and covariance separately for each component. Empirical estimates of the output distribution are again obtained as histograms of the network output for data samples of the test dataset.

We first consider a network of depth $L = 1$ and width $N = 10$ for which we obtain a classification accuracy of $P_\theta = 91.3\%$. We exemplarily show the theoretical and empirical curves of the components $m = 1$ and $m = 3$ belonging to $t = 0$ and $t = 1$, respectively, in Fig. 6(a). The corresponding histograms are noticeably skewed, suggesting that higher order cumulants become non-negligible. In Section 2.4, this effect occurs to such an extent only for deeper networks while here it already appears for shallow networks. This is probably caused by network training introducing dependencies among network parameters θ so that the assumptions of the disorder average are only partially fulfilled. The other two components behave similarly (see Appendix A.5).

Furthermore, we look at the classwise output distributions $p_{t,\,z^L}(z^L)$ which are obtained by superposition of the respective components. These are depicted in Fig. 6(b) along with the overall output distribution. The gray dashed line indicates the threshold of the step function used for label assignment. Correspondingly, the overlap of the classwise distributions represents the classification error $\mathcal{E}_\theta$. The agreement between theory

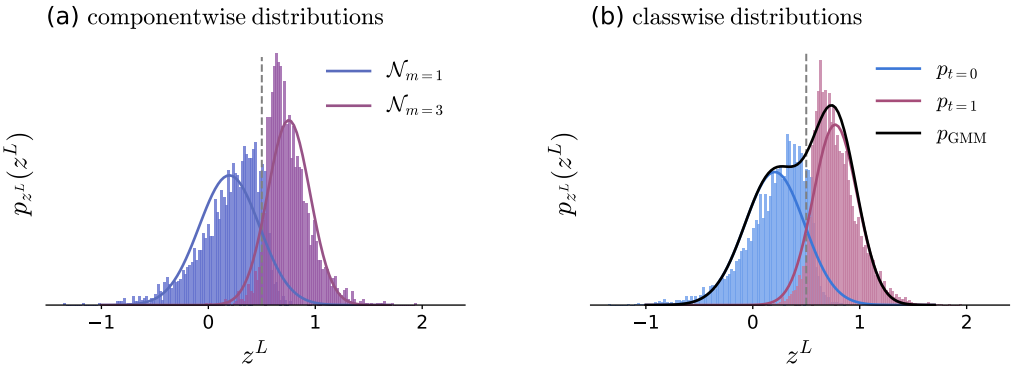


Figure 6: Distribution of the network output $z^L \in \mathbb{R}$ for a network of depth $L = 1$ and width $N = 10$ that is trained to solve the XOR problem. Normalized histograms of the network output for a test dataset of size $D_{\mathrm{eval}} = 10^4$ act as empirical estimates of the probability density function. Their theoretical counterparts are given by Gaussian distributions (solid lines) with mean and covariance calculated according to Section 2.3. In (a), we show the individual distributions for the components $m = 1$ and $m = 3$ belonging to $t = 0$ and $t = 1$, respectively. In (b), the classwise distributions $p_{t=0,\,1}$ and the overall distribution p_{GMM} are depicted.

and simulation is still reasonable, yielding a value of $\hat{D}_{KL}(p_{z^L,\,\mathrm{emp.}} \| p_{\mathrm{GMM},\,z^L,\,\mathrm{theo.}}) = 1.3\%$ with respect to the overall distribution.

For a network of depth $L = 4$ and width $N = 10$, the resulting classwise and overall distributions become significantly more irregular, as illustrated in Fig. 7(a). This can only be partially reflected by the theoretical curves, thus yielding an increased relative Kullback-Leibler divergence of $\hat{D}_{KL}(p_{z^L,\,\mathrm{emp.}} \| p_{\mathrm{GMM},\,z^L,\,\mathrm{theo.}}) = 4.7\%$. The increased network depth yields a minor performance improvement with $P_\theta = 91.6\%$. Increasing the network width to $N = 100$ does not result in a distinct qualitative improvement of the agreement between theoretical and empirical curves, as shown in Fig. 7(b). On a quantitative level, we get a slight decrease in deviations with $\hat{D}_{KL}(p_{z^L,\,\mathrm{emp.}} \| p_{\mathrm{GMM},\,z^L,\,\mathrm{theo.}}) = 3.6\%$. Furthermore, we achieve a performance of $P_\theta = 91.1\%$. In both examples, the deviations between theory and simulation are larger compared to networks with randomly sampled parameters in Section 2.4. In the latter case, the approximation error is caused by the description of the data distribution at intermediate layers as Gaussian. Here, the further increase in deviations is presumably caused by dependencies among network parameters which are introduced during training.

As for untrained networks, we check whether these effects appear consistently over different training runs. To this end, we calculate both achieved performance and relative Kullback-Leibler divergence averaged over networks that are initialized on $n_{\mathrm{seeds}} = 10^2$ different seeds and subsequently trained as described above. For consistency, the same training and test datasets are used. In Fig. 8(a), we see that networks of different depth L and width N perform similarly well relative to the theoretical performance limit P_{max}. The corresponding standard errors show that this result is consistent across different network initializations.

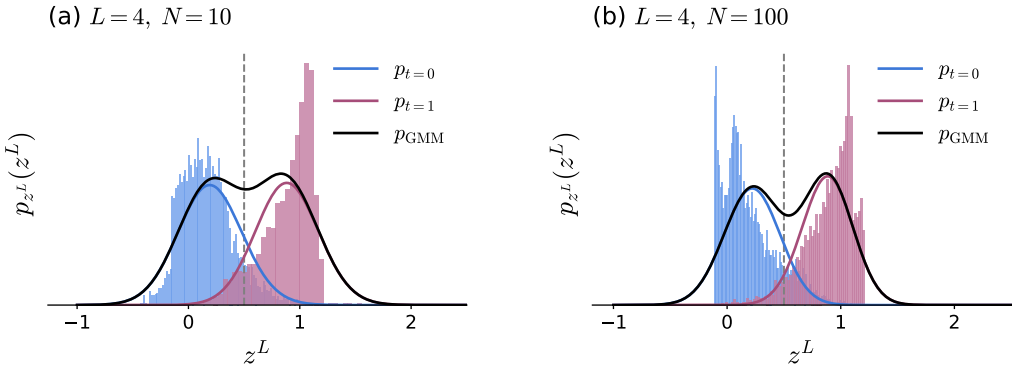


Figure 7: Distribution of the network output $z^L \in \mathbb{R}$ for networks of depth $L = 4$ and width $N = 10$ (a) or $N = 100$ (b) that are trained to solve the XOR problem. Normalized histograms of the network output for a test dataset of size $D_{\text{eval}} = 10^4$ act as empirical estimates of the probability density function. Their theoretical counterparts are given by Gaussian distributions (solid lines) with mean and covariance calculated according to Section 2.3. We show both the classwise distributions $p_{t=0,1}$ as well as the overall distribution p_{GMM}.

In line with expectations, the deviations between theory and simulation increase with the network depth L as illustrated in Fig. 8(b). However, this trend is less pronounced and less strictly adhered to compared to the untrained case (see Fig. 4 in Section 2.4). Furthermore, the deviations between theory and simulation increase with the network width N (see Fig. 8(c)) contrary to the implications of the disorder average in Appendix A.1. For broader networks, the obtained values of the relative Kullback-Leibler divergence for trained networks also approach a plateau for a value that is noticeably larger than for untrained networks. When considering Fig. 7, we see that the empirical probability density functions become more skewed with increasing network width N. Since such a behavior cannot be captured by an approximation of the output distribution as Gaussian, this explains the increase in the relative Kullback-Leibler divergence for broader networks. The dependencies among network parameters, which result from network training, presumably amplify higher order cumulants beyond mean and covariance.

In addition, the cut-off of the distribution in Fig. 7(b) might be linked to the activation function $\phi(z)$ having a global minimum and thus its image values being constrained by this lower bound. As the last operation of the network mapping amounts to an affine linear transformation, this property of the activation function $\phi(z)$ does not constitute a hard constraint. However, using this property in the solution strategy chosen by the network might be beneficial in terms of maximizing the classification performance.

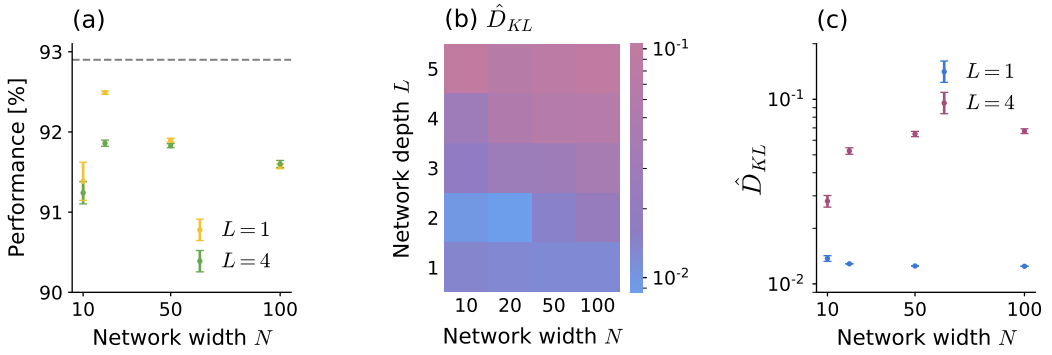


Figure 8: Achieved classification performance and relative Kullback-Leibler divergence for different network architectures, averaged over networks that are initialized on $n_{\text{seeds}} = 10^2$ different seeds and subsequently trained to solve the XOR problem. In (a), mean and corresponding standard error of the achieved performance are shown for varying network width N with the theoretical performance limit of $P_{\text{max}} = 92.9\%$ given as reference (gray dashed line). In (b), the mean value of the relative Kullback-Leibler divergence is depicted for different network architectures. In (c), mean and corresponding standard error of the relative Kullback-Leibler divergence are shown for varying network width N.

4 Network training on Gaussian statistics

In the previous chapter, we saw that deviations between theory and simulation increase for trained networks, presumably due to dependencies among network parameters generated during training. This leads back to the theoretical question of how accurate the description of the network as a non-linear mapping of mean and covariance is. In particular, we are interested in which features of a solution strategy can be captured by this perspective.

To address these questions, we study the network loss used for training which is typically given as a sample mean of the loss function with respect to a set of data samples. In the limit of infinitely many data samples, this sample mean converges to the expected value and we can relate the network loss directly to the cumulants of the network output (see Section 4.1). Furthermore, we determine an error estimate for the resulting expressions in the case of large but finite datasets, which one typically encounters in practice (see Section 4.2).

4.1 Statistical formulation of network loss

For network training, we use the mean squared error between the network output $g_\theta(x) = z^L$ and the corresponding target t as loss function $\ell(x,\, t;\, \theta) = \|g_\theta(x) - t\|^2$. We first consider the case of a dataset $\{(x^{(d)}, t^{(d)})\}_{d=1,\dots,D}$ with a single target label $t^{(d)} = t$ for $d = 1, \dots, D$, where the superscript d refers to a particular data sample. For simplicity, we take the full dataset as a single batch $\mathcal{B} = \{(x^{(d)}, t^{(d)})\}_{d=1,\dots,D}$ and write all expressions in terms of the sample index d. In the limit of infinitely many data samples, we have

$$\mathcal{L}\big(\{(x^{(d)}, t)\}_{d=1,\dots,D};\, \theta\big) = \frac{1}{D}\sum_{d=1}^{D} \ell(x^{(d)},\, t;\, \theta) \xrightarrow{D\to\infty} \big\langle \ell(x,\, t;\, \theta)\big\rangle_{x\,\sim\, p(x|t)} \tag{47}$$

$$= \big\langle \|g_\theta\,(x) - t\|^2 \big\rangle_{x\,\sim\, p(x|t)} \tag{48}$$

$$= \big\langle \|z^L - t\|^2 \big\rangle_{z^L}\,. \tag{49}$$

Calculating the average yields a formulation of the network loss in terms of mean and covariance of the network output

$$\big\langle \|z^L - t\|^2 \big\rangle_{z^L} = \Big\langle \sum_i \big(z_i^L\big)^2 - 2\, z_i^L\, t_i + t_i^2 \Big\rangle_{z^L} \tag{50}$$

$$= \sum_i \mu^2_{z^L,\,i} + \Sigma_{z^L,\,ii} - 2\,\mu_{z^L,\,i}\, t_i + t_i^2 \tag{51}$$

$$= \operatorname{tr} \Sigma_{z^L} + \|\mu_{z^L} - t\|^2. \tag{52}$$

The above expression matches the intuitive expectation for classification tasks: The output mean is mapped to the given target value while the spread of data samples in the output, its variance, is minimized.

Extending this result to K classes, we need to take into account the probability p_k for a sample to belong to class k. Assuming the attribution of class memberships to be independent from one another, we get

$$\frac{1}{D}\sum_{d=1}^{D}\|g_\theta(x^{(d)}) - t^{(d)}\|^2 = \sum_{k=1}^{K}\frac{D_k}{D}\frac{1}{D_k}\sum_{d_k=1}^{D_k}\|g_\theta(x^{(d_k)}) - t^k\|^2 \tag{53}$$

$$\xrightarrow{D\to\infty} \quad \sum_{k=1}^{K} p_k \left\langle \|z^L - t^k\|^2 \right\rangle_{z^L \sim p(z^L|t^k)} \tag{54}$$

$$= \sum_{k=1}^{K} p_k \big(\operatorname{tr} \Sigma^k_{z^L} + \|\mu^k_{z^L} - t^k\|^2\big) \tag{55}$$

$$=: \mathcal{L}_{\text{stat.}}\big(\{\mu^k_{z^L}, \Sigma^k_{z^L}\}_{k=1,\dots,K}\big), \tag{56}$$

with $\mu^k_{z^L}$ and $\Sigma^k_{z^L}$ being the classwise mean and covariance of the network output. These two quantities are generally given as functions $\mu_{z^L} = f_\mu\big(\{G_x^{(n),\,k}\}_n;\, \theta\big)$ and $\Sigma_{z^L} = f_\Sigma\big(\{G_x^{(n),\,k}\}_n;\, \theta\big)$ of the classwise cumulants of the network input and the network parameters.

With the expressions for the Gaussian description of the network mapping in Section 2.3, we can determine theoretical values for $\mu^k_{z^L,\,\text{theo.}}$ and $\Sigma^k_{z^L,\,\text{theo.}}$ as functions of the classwise means and covariances of the network input and the network parameters $\mu^k_{z^L,\,\text{theo.}} = \hat{f}_\mu\big(\mu^k_x,\, \Sigma^k_x;\, \theta\big)$ and $\Sigma^k_{z^L,\,\text{theo.}} = \hat{f}_\Sigma(\mu^k_x,\, \Sigma^k_x;\, \theta)$. Thus, we obtain

$$\mathcal{L}_{\text{stat.}} \approx \hat{\mathcal{L}}_{\text{stat.}}\big(\{\mu^k_x,\, \Sigma^k_x\}_{k=1,\dots,K};\, \theta\big). \tag{57}$$

Using this expression as network loss, we can train networks directly and exclusively on the classwise means and covariances of the network input, thereby allowing us to check the assumption that these quantities largely contain the information necessary to solve a given classification task. Since in this case we take the Gaussian description of the network mapping as surrogate during training, we can thereby test whether this description is sufficiently accurate to capture the principal features of a viable solution strategy.

Note that the first step of the derivation presented in this section is applicable to any network loss of the form

$$\mathcal{L}\big(\{(x^{(d)}, t^{(d)})\}_{1\le d\le D};\, \theta\big) = \frac{1}{D}\sum_{d=1}^{D}\ell(x^{(d)},\, t^{(d)};\, \theta). \tag{58}$$

For different choices of the loss function $\ell(x,\, t;\, \theta)$, this yields different dependencies of the network loss $\mathcal{L}_{\text{stat.}}$ on the cumulants of the network output.

4.2 Variance of network loss for finite datasets

As a measure for the accuracy of the above result in the case of large but finite datasets, we calculate the variance of the network loss as a function of the dataset size D. As before, we start with the case of a single target $t^{(d)} = t$ for $d = 1, \dots, D$ and obtain

$$\mathrm{Var}\left(\mathcal{L}\right) = \mathrm{Var}\Big(\frac{1}{D}\sum_{d=1}^{D}\big(g_\theta(x^{(d)})\big)^2 - 2\,g_\theta(x^{(d)})\,t + t^2\Big) \tag{59}$$

$$= \frac{1}{D^2}\sum_{d=1}^{D}\mathrm{Var}\Big(\big(g_\theta(x^{(d)})\big)^2 - 2\,g_\theta(x^{(d)})\,t\Big) \tag{60}$$

$$= \frac{1}{D}\,\mathrm{Var}\Big(\big(g_\theta(x)\big)^2 - 2\,g_\theta(x)\,t\Big) \tag{61}$$

$$= \frac{1}{D}\,\mathrm{Var}\Big(\big(z^L\big)^2 - 2\,z^L\,t\Big), \tag{62}$$

assuming all samples $x^{(d)}$ to be drawn independently. We further rewrite the appearing variance in terms of the cumulants of the network output z^L as

$$\mathrm{Var}\Big(\big(z^L\big)^2 - 2\,z^L\,t\Big) = \mathrm{Var}\Big(\big(z^L\big)^2\Big) + \mathrm{Var}\Big(-2\,z^L\,t\Big) + 2\,\mathrm{Cov}\Big(\big(z^L\big)^2, -2\,z^L\,t\Big) \tag{63}$$

$$= G^{(4)}_{z^L} + 2\big(\Sigma_{z^L}\big)^2 + 4\,G^{(3)}_{z^L}\left[\mu_{z^L} - t\right] + 4\,\Sigma_{z^L}\left[\mu_{z^L} - t\right]^2. \tag{64}$$

The detailed calculation can be found in Appendix A.4. For K classes, we get, analogously to before,

$$\mathrm{Var}\left(\mathcal{L}\right) = \frac{1}{D^2}\sum_{d=1}^{D}\mathrm{Var}\Big(\big(g_\theta(x^{(d)})\big)^2 - 2\,g_\theta(x^{(d)})\,t^k + \big(t^k\big)^2\Big) \tag{65}$$

$$= \frac{1}{D^2}\sum_{k=1}^{K} D_k\,\mathrm{Var}_{x|t^k}\Big(\big(g_\theta(x)\big)^2 - 2\,g_\theta(x)\,t^k\Big) \tag{66}$$

$$= \frac{1}{D}\sum_{k=1}^{K}\hat{p}_k\,\mathrm{Var}_{z^L|t^k}\Big(\big(z^L\big)^2 - 2\,z^L\,t^k\Big) \tag{67}$$

$$= \frac{1}{D}\sum_{k=1}^{K}\hat{p}_k\left(G^{(4),\,k}_{z^L} + 2\big(\Sigma^k_{z^L}\big)^2 + 4\,G^{(3),\,k}_{z^L}\left[\mu^k_{z^L} - t^k\right] + 4\,\Sigma^k_{z^L}\left[\mu^k_{z^L} - t^k\right]^2\right). \tag{68}$$

Here, $\hat{p}_k = \frac{D_k}{D}$ denotes the relative sample occurrence which acts as an empirical estimator for the probability p_k.

We see that the variance $\mathrm{Var}(\mathcal{L})$ of the network loss scales down with the size of the dataset D as expected, thus showing that the expression for the network loss $\mathcal{L}_{\mathrm{stat.}}$ in (55) continues to be a suitable approximation for large but finite datasets. In the case of trained networks, this value might decrease even further since all terms proportional to $\mu^k_{z^L}$-t^k and $\Sigma^k_{z^L}$ become small.

5 Information coding paradigms for XOR

In the previous chapter, we derived a formulation of the network loss in terms of the classwise means and covariances of the network output. By using our description of the network as a non-linear mapping of mean and covariance as described in Section 2.3, we can determine these quantities as functions of both the network parameters and the classwise means and covariances of the input data. Combining these two steps allows us to train networks directly on the classwise means and covariances of the input data.

For our adaptation of the XOR problem in Section 3.1, the information regarding class membership is encoded in the classwise covariances while the means are identical for both classes. When training on the classwise means and covariances, networks thus employ a form of *covariance coding* to find a viable solution strategy (see Section 5.1).

Since we use a Gaussian mixture model to generate data samples for this task and thereby know the exact distribution of the input data, we can alternatively train on the componentwise means and covariances. In this setting, the componentwise covariances are identical while the means of all components differ among one another. Thus, the networks employ a form of *mean coding* to find a viable solution strategy (see Section 5.2).

After discussing these two methods, we compare the solution strategies chosen by the networks when trained using either one of the two above methods or when trained on data samples (see Section 5.3). Finally, we briefly summarize the insights gained with regard to the network mapping for our adaptation of the XOR problem (see Section 5.4).

5.1 Covariance coding based on classwise description

We use the network loss $\hat{\mathcal{L}}_{\text{stat.}}$ derived in Section 4.1 (see Eq. (57)) to directly train networks on the classwise means and covariances of the input data. The latter quantities are given by

$$\mu_x^t = \sum_{m'} \tilde{\pi}_{m'}\, \mu_x^{m'}, \tag{69}$$

$$\Sigma_x^t = \left(\sum_{m'} \tilde{\pi}_{m'} \left[\Sigma_x^{m'} + \mu_x^{m'} \left(\mu_x^{m'}\right)^\mathsf{T} \right] - \sum_{m_1', m_2'} \tilde{\pi}_{m_1'}\, \tilde{\pi}_{m_2'}\, \mu_x^{m_1'} \left(\mu_x^{m_2'}\right)^\mathsf{T} \right), \tag{70}$$

where the sum over m' comprises only the components of the respective class label t. Here, we use the mixture weights $\tilde{\pi}_{m'}$ of the classwise distributions, which are given by $\tilde{\pi}_{m'} = \frac{\pi_{m'}}{\pi_t}$ with $\pi_t = 1/2$ for $t = 0, 1$. Using the definition of the input distribution in

(35)-(37), we get

$$t = 0: \qquad \mu_x^{t=0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \Sigma_x^{t=0} = \begin{pmatrix} 0.21 & -0.16 \\ -0.16 & 0.21 \end{pmatrix}, \tag{71}$$

$$t = 1: \qquad \mu_x^{t=1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \Sigma_x^{t=1} = \begin{pmatrix} 0.21 & 0.16 \\ 0.16 & 0.21 \end{pmatrix}. \tag{72}$$

We see that class membership can be uniquely assigned based on these quantities and is effectively encoded in the off-diagonal terms of the covariances. Hence, we refer to this setting as *covariance coding*.

We aim for an equal comparison between the training method based on classwise means and covariances described in Section 4.1 and the training method based on data samples used in Section 3.2. Therefore, we initialize the networks that are studied in this section on the same seeds as in Section 3.2. Furthermore, we use the same training specifics apart from a single adaptation: When training on data samples, the total number of optimization or training steps is indirectly determined by the size D_{train} of the training dataset, the batch size B and the number of training epochs n_{epochs} according to

$$n_{\text{training steps}} = \frac{D_{\text{train}}}{B}\, n_{\text{epochs}}. \tag{73}$$

When training on classwise means and covariances, we directly choose the parameter $n_{\text{training steps}}$. Accordingly, we set the number of training steps to the value used in Section 3.2, $n_{\text{training steps}} = 2000$. For evaluation of the classification performance P_θ, we use the same dataset of size $D_{\text{eval}} = 10^4$ as before. Consequently, the difference between networks studied in this section and networks in Section 3.2 results directly from the difference in training dynamics due to the adaptation of the network loss. If not specified otherwise, we keep these training specifics fixed when training on data statistics in subsequent sections.

For a network of depth $L = 1$ and width $N = 10$, we achieve a high performance of $P_\theta = 92.6\%$ compared to $P_{\max} = 92.9\%$. This result shows that it is indeed possible to obtain a meaningful network mapping when training exclusively on classwise means and covariances. We emphasize that the network did not process any data samples prior to performance evaluation. Instead, the resulting network mapping is entirely based on our Gaussian description of the network as a non-linear mapping of mean and covariance. Thus, this description is sufficiently accurate to serve as an adequate representation of the network mapping during training.

We describe the distribution $p_x(x)$ of the network input as a superposition of Gaussian distributions $\mathcal{N}_{t=0,1}(x)$ per class during training. Accordingly, we can determine theoretical results for the probability density function of the network output z^L by tracing the transformation of the classwise means and covariances. This approach yields

$$p_{z^L}(z^L) = \sum_{t=0,1} \pi_t\, p_{t,\,z^L}(z^L) \tag{74}$$

$$\approx \frac{1}{2} \sum_{t=0,1} \mathcal{N}(z^L | \mu^t_{z^L,\,\text{theo.}},\, \Sigma^t_{z^L,\,\text{theo.}}), \tag{75}$$

where we calculate theoretical values for the classwise means and covariances of the network output according to $\mu^t_{z^L,\,\text{theo.}} = \hat{f}_\mu\big(\mu_x^t,\, \Sigma_x^t;\, \theta\big)$ and $\Sigma^t_{z^L,\,\text{theo.}} = \hat{f}_\Sigma\big(\mu_x^t,\, \Sigma_x^t;\, \theta\big)$

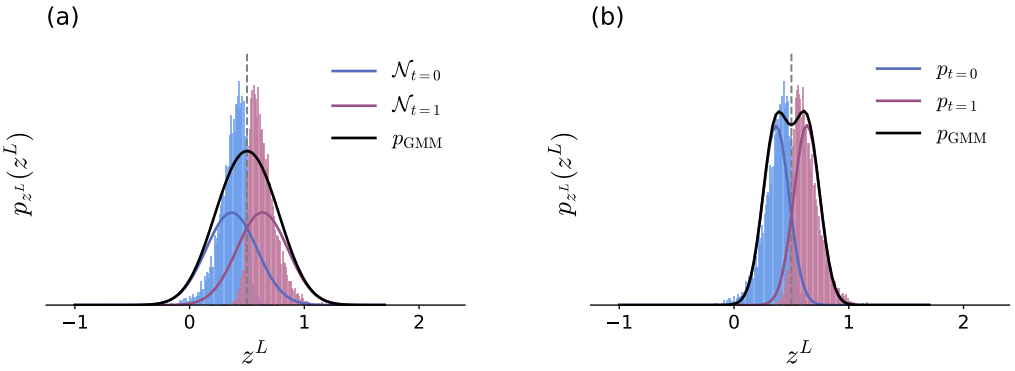


Figure 9: Distribution of the network output $z^L \in \mathbb{R}$ for a network of depth $L = 1$ and width $N = 10$. The network was trained to solve the XOR problem using the network loss determined from the classwise means and covariances, as derived in Section 4.1. Normalized histograms of the network output for a test dataset of size $D_{\mathrm{eval}} = 10^4$ act as empirical estimates of the probability density function. In (a), the theoretical results for the classwise distributions $p_{t=0,1}$ are given by Gaussian distributions $\mathcal{N}_{t=0,1}$ that are calculated using the classwise means and covariances according to Section 2.3. In (b), the theoretical probability density functions $\mathcal{N}_{m,\,z^L}$ for each component m are calculated separately and the classwise distributions $p_{t=0,1}$ are given by their respective superpositions. We further show the resulting distribution p_{GMM} in both cases.

(see Section 2.3). Furthermore, we again approximate the classwise distributions $p_{t,\,z^L}$ as Gaussian distributions $\mathcal{N}(z^L|\mu^t_{z^L,\,\mathrm{theo.}},\, \Sigma^t_{z^L,\,\mathrm{theo.}})$ and use $\pi_t = 1/2$ for $t = 0, 1$. Empirical estimates of the output distribution are again obtained as a histogram of the network output for a test dataset of size $D_{\mathrm{eval}} = 10^4$.

The corresponding probability density functions resulting from theory and simulation are illustrated in Fig. 9(a). The theoretical curves show a poorer agreement with the empirical curves compared to the results in Section 3.2. The increase in deviations is reflected in an increase of the relative Kullback-Leibler divergence, which is $\hat{D}_{KL}(p_{z^L,\,\mathrm{emp.}} \| p_{\mathrm{GMM},\,z^L,\,\mathrm{theo.}}) = 3.4\%$.

However, in Section 3.2 we used the exact form of the input distribution given as a Gaussian mixture model with $M = 4$ components to determine theoretical results. This method does not entail any approximation with regard to the input distribution in contrast to the above method and is therefore expected to yield a more accurate description of the distribution of the network output. To allow for an equal comparison of the results presented here and the results in Section 3.2, we also use the component-wise definition of the input distribution here to obtain the theoretical curves depicted in Fig. 9(b). Note that the network is not trained again and its parameters θ are the same as for Fig. 9(a). In line with expectations, theory and simulation agree significantly better compared to the classwise approach in Fig. 9(a). Correspondingly, the relative Kullback-Leibler divergence yields $\hat{D}_{KL}(p_{z^L,\,\mathrm{emp.}} \| p_{\mathrm{GMM},\,z^L,\,\mathrm{theo.}}) = 1.3\%$.

For a network of depth $L = 4$ and width $N = 10$, we achieve a performance of $P_\theta = 92.1\%$. However, we are going to see that this is not necessarily representative of this network architecture, where we observe a larger variability in the obtained performance values with regard to the initial values of the network parameters. The corresponding average performance $\overline{P_\theta} = 84.8\%$ is significantly lower while the observed variability is reflected in a relatively large standard deviation of $\sigma_{P_\theta} = 6.7\%$. The

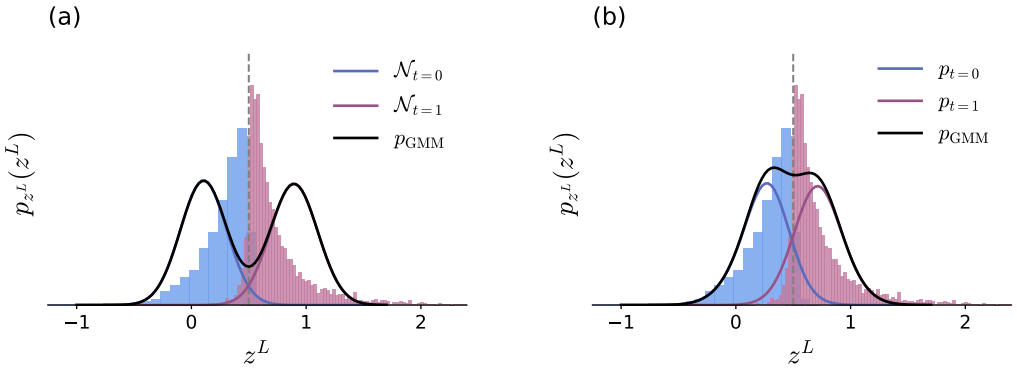


Figure 10: Distribution of the network output $z^L \in \mathbb{R}$ for a network of depth $L = 4$ and width $N = 10$. The network was trained to solve the XOR problem using the network loss determined from the classwise means and covariances, as derived in Section 4.1. Normalized histograms of the network output for a test dataset of size $D_{\mathrm{eval}} = 10^4$ act as empirical estimates of the probability density function. In (a), the theoretical results for the classwise distributions $p_{t=0,1}$ are given by Gaussian distributions $\mathcal{N}_{t=0,1}$ that are calculated using the classwise means and covariances according to Section 2.3. In (b), the theoretical probability density functions $\mathcal{N}_{m,\,z^L}$ for each component m are calculated separately and the classwise distributions $p_{t=0,1}$ are given by their respective superpositions. We further show the overall distribution p_{GMM} in both cases.

theoretical and empirical curves that result when using the classwise and the componentwise description of the input distribution are depicted in Fig. 10(a) and Fig. 10(b), respectively. As in the above case for $L = 1$, the curves agree significantly better when using the componentwise definition of the input distribution to derive the theoretical curves. Correspondingly, we get $\hat{D}_{KL}(p_{z^L,\,\mathrm{emp.}} \| p_{\mathrm{GMM},\,z^L,\,\mathrm{theo.}}) = 27.6\%$ and $\hat{D}_{KL}(p_{z^L,\,\mathrm{emp.}} \| p_{\mathrm{GMM},\,z^L,\,\mathrm{theo.}}) = 10.7\%$, respectively. These values are significantly increased compared to both untrained networks and networks trained on data samples. In Fig. 10, we see that the respective empirical probability density function of the network output is noticeably skewed for each class. This behavior cannot be captured by approximating the output distribution of each component as Gaussian since it results from non-negligible higher order cumulants beyond mean and covariance. As discussed before, the dependencies among network parameters introduced during network training presumably amplify higher order cumulants.

To check whether the effects discussed here appear consistently for different training runs, we determine both achieved performance and relative Kullback-Leibler divergence averaged over networks that are initialized on $n_{\mathrm{seeds}} = 10^2$ different seeds and subsequently trained on the classwise means and covariances. We compare these results for different network architectures with respect to training on data samples and training on classwise means and covariances. To allow for an equal comparison, we determine in both cases theoretical curves for the probability density function by using the definition of the input distribution as a Gaussian mixture model with $M = 4$ components. These results are then used to calculate the relative Kullback-Leibler divergence. The results for both the achieved performance and the relative Kullback-Leibler divergence are shown in Fig. 11, together with the corresponding standard error of the mean.

For networks of depth $L = 1$, training on classwise means and covariances yields performance values very close to its theoretical upper limit of $P_{\mathrm{max}} = 92.9\%$ while training

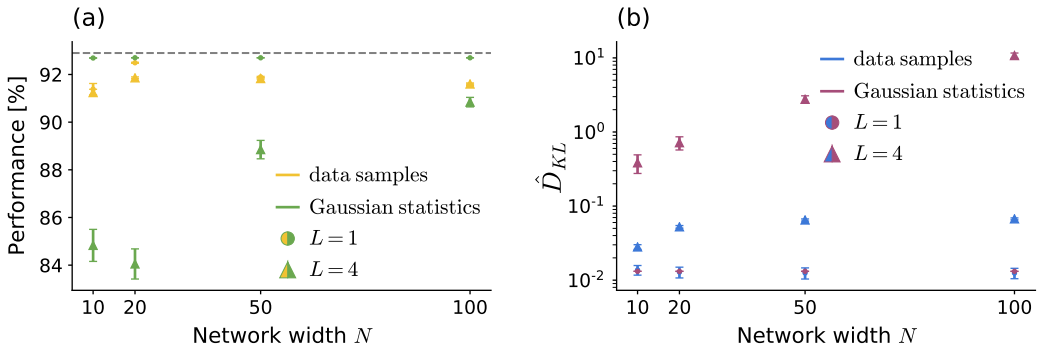


Figure 11: Achieved classification performance and relative Kullback-Leibler divergence for different network architectures, averaged over networks that are initialized on $n_{\text{seeds}} = 10^2$ different seeds. The networks were trained to solve the XOR problem based on either data samples or the classwise means and covariances. In (a), mean and corresponding standard error of the achieved performance are shown for varying network width N, with the theoretical performance limit of $P_{\text{max}} = 92.9\%$ given as reference (gray dashed line). In (b), mean and corresponding standard error of the relative Kullback-Leibler divergence are shown for varying network width N.

on data samples yields slightly lower results. Considering deeper network architectures of depth $L = 4$, we see that training on data samples yields results similar to those for shallow networks, while for training on classwise means and covariances the average performance drops significantly. This effect is particularly pronounced for narrow networks and diminishes with increasing network width N. In Section 2.4 and Section 3.2, we saw that our description of the network mapping becomes less accurate for deeper networks. Since we apply this description during network training whereas we evaluate performance based on the actual network mapping, this decrease in accuracy for deeper network architectures explains the drop in performance for training on classwise means and covariances. Transitioning to broader networks mitigates this effect, which is again in line with expectations according to the theory in Section 2.4.

The behavior of the achieved performance values is reflected in the obtained values for the relative Kullback-Leibler divergence. For networks of depth $L = 1$, training on data samples and training on classwise means and covariances yields similar results. In contrast, for network depth $L = 4$ the relative Kullback-Leibler divergence is significantly increased for networks trained on classwise means and covariances compared to networks trained on data samples, quantifying the less accurate description of the network output.

Nevertheless, the results presented here show that training on classwise means and covariances yields competitive performance values in comparison to training on data samples. Thus, our adaptation of the XOR problem is an example where the assumption holds true that the essential information is contained in the classwise means and covariances. Furthermore, it follows that the covariance coding used in training on classwise means and covariances can serve as a viable information coding paradigm.

To obtain a high classification performance according to the definition given in Section 3.2, the classwise means of the network output need to be sufficiently separated. Since the information regarding class membership is exclusively encoded in the classwise covariances of the input data, an information exchange between mean and covariance is thus essential for this coding paradigm. This information exchange occurs when apply-

ing the quadratic activation function which yields a contribution of the preactivations' covariance Σ_z to the mean μ_y of the postactivations (see Eq. (23)(c) in Section 2.2). However, this contribution involves only diagonal elements of the covariance while the necessary information is encoded in the off-diagonal terms. Therefore, performing an affine linear transformation prior to the application of the quadratic activation function is essential for this coding paradigm as well.

Lastly, we want to point out that we used our knowledge regarding the exact distribution of the network input to obtain values for the classwise means and covariances. Alternatively, we can determine empirical estimates for these quantities based on the dataset used when training on data samples. Using these empirical estimates for training on classwise means and covariances yields results that are very similar to those obtained using the exact values. Thus, information encoding using the classwise means and covariances is robust to noise that is introduced by using empirical estimates of these quantities. By using empirical estimates for the classwise means and covariances, this training method is thus applicable to any dataset where the underlying distribution of the input data might be unknown.

5.2 Mean coding based on componentwise description

In the previous section, we found that training on the classwise means and covariances on average yields suboptimal performance values for deeper network architectures. This effect most likely results from our description of the network output becoming less accurate. At the same time, we saw that using the componentwise instead of the classwise description of the input distribution yields a better agreement between theoretical and empirical results for the probability density function of the network output. Since we know the exact form of the input distribution, we can directly train on the componentwise means and covariances as given in (35) and (36). The class membership is then effectively encoded in the componentwise means while the covariances are identical for all components. We thus refer to this setting as *mean coding*.

We use the same training specifics as in the previous section. For a network of depth $L = 1$ and width $N = 10$, we achieve a performance of $P_\theta = 92.5\%$. Since we train on the componentwise means and covariances, we use these quantities to determine the theoretical probability density function of the network output. The resulting curves agree well with the empirical results as shown in Fig. 12(a). For the relative Kullback-Leibler divergence, we get $\hat{D}_{KL}(p_{z^L,\,\mathrm{emp.}} \| p_{\mathrm{GMM},\,z^L,\,\mathrm{theo.}}) = 1.3\%$ which is similar to the results obtained for networks that were trained using one of the other two methods.

For a deeper network architecture of depth $L = 4$ and width $N = 10$, we obtain a performance value of $P_\theta = 92.0\%$. The corresponding empirical distribution of the network output is significantly skewed, as shown in Fig. 12(b). While the theoretical results give reasonable values for mean and covariance, describing the output distribution as a superposition of two Gaussian distributions per class cannot account for the skewness of the output distribution. In consequence, the relative Kullback-Leibler divergence is $\hat{D}_{KL}(p_{z^L,\,\mathrm{emp.}} \| p_{\mathrm{GMM},\,z^L,\,\mathrm{theo.}}) = 24,079\%$.

To check whether the above discussed effects appear consistently for different training runs, we determine both achieved performance and relative Kullback-Leibler divergence averaged over networks that are initialized on $n_{\mathrm{seeds}} = 10^2$ different seeds and subsequently trained on the componentwise means and covariances. We compare these results for different network architectures with respect to training on data samples and training on componentwise means and covariances. The results are shown in Fig. 13,

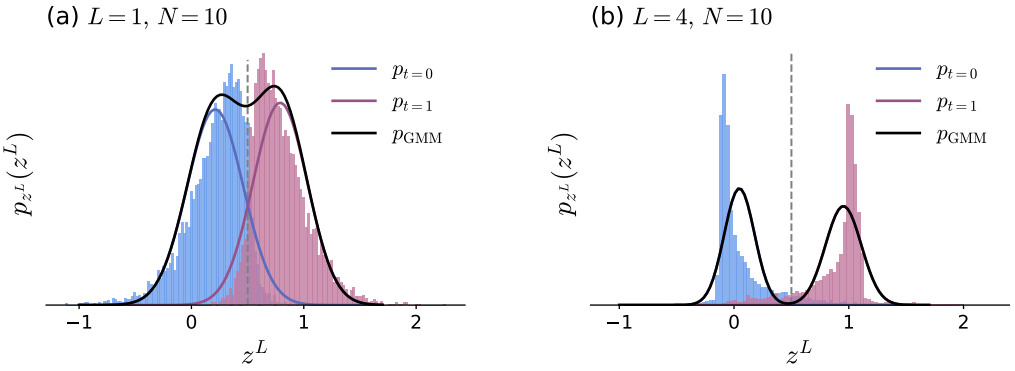


Figure 12: Distribution of the network output $z^L \in \mathbb{R}$ for a network of depth $L = 1$ (a) or $L = 4$ (b) and width $N = 10$. The networks were trained to solve the XOR problem using the network loss determined from the componentwise means and covariances, as derived in Section 4.1. Normalized histograms of the network output for a test dataset of size $D_{\mathrm{eval}} = 10^4$ act as empirical estimates of the probability density function. The theoretical probability density functions $\mathcal{N}_{m,\,z^L}$ for each component m are calculated separately and the classwise distributions $p_{t=0,\,1}$ are given by their respective superpositions. We further show the overall distribution p_{GMM} in both cases.

together with the corresponding standard error of the mean.

Similar to the results in the previous section, we obtain performance values close to the upper limit $P_{\mathrm{max}} = 92.9\%$ in the case of networks of depth $L = 1$ that are trained using the componentwise means and covariances. In contrast to before, performance does not drop with increasing network depth when training on componentwise means and covariances. Instead, this method yields similar results as training on data samples. We obtain higher performance values when training on componentwise means and covariances compared to training on classwise means and covariances since the componentwise means and covariances give an exact description of the distribution of the input data and in consequence generally yield a more accurate description of the network output.

At the same time, the relative Kullback-Leibler is significantly increased for deeper network architectures compared to both training on data samples and training on classwise means and covariances. This is due to the empirical output distributions of each class being significantly skewed, which cannot be captured by approximation as a superposition of two Gaussian distributions. This behavior results from higher order cumulants beyond mean and covariance becoming non-negligible. As discussed in previous sections, higher order cumulants might be amplified by dependencies among network parameters introduced during network training.

Altogether, the results presented here show that the information encoded in the componentwise means is sufficient to train networks. Consequently, training on componentwise means and covariances achieves competitive performance in comparison to both training on data samples and training on classwise means and covariances. Thus, using mean coding in training on componentwise means and covariances can serve as an alternative information coding paradigm besides covariance coding. In contrast to the covariance coding discussed in the previous section, using the information encoded in the componentwise means does not necessarily rely on information exchange between mean and covariance within the network. Nevertheless, this information exchange might

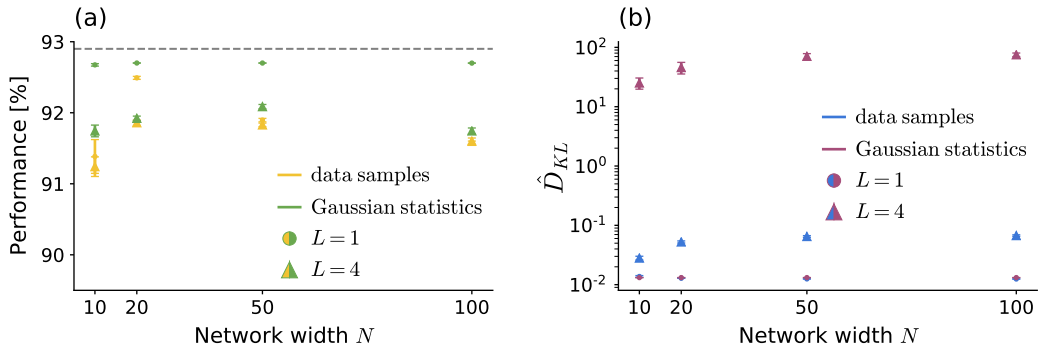


Figure 13: Achieved classification performance and relative Kullback-Leibler divergence for different network architectures, averaged over networks that are initialized on $n_{\text{seeds}} = 10^2$ different seeds. The networks were trained to solve the XOR problem based on either data samples or the componentwise means and covariances. In (a), mean and corresponding standard error of the achieved performance are shown for varying network width N, with the theoretical performance limit of $P_{\text{max}} = 92.9\%$ given as reference (gray dashed line). In (b), mean and corresponding standard error of the relative Kullback-Leibler divergence are shown for varying network width N.

be beneficial with regard to the separability of data in the output space and thus finding a viable solution strategy.

Unlike when training on classwise means and covariances, we cannot directly obtain empirical estimates of the componentwise means and covariances to use such estimates during training. However, for a dataset where the underlying distribution of the input data is unknown, an analogous adaptation is to fit a Gaussian mixture model to the classwise distributions in order to improve the description of the input data and thus also of the output data.

5.3 Comparison of coding paradigms with network training on data samples

We have seen that for training on Gaussian data statistics there exist two possible information coding paradigms which can be used for viable solution strategies. As a next step, we study to what extent these solution strategies differ from one another as well as how they compare to the solution strategy chosen by the network when training on data samples. To keep the arguments concise, in the following we refer to training on data samples as *sample coding*.

For a network of depth $L = 1$ and width $N = 10$, we train a particular network that is initialized on a single seed using all of the above-mentioned training methods and illustrate the resulting network mappings in Fig. 14. The implemented decision boundaries correspond well to the optimal decision boundaries, which is in line with expectations since all trained networks yield performance values close to the theoretical upper limit.

The main difference appears in the area around zero which corresponds to a small proportion of the probability mass of each class. Therefore, in relation to the size of the full test dataset, this difference only applies to a small number of data samples. Consequently, the classification performance is not strongly affected by the behavior

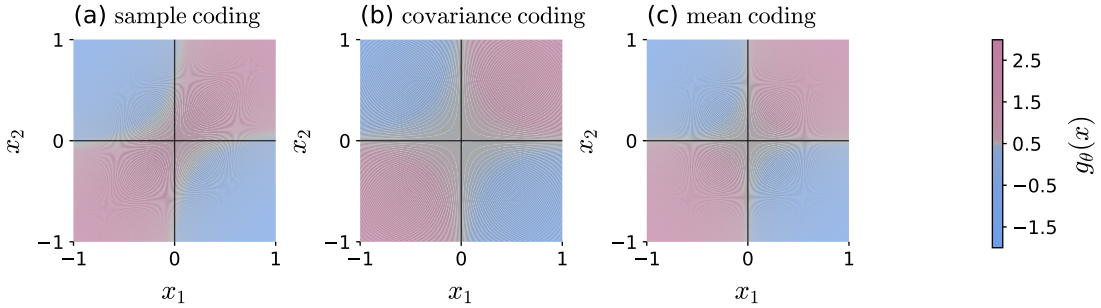


Figure 14: Illustration of resulting network mapping for different coding paradigms. All points within an area of one color (blue, red) are assigned to the respective class label ($t = 0, 1$), while the gray areas indicate the respective decision boundaries. We give the optimal decision boundaries as reference (black lines).

of the network mapping concerning that area. In the case of sample coding, the low probability mass around zero also implies that it can be more difficult to infer the position of the optimal decision boundaries within that area, as can be seen in Fig. 14(a). However, overall the resulting network mappings behave similarly for the different coding paradigms. Therefore, the network mappings are uninformative with respect to differences between the solution strategies chosen by the networks.

Since the number of parameters within a network is rather large, two different choices of network parameters can yield a similar network mapping $g_{\theta_1}(x) \approx g_{\theta_2}(x)$ for $x \in \mathbb{R}^{d_{\text{in}}}$. However, the internal information processing might still be based on different solution strategies. Therefore, we study whether a solution strategy chosen by a network that was trained using one method is a viable solution strategy when subsequently training with a different method. In particular, we consider all six combinations of pairs of coding paradigms (sample coding, covariance coding, and mean coding). In all settings, we choose the same training specifics as in the previous sections, in particular we have a total of $n_{\text{training steps}} = 2000$ optimization steps for each training run. Accordingly, the used coding paradigm switches at training step $T_{\text{switch}} = 2.000$, where we in general denote the training step as T.

We study the network loss over training time and the absolute change of network parameters within the past $\Delta_T = 10$ training steps which is determined according to $\Delta\|\theta\|_2\big|_T = \|\theta_{T+\Delta_T} \text{-} \theta_T\|_2$. When comparing mean and covariance coding as shown in Fig. 15(a) and Fig. 15(b), we see that the value of the network loss changes significantly at T_{switch} and the network parameters are adapted accordingly. In particular, the resulting change of the network parameters $\Delta\|\theta\|_2$ at T_{switch} is of the same order of magnitude as at the beginning of network training which indicates that the network is trained anew. Thus, mean coding and covariance coding find two different solution strategies to solve the given task.

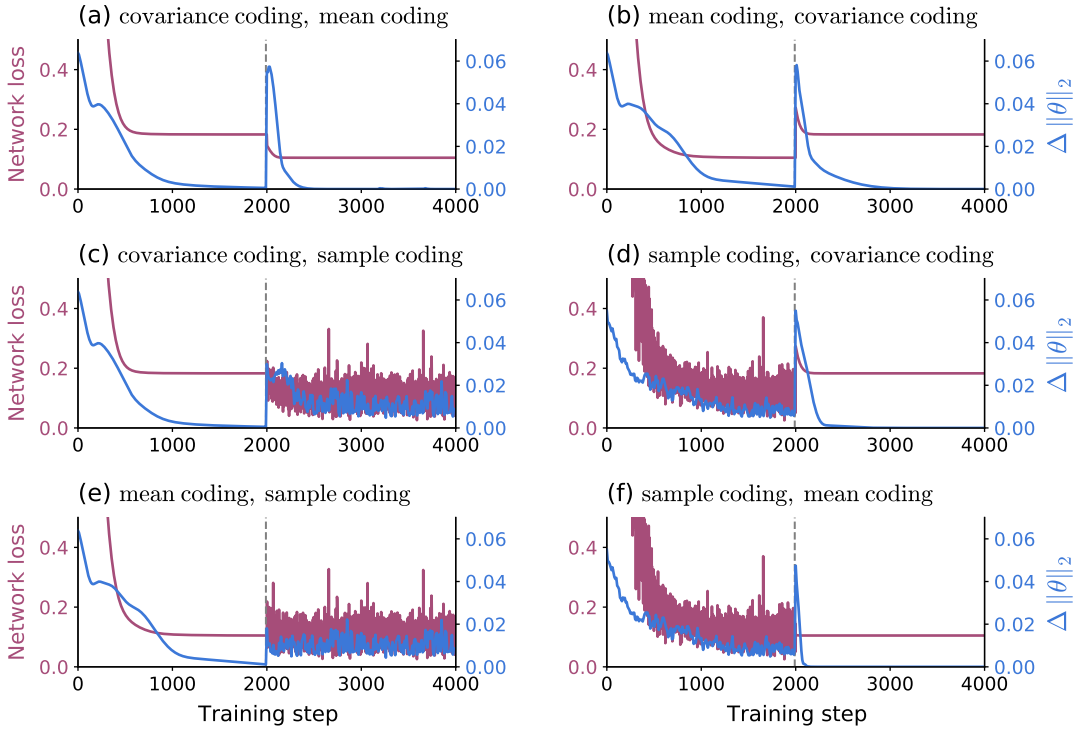


Figure 15: Network loss (red) and absolute change of network parameters (blue) for a network of depth $L = 1$ and width $N = 10$. The network was trained using one coding paradigm until $T_{\text{switch}} = 2.000$ (dashed gray line) and subsequently trained using a different coding paradigm as indicated.

As a next step, we compare covariance coding and sample coding with one another, as shown in Fig. 15(c) and Fig. 15(d). Due to the stochasticity introduced by determining the network loss as the sample mean of a batch $\mathcal{B}$ of size $B = 10$ of the full training dataset, both network loss and network parameters continue to fluctuate slightly even after training has converged. When first training with covariance coding and subsequently with sample coding, the network loss on average decreases slightly at T_{switch} and the network parameters are slightly adapted accordingly, as shown in Fig. 15(c). When switching the order of these two coding paradigms, the network loss changes significantly at T_{switch} and we see a corresponding peak in the change of network parameters $\Delta\|\theta\|_2$ at T_{switch} (see Fig. 15(d)). Contrary to setting in Fig. 15(c), the magnitude of the change of network parameters $\Delta\|\theta\|_2$ at T_{switch} is similar to the magnitude at the beginning of network training, which indicates that the network is trained anew.

Since evaluation of classification performance is sample-based, we expect any network that yields a high performance to implement a solution strategy that is at least close to a solution strategy viable for sample coding. In line with expectations, Fig. 15(d) shows that sample coding can implement a solution strategy that is similar to the one that is used for covariance coding. However, covariance coding yields a different solution strategy than the one found by sample coding when initialized with randomly sampled network parameters. Taken together, these results indicate that training on data samples finds yet another solution strategy that differs from both mean and covariance coding.

Lastly, we compare mean coding and sample coding which is shown in Fig. 15(e)

and Fig. 15(f). When first training with mean coding and subsequently with sample coding, the network parameters show only slight changes, indicating that mean coding is in fact a viable solution also for sample coding. The switch between methods introduces fluctuations in both network loss and network parameters after T_{switch}. These fluctuations result from the stochasticity due to the use of batches in network training (see Fig. 15(e)). When switching the order of these two coding paradigms, the network loss decreases slightly at T_{switch} while the network parameters are adapted noticeably, as shown in Fig. 15(f). The order of magnitude of the change of the network parameters $\Delta\|\theta\|_2$ at T_{switch} is similar to the magnitude at the beginning of network training. Thus, mean coding yields a different solution strategy than the one found by sample coding when initialized with randomly sampled network parameters.

To understand these results, we consider the definition of the input distribution as a Gaussian mixture model. In the case that the Gaussian distribution of each component is rather narrow, we can approximate

$$x|_{x\sim\mathcal{N}(x|\mu_x^m,\,\Sigma_x^m)} \approx \mu_x^m \tag{76}$$

and rewrite the network loss used during training on data samples as

$$\mathcal{L}\big(\{(x^{(b)},t^{(b)})\}_{b=1,\dots,B};\,\theta\big) = \frac{1}{B}\sum_{b=1}^{B}\|g_\theta\big(x^{(b)}\big)-t^{(b)}\|^2 \tag{77}$$

$$\approx \sum_m \pi_m \|g_\theta\big(\mu_x^m\big)-t^m\|^2. \tag{78}$$

This corresponds to a componentwise mean-field approximation of the network as

$$g_{\theta,\,\text{MF}} = \sum_m \pi_m\, g_\theta\big(\mu_x^m\big). \tag{79}$$

Here, we do not get a contribution from the diagonal terms of the covariance in comparison to our description as a non-linear mapping of mean and covariance (see Eq. (23)(c) in Section 2.2 and Section 2.3). In the case that the Gaussian distribution of each component is sufficiently narrow, this contribution becomes negligible and thus mean coding matches sample coding, which explains the result in Fig. 15(e).

In the case of covariance coding, we would need to replace the componentwise by the classwise means in the above derivations. However, this approach does not yield a sensible description of the network mapping with regard to the given task since the classwise means neither correspond to the actual values of the data samples nor contain any information regarding class membership.

Compared to the other two training methods, sample coding can infer information contained in higher order cumulants and might therefore yield a slightly different solution strategy than mean and covariance coding, which explains the results in Fig. 15(d) and Fig. 15(f). Altogether, these considerations correspond well to the results presented in Fig. 15.

5.4 Interim conclusions

Based on the results that we have discussed so far, we can conclude that our description of the network mapping as a non-linear mapping of mean and covariance of the input distribution already gives a reasonable approximation of the network output. In

particular, it is sufficiently accurate to train networks on our adaptation of the XOR problem so that they yield competitive performance values compared to training on data samples. Consequently, our assumption that the information relevant to solve the given task is contained in the classwise means and covariances has proven valid for our adaptation of the XOR problem.

However, we also saw that for deeper network architectures the network output is significantly skewed, which cannot be captured by approximating the output distribution as Gaussian. This behavior probably results from the higher order cumulants beyond mean and covariance becoming non-negligible. Dependencies among network parameters that are introduced during network training presumably amplify higher order cumulants.

Furthermore, there exist different information coding paradigms which can be used by the network to implement different solution strategies. More precisely, training on data samples can utilize both mean coding and covariance coding as well as further solution strategies that possibly leverage information encoded in higher order cumulants. Altogether, decomposing the network in terms of cumulants allows us to gain insights into the information used for a particular solution strategy as well as the functional principles this strategy is based on.

6 Classification on MNIST

So far, we have seen that the description of the network as a non-linear mapping of mean and covariance is sufficiently accurate to train networks on the XOR problem described in Section 3.1. As a next step, we apply this training method to classification of the MNIST database as an example of non-synthetic data. In this case, we exclusively utilize empirical estimates of the classwise means and covariances of the input data during training. This allows us to check to which extent a viable solution strategy can be based on these features.

We first give an overview of the MNIST database and provide an intuition for the information contained in the corresponding Gaussian data statistics (see Section 6.1). We then train networks both on data samples and on the classwise means and covariances to compare the achieved performance values (see Section 6.2).

6.1 MNIST database

The MNIST database (Modified National Institute of Standards and Technology database) provides gray-scale images of handwritten digits from zero to nine and is a widely used example in machine learning. It was first introduced in 1998 by LeCun et al. [25] and forms a subset of a larger dataset from the National Institute of Standards and Technology (NIST). The images have been modified in such a way that the digits are normalized with respect to their size and centered in images of 28×28 pixels. The MNIST database consists of a training dataset of $60,000$ images and a test dataset of $10,000$ images [26]. We show examples of images from the training dataset in Fig. 16.

The classification task consists in assigning each image to the corresponding digit. For data processing, the gray-scale values of each image are vectorized, yielding input data of size $d_{\text{in}} = 28 \times 28 = 784$. Apart from this, no further pre-processing is applied.

We use network architectures with $d_{\text{out}} = 10$ output units and choose the class labels as $t^k = \hat{e}_k$ for $k = 1, \ldots, 10$ with $\hat{e}_k$ being the k^{th} unit vector ($(\hat{e}_k)_i = \delta_{ik}$). Here, the class k contains all images showing digit $k-1$. This choice of class labels is called *one-hot-encoding* and avoids introducing any hierarchy among class labels which can harm performance [27]. We adapt the assignment of predicted class labels $\hat{t}$ for the network

Figure 16: Example images from the MNIST database for the digits from zero to nine.

mapping $g_\theta(x) \in \mathbb{R}^{10}$ as follows

$$\hat{t} = t^{\hat{k}} \quad \text{for } \hat{k} = \left[\underset{\mathrm{i}}{\operatorname{argmax}} \left(g_\theta(x)\right)_i\right]. \tag{80}$$

For training on the classwise means and covariances, we calculate these quantities from the full training dataset. From the classwise means shown in Fig. 17, we can see that these act as representatives of their respective class.

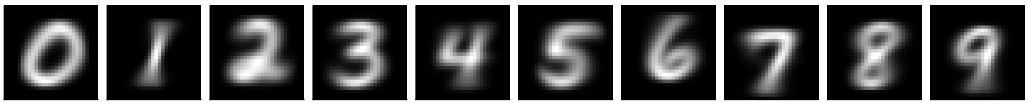


Figure 17: Illustration of classwise means calculated from the full training dataset for the digits from zero to nine.

To obtain an understanding of the information available to the network when training on the classwise means and covariances, we can draw samples from the corresponding multivariate Gaussian distributions. For this purpose, we use $\tilde{\Sigma}_x^t = \Sigma_{x,\,\mathrm{emp.}}^t + 10^{-5}\mathbb{I}$ for the covariance to ensure that it is non-singular. For the digit three, data samples drawn from the corresponding Gaussian distribution as well as images from the training dataset are depicted in Fig. 18. The comparison shows that approximating the classwise distribution as Gaussian can already account for a large amount of the variability within each class. However, we also see that it cannot cover the original distribution to its full extent.

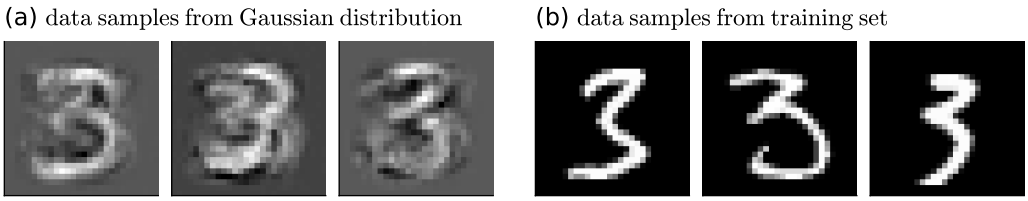


Figure 18: Illustration of input data for the digit three. **(a)** Data samples drawn from the corresponding Gaussian distribution. **(b)** Data samples from the training dataset.

6.2 Results for different training methods

We compare training on data samples and training on classwise means and covariances as described in Section 4.1. For training on data samples, we use the full training dataset and choose a batch size of $B = 100$ and $n_{\mathrm{epochs}} = 2$ training epochs. These training specifics correspond to $n_{\mathrm{training\ steps}} = 1200$ training steps (see Eq. (73) in Section 5.1), which we thus use for training on classwise means and covariances.

We calculate the achieved classification performance averaged over networks that are initialized on $n_{\mathrm{seeds}} = 10$ different seeds and subsequently trained using one of the above methods. To allow for an equal comparison, we use the same network initializations for both training methods. Compared to previous sections, we here average over a relatively small number of networks because each training run is significantly more expensive

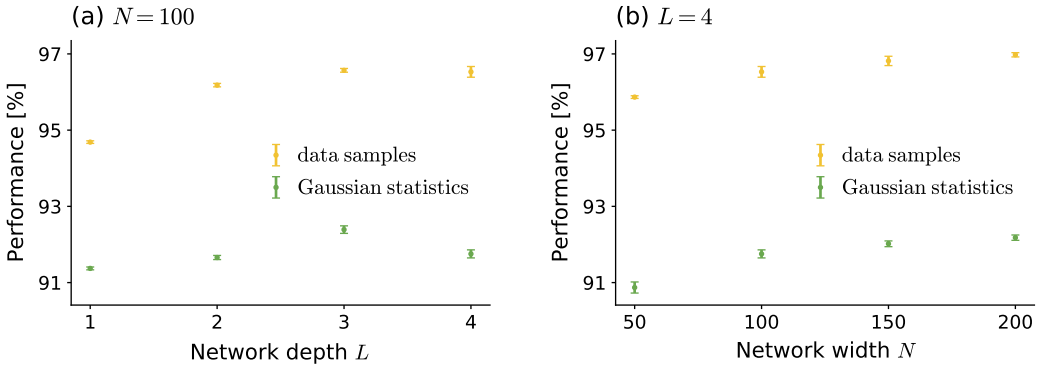


Figure 19: Achieved classification performance on MNIST for different network architectures, averaged over networks that are initialized on $n_{\text{seeds}} = 10$ different seeds. The networks were trained on either data samples (yellow) or the classwise means and covariances (green). We show both mean and corresponding standard error for varying network depth L and width N, respectively.

in terms of computation time. Therefore, we examine the standard deviations of the obtained performance values and its corresponding error estimates in more detail below.

The obtained performance values for network architectures of different depths and widths are shown in Fig. 19(a) and Fig. 19(b), respectively. For both training methods, performance increases with network depth L and network width N. This is in line with expectations since larger networks are typically more powerful. At the same time, we observe a performance gap of ca. $4 - 5\%$ between the two training methods which appears consistently for all used network architectures. This suggests that this gap does not arise due to a limitation of the expressive power of a particular network architecture. Instead, training on data samples can presumably leverage information contained in higher order cumulants beyond mean and covariance for each class, allowing for a better separation of data samples in the output space.

Due to the relatively small number of seeds, we examine in detail the corresponding standard deviations, which act as estimates for the variability in performance across networks, and their error estimates to assess whether the achieved performance values and in particular the observed performance gap are consistent across different network initializations. The values for the standard deviation are all smaller than 0.5%, as shown in Fig. 20. For two network architectures, the error estimates when training on data samples are relatively large so that the corresponding errorbars pass to the negative axis, which is a pure artifact from the statistical analysis. Nevertheless, the error estimates are in a reasonable range and thus indicate that the obtained performance values are consistent across different network initializations. In particular, the performance gap does not result from variability of classification performance for different network initializations since its magnitude is around $4 - 5\%$.

Furthermore, we see in Fig. 19 that the achieved performance reaches a plateau for both training methods, yielding average performance values of up to $\overline{P_\theta} = 97.0\%$ for $L = 4$ and $N = 250$ when training on data samples. In comparison, a performance of $P_{\text{lit.}} = 96.95\%$ was obtained in the original work by LeCun et al., 1998 [25]. In this work, a multi-layer perceptron with three layers of widths $28 \times 28 - 300 - 100$ was trained using the mean squared error as loss function. More recent works report performance values of up to $P_{\text{lit.}} = 98.6\%$ for a similar network architecture of a multi-

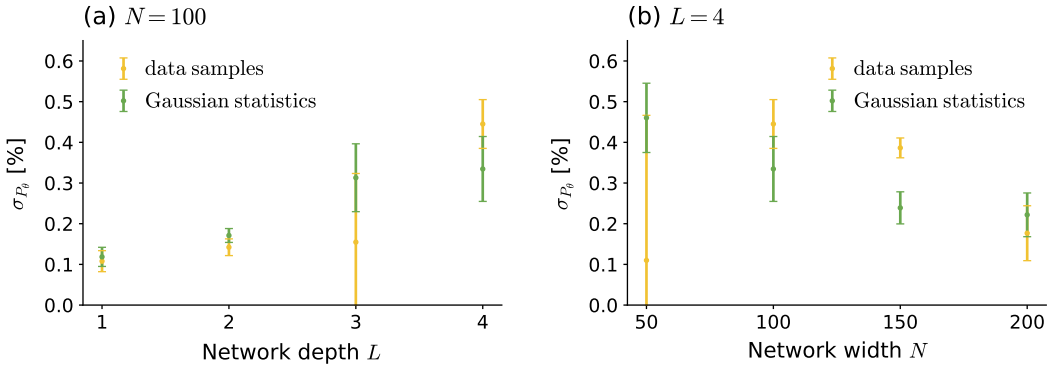


Figure 20: Standard deviations of the achieved classification performance on MNIST and corresponding error estimates for different network architectures, averaged over networks that are initialized on $n_{\text{seeds}} = 10$ different seeds. The networks were trained on either data samples (yellow) or the classwise means and covariances (green).

layer perceptron. In that case, no additional pre-processing of the images was used, but the networks were trained with other training specifics, in particular a different loss function [28]. Thus, our choice of network architecture and training specifics yields performance results comparable to those of previous studies on this topic.

7 Limitations of network training on Gaussian statistics

In the previous chapter, we saw that, while training on classwise means and covariances yields reasonable results, it can be suboptimal compared to training on data samples. Most likely, training on data samples can leverage additional information that is contained in higher order cumulants to find a superior solution strategy.

To test this hypothesis, we create a classification task for which the necessary information is exclusively contained in higher order cumulants beyond mean and covariance (see Section 7.1). Consequently, we expect training on Gaussian data statistics to fail while training on data samples should still achieve high performance which we validate by applying both training methods (see Section 7.2).

7.1 Construction of the problem of alternating hills

We construct a dataset using a Gaussian mixture model (see Section 3.1) for which the classwise means and covariances are identical while higher order cumulants differ between classes. To this end, we choose $M = 4$ components for $d_{\text{in}} = 2$ and set

$$t = 0: \quad \begin{cases} \mu_x^{m=1} = -\begin{pmatrix} 1.5 \\ 0 \end{pmatrix} & \Sigma_x^{m=1} = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.05 \end{pmatrix}, \\ \mu_x^{m=2} = \begin{pmatrix} 0.5 \\ 0 \end{pmatrix} & \Sigma_x^{m=2} = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.05 \end{pmatrix}, \end{cases} \tag{81}$$

$$t = 1: \quad \begin{cases} \mu_x^{m=3} = -\begin{pmatrix} 0.5 \\ 0 \end{pmatrix} & \Sigma_x^{m=3} = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.05 \end{pmatrix}, \\ \mu_x^{m=4} = \begin{pmatrix} 1.5 \\ 0 \end{pmatrix} & \Sigma_x^{m=4} = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.05 \end{pmatrix}. \end{cases} \tag{82}$$

The components are weighted according to

$$\pi_m = \begin{cases} \frac{1}{8} & \text{for } m = 1, 4; \\ \frac{3}{8} & \text{else.} \end{cases} \tag{83}$$

The resulting distribution is shown in Fig. 21(a). The different weighting of components is illustrated in Fig. 21(b), where we show a histogram of the data samples when

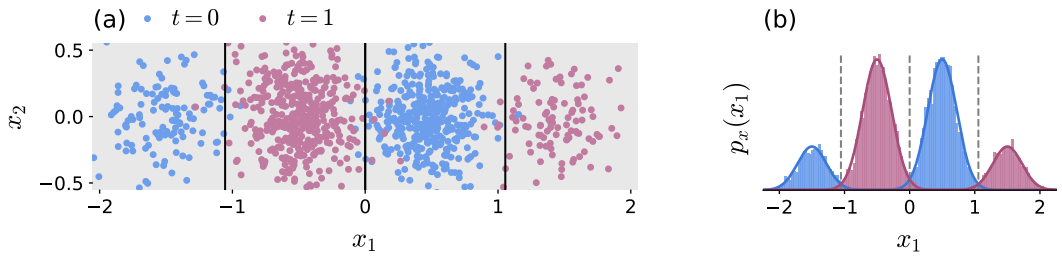


Figure 21: Distribution of data samples $x \in \mathbb{R}^2$ for the problem of alternating hills. Data samples are drawn from the Gaussian mixture model defined in (81)-(83) with class labels $t \in \{0, 1\}$ (blue, red) assigned accordingly. In (a), we show the spatial distribution of the data samples as well as the optimal decision boundaries for solving the task (black lines). For illustrative purpose, we show a subset of 10^3 data samples of the full dataset of size $D = 10^4$ and constrain the depicted area in such a way that the majority of data samples is shown. In (b), we show a histogram of the full dataset projected to the x_1-axis with corresponding one-dimensional classwise distributions (solid blue and red lines). The intersection points of the classwise probability density functions (gray dashed lines) define the optimal decision boundaries.

projected to the x_1-axis. The probability density function takes the form of four adjacent hill-like structures which are assigned to the two classes in an alternating manner. Hence, we call this classification task the *problem of alternating hills*.

The classwise means and covariances are given by

$$\mu_x^t = \sum_{m'} \tilde{\pi}_{m'}\, \mu_x^{m'}, \tag{84}$$

$$\Sigma_x^t = \left(\sum_{m'} \tilde{\pi}_{m'} \left[\Sigma_x^{m'} + \mu_x^{m'} \left(\mu_x^{m'}\right)^\intercal \right] - \sum_{m_1', m_2'} \tilde{\pi}_{m_1'}\, \tilde{\pi}_{m_2'}\, \mu_x^{m_1'} \left(\mu_x^{m_2'}\right)^\intercal \right), \tag{85}$$

where the sum over m' comprises only the components of the respective class label t. Here we use the mixture weights $\tilde{\pi}_{m'}$ of the classwise distributions, which are given by $\tilde{\pi}_{m'} = \frac{\pi_{m'}}{\pi_t}$ with $\pi_t = 1/2$ for $t = 0, 1$. This yields

$$\mu_x^{t=0,1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \tag{86}$$

$$\Sigma_x^{t=0,1} = \begin{pmatrix} 0.8 & 0 \\ 0 & 0.05 \end{pmatrix}. \tag{87}$$

Furthermore, we can determine the third order cumulant according to its definition and obtain

$$G^{(3),t}_{x,(r_1,r_2,r_3)} = \begin{cases} -\delta_{r_1 r_2 r_3}\, \delta_{r_1 1}\, 0.75 & \text{for } t = 0, \\ \delta_{r_1 r_2 r_3}\, \delta_{r_1 1}\, 0.75 & \text{else.} \end{cases} \tag{88}$$

Thus, while mean and covariance for each class are the same, the respective third order cumulants differ from one another by a sign factor. Consequently, the third order cumulant is the cumulant of lowest order that contains information regarding class membership. In particular, this information is also sufficient to correctly assign the respective class.

Theoretical performance limit

As for our adaptation of the XOR problem in Section 3.1, the overlap of the probability density functions corresponding to different components implies a theoretical limit of the achievable performance. We first determine the optimal decision boundaries for the given task which correspond to the intersection points of the classwise probability density functions. Due to the symmetry of the distribution, one of these boundaries coincides with the x_2-axis while the other two are parallel to the x_2-axes and obey

$$\frac{1}{8}\,\frac{1}{\sqrt{2\pi\,0.05}}\,\exp\left(\frac{(x^*_{1,\pm}\mp 1.5)^2}{2\cdot 0.05}\right) = \frac{3}{8}\,\frac{1}{\sqrt{2\pi\,0.05}}\,\exp\left(\frac{(x^*_{1,\pm}\mp 0.5)^2}{2\cdot 0.05}\right),$$

from which follows

$$\begin{aligned} x^*_{1,\pm} &= \pm(1 + 0.05\,\ln 3) \\ &\approx \pm 1.06. \end{aligned}$$

The minimal expected classification error ε_m of each component m is given by the respective probability mass which is not located within the corresponding decision boundaries. Thus, we have

$$\begin{aligned} \varepsilon_{m=1,4} &= 1 - \Phi_{0,1}\left(\frac{x^*_{1,\pm}\pm 1.5}{\sqrt{0.05}}\right) \\ &\approx 2.3\%, \\ \varepsilon_{m=2,3} &= \left[1 - \Phi_{0,1}\left(\frac{x^*_{1,\pm}\mp 0.5}{\sqrt{0.05}}\right)\right] + \left[1 - \Phi_{0,1}\left(\frac{0\mp 0.5}{\sqrt{0.05}}\right)\right] \\ &\approx 1.9\%, \end{aligned}$$

yielding a minimal expected cumulative classification error of

$$\begin{aligned} \mathcal{E}_{\min} &= 2\,\frac{1}{8}\,\varepsilon_{m=1,4} + 2\,\frac{3}{8}\,\varepsilon_{m=2,3} \\ &\approx 2.0\%. \end{aligned}$$

Consequently, the achievable performance for this task is bounded by $P_{\max} = 98.0\%$, which we take as baseline for network evaluation in the following sections.

7.2 Results for different training methods

We train networks using three different approaches: As a first method, we train on the exact classwise means and covariances according to Section 4.1. We expect this method to fail since the classwise means and covariances are identical for the two different classes. In addition, we train on data samples in which case we expect to obtain high performance results as the distribution of the input data can be estimated from data samples. Lastly, we train on the componentwise means and covariances as a cross-check. In contrast to the classwise means and covariances, these give an exact description of the distribution of the input data. In particular, the mean values of components belonging to different classes differ from each other and we therefore expect to obtain high performance results. To allow for an equal comparison, we train networks that are initialized on the same seeds when using the different methods.

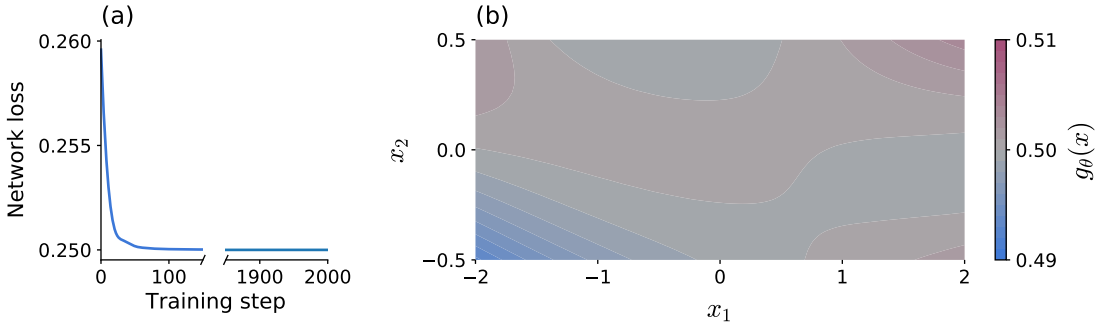


Figure 22: **(a)** Evolution of network loss during training for a network of depth $L = 2$ and width $N = 10$. The network was trained on the classwise means and covariances to solve the problem of alternating hills. **(b)** Illustration of the resulting network mapping.

For a network of depth $L = 2$ and width $N = 10$, we obtain a performance of $P_{\theta,\,\text{classwise}} = 46.2\%$ when using the first training method. This result is in line with our expectations, being even slightly worse than assigning class labels by chance, which results in $P_{\text{chance}} = 50\%$. At the same time, since the network loss ceases changing on a noticeable scale, the network has been trained to its full extent when using this method (see Fig. 22(a)).

To understand the solution strategy chosen by the network, we study the training objective, which in general amounts to minimizing the network loss with respect to the network parameters θ. According to the derivations in Section 4.1, the network loss can be written in terms of the classwise means and covariances of the network output. For training on classwise data statistics, the training objective is thus generally given by

$$\min_\theta \mathcal{L}_{\text{stat.}}\big(\{\mu^k_{z^L},\, \Sigma^k_{z^L}\}_{k=1,2}; \theta\big) = \min_\theta \sum_{k=1,2} p_k \big(\operatorname{tr} \Sigma^k_{z^L} + \|\mu^k_{z^L} - t^k\|^2\big). \tag{89}$$

During training on classwise means and covariances, we calculate theoretical values for mean and covariance of the network output according to $\mu^k_{z^L,\,\text{theo.}} = \hat{f}_\mu\big(\mu^k_x,\, \Sigma^k_x;\, \theta\big)$ and $\Sigma^k_{z^L,\,\text{theo.}} = \hat{f}_\Sigma\big(\mu^k_x,\, \Sigma^k_x;\, \theta\big)$ (see Section 2.3). Since the classwise means and covariances of the input distribution are identical for both classes, the corresponding theoretical values for mean and covariance of the network output are also identical. Consequently, the network loss used for training on classwise means and covariances is given by

$$\min_\theta \hat{\mathcal{L}}_{\text{stat.}}\big(\{\mu^k_x,\, \Sigma^k_x\}_{k=1,2}; \theta\big) = \min_\theta \frac{1}{2} \sum_{t=0,1} \|\mu_{z^L,\,\text{theo.}} - t\|^2 + \operatorname{tr} \Sigma_{z^L,\,\text{theo.}}, \tag{90}$$

where $\hat{\mathcal{L}}_{\text{stat.}}\big(\{\mu^k_x,\, \Sigma^k_x\}_{k=1,2}; \theta\big) = \mathcal{L}_{\text{stat.}}\big(\{\hat{f}_\mu(\mu^k_x,\, \Sigma^k_x;\, \theta),\, \hat{f}_\Sigma(\mu^k_x,\, \Sigma^k_x;\, \theta)\}_{k=1,2}; \theta\big)$.

The first term in the above expression is minimized for $z^L = g_\theta(x) \equiv 0.5$. However, network training does not precisely yield this solution, but constrains the network output to values close to this solution so that $g_\theta(x) \approx 0.5$ for any $x \in \mathbb{R}^{d_{\text{in}}}$, which matches the behavior of the network illustrated in Fig. 22(b). As described in Section 3.2, the threshold for the assignment of the predicted labels $\hat{t}$ is set to 0.5. Hence, the network behaves similarly to assigning class labels by chance, which explains the obtained performance value.

When training on data samples, we achieve a performance of $P_{\theta,\,\text{samples}} = 91.5\%$ for the network initialized to the same parameters. As a cross-check, we train on the compo-

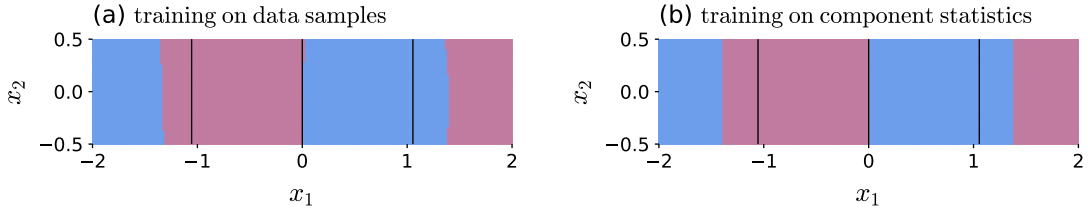


Figure 23: Decision boundaries implemented by a network of depth $L = 2$ and width $N = 10$ for the problem of alternating hills. In (a), the network was trained on data samples. In (b), the network was trained on the componentwise means and covariances. All points within a region of one color (blue, red) are assigned to the respective class label ($t = 0, 1$). We show the optimal decision boundaries (black lines) as reference.

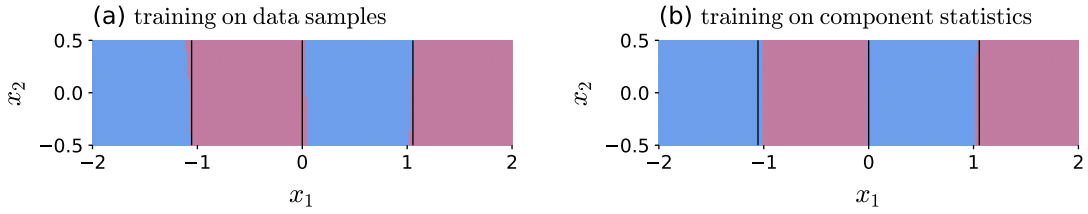


Figure 24: Decision boundaries implemented by a network of depth $L = 4$ and width $N = 10$ for the problem of alternating hills. In (a), the network was trained on data samples. In (b), the network was trained on the componentwise means and covariances. All points within a region of one color (blue, red) are assigned to the respective class label ($t = 0, 1$). We show the optimal decision boundaries (black lines) as reference.

nentwise means and covariances, which yields a similar performance of $P_{\theta,\,\text{componentwise}} = 90.5\%$. The respective decision boundaries for the trained networks are shown in Fig. 23.

Both networks exhibit a similar behavior: In each case, one of the resulting decision boundaries aligns well to the optimal decision boundary at $x_1 = 0$. The other two decision boundaries are however set further outwards from the center in comparison to the optimal decision boundaries. This explains the slight gap between the achieved performances and the upper limit P_{max}. For deeper network architectures, however, we can achieve performance values close to this upper limit when using the latter two methods. In these cases, the decision boundaries of the trained networks agree well with the optimal decision boundaries (see Fig. 24). Therefore, the performance gap for $L = 2$ might result from constraints of the network mapping due to the particular network architecture.

As in previous sections, we check whether the behavior of the three different training methods is consistent across different training runs. To this end, for each training method we calculate the achieved performance averaged over networks that are initialized on $n_{\text{seeds}} = 10^2$ different seeds. Furthermore, we compare the obtained results for network architectures of different depth L and width N, which are collectively presented in Fig. 25. We see that training on classwise means and covariances consistently yields performance values that correspond to assigning class labels by chance while the other two methods achieve performance values close to the upper limit P_{max}.

The observed difference in achieved performance stems from the fact that the latter two training methods can leverage information essential for solving the given task. As discussed in the previous section, this information is contained in higher order cumulants

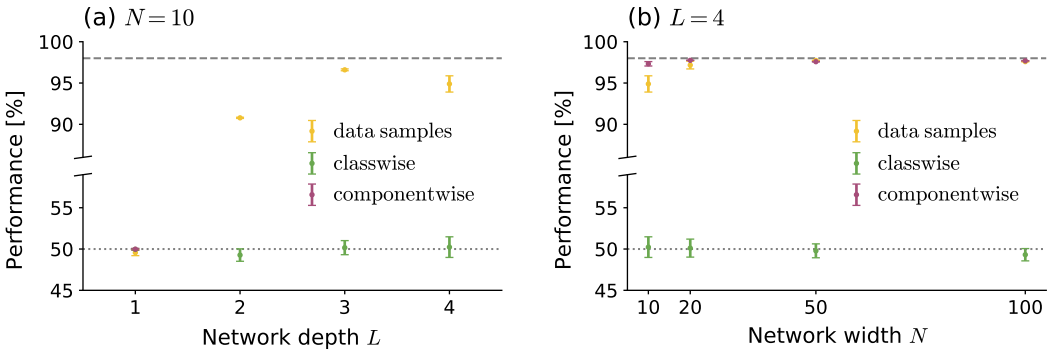


Figure 25: Achieved classification performance for different network architectures, averaged over networks that are initialized on $n_{\text{seeds}} = 10^2$ different seeds. The networks were trained using different training methods to solve the problem of alternating hills. We show both mean and corresponding standard error. The theoretical performance limit of $P_{\text{max}} = 98.0\%$ (gray dashed line) and the performance value of $P_{\text{chance}} = 50\%$ corresponding to class assignment by chance (gray dotted line) are given as references.

of the classwise distributions beyond mean and covariance. Training on data samples can probably infer these quantities from given input data. When training on componentwise means and covariances, the classwise distributions are given as superpositions of the corresponding components. These thereby effectively include higher order cumulants beyond mean and covariance. While these assertions seem sensible on a conceptual level, we yet need to verify them, which we do in Section 9.1.

The case $L = 1$ forms an exception with respect to the achieved performance values shown in Fig. 25(a): In addition to training on classwise means and covariances, both training on data samples and training on componentwise means and covariances do not find a network mapping that solves the given task. Effectively, this network architecture and in particular the quadratic activation function together with the used loss function are not powerful enough to leverage information from higher order cumulants beyond mean and covariance of each class to separate these classes. We discuss this effect in more detail in Chapter 10.

As in Section 5.1, we can alternatively determine empirical estimates of the classwise means and covariances and train networks using these estimates instead of the exact values. Even though these empirical estimates differ marginally between the two classes, this difference is an artifact of the particular dataset and thus does not contain any information regarding class membership. In line with expectations, we thus obtain very similar results when using empirical estimates of the classwise means and covariances for network training.

Part II

Higher Order Correlations in Neural Networks

8 Transformation of higher order statistics within networks

In previous chapters, we saw examples where higher order cumulants beyond mean and covariance contain information relevant to solving a given task and are thus important for understanding the solution strategy chosen by the network. Furthermore, we found that deviations between theory and simulation increase for deeper network architectures. These deviations result from an accumulation of approximation errors due to neglecting higher order cumulants.

Therefore, we here study the influence of contributions from these higher order cumulants (see Section 8.1). To this end, we extend the Gaussian description of the network mapping derived in Section 2.3 to consistently include contributions from higher order cumulants (see Section 8.2). We compare the resulting theoretical values with both empirical results and theoretical values determined using the Gaussian description of the network mapping for untrained networks (see Section 8.3).

8.1 Cumulant transformation by a single network layer

We consider the case that the layer input $x \in \mathbb{R}^{N_\text{in}}$ follows an arbitrary non-Gaussian distribution $p_x(x)$. Thus, the distribution of the input data is described by non-zero cumulants $G_x^{(n)}$ of arbitrary order n. Since taking into account cumulants up to arbitrary order is infeasible, we truncate the series of cumulants at a certain order n_max. This truncation corresponds to approximating the data distribution as

$$p_x(x|G_x^{(1)}, G_x^{(2)}, \dots, G_x^{(n_\text{max})}, \dots) \approx \hat{p}_x(x|G_x^{(1)}, G_x^{(2)}, \dots, G_x^{(n_\text{max})}). \tag{91}$$

Based on this, we want to determine an approximation $\hat{p}_y(y|G_y^{(1)}, G_y^{(2)}, \dots, G_y^{(\tilde{n}_\text{max})})$ for the distribution of the layer output y including higher order cumulants beyond mean and covariance up to order $\tilde{n}_\text{max}$. To keep the notation concise, we drop the layer index l in this section.

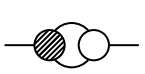
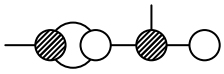
For the pre-activations $z = Wx + b$, we derived relations for cumulants of arbitrary order in Section 2.1 which are given by

$$G^{(n)}_{z,(r_1,\dots,r_n)} = \begin{cases} \sum_s w_{rs}\, G^{(1)}_{x,(s)} + b_r & \text{for } n = 1, \\ \sum_{s_1,\dots,s_n} w_{r_1 s_1} \dots w_{r_n s_n}\, G^{(n)}_{x,(s_1,\dots,s_n)} & \text{else.} \end{cases}$$

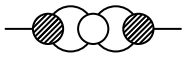
The analytical derivations in Section 2.1 for the cumulants of the post-activations $y = z + \alpha z^2$ rely entirely on the fact that we perform an average with regard to a Gaussian distribution, which does not apply in this case. Nonetheless, the expressions

for cumulants $G_y^{(n)}$ of order n (see Eq. (20) in Section 2.1) still account for any contribution that is generated exclusively by mean μ_z and covariance Σ_z of the pre-activations. To take into account additional contributions that involve higher order cumulants, we here use Feynman diagrams as described in Section 2.2. According to the rules for composing these diagrams, there are no additional contributions to the mean $\mu_y = G_y^{(1)}$ of the post-activations. For the covariance $\Sigma_y = G_y^{(2)}$, we additionally get:

$$\begin{aligned}
\Sigma_{y,rs}\,|_{\text{add.}} &= \alpha\left(G^{(3)}_{z,(r,s,s)} + G^{(3)}_{z,(s,r,r)}\right) + 2\alpha^2\left(G^{(1)}_{z,(r)}\,G^{(3)}_{z,(r,s,s)} + G^{(1)}_{z,(s)}\,G^{(3)}_{z,(s,r,r)}\right) \\
&\quad + \alpha^2\,G^{(4)}_{z,(r,r,s,s)}
\end{aligned}$$

$=$  $+$ 

(a) (b)

$+$ 

(c)

(92)

Since cumulants are symmetric with respect to their indices, we obtain symmetrized terms for any non-symmetric diagram, as in (92)(a) and (92)(b).

While the total number of Feynman diagrams and the corresponding contributions are manageable for both covariance and mean, the amount of diagrams increases significantly for higher order cumulants. In consequence, keeping track of all contributions to a cumulant of order n that are generated in a single network layer from cumulants up to order $n_{\max}$ becomes infeasible already at order $n = 3$. Instead, we need some guiding principle according to which we take into account contributions that involve higher order cumulants in a consistent manner.

8.2 Linear approximation to include higher order data statistics

The method of Feynman diagrams, which we use to determine contributions from higher order cumulants, is based on an expansion of the exponential term in the cumulant-generating function

$$\mathcal{W}_y(j) = \ln\,\left\langle \exp\left(j^\mathsf{T} y\right)\right\rangle_y \tag{93}$$

$$= \ln\,\left\langle \exp\left(j^\mathsf{T} z + \alpha\sum_r j_r\,z_r^2\right)\right\rangle_z \tag{94}$$

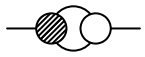
$$= \ln\,\left\langle \prod_r\left(\sum_{k=0}^{\infty}\frac{1}{k!}(j_r\,z_r + \alpha j_r\,z_r^2)^k\right)\right\rangle_z . \tag{95}$$

A natural way to arrange and subsequently truncate this expansion series is to use the strength α of the quadratic term in the non-linear activation function $\phi(z)$ as an expansion parameter. If the expansion parameter is small, all contributions that scale

with larger powers of this parameter become negligible. Thus, we get a sensible approximation for the cumulant-generating function as well as the cumulants of the layer output.

Since we want to determine corrections to our Gaussian description of the network mapping in Section 2.3, we apply this truncation exclusively to contributing terms that involve higher order cumulants beyond mean and covariance. Including contributions up to linear order in α and cumulants up to third order $G_z^{(3)}$, we get

$$\begin{aligned} \Sigma_{y,rs}\,|_{\text{add.}} &= \alpha \left(G^{(3)}_{z,(r,s,s)} + G^{(3)}_{z,(s,r,r)} \right) \\ &= \end{aligned}$$



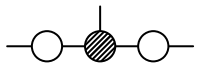
(a)

(96)

$$\begin{aligned} G^{(3)}_{y,(r,s,t)} &= G^{(3)}_{z,(r,s,t)} + \alpha \left(\Sigma_{z^l,rs}\,\Sigma_{z^l,st} + \Sigma_{z^l,sr}\,\Sigma_{z^l,rt} + \Sigma_{z^l,rt}\,\Sigma_{z^l,ts} \right) \\ &\quad + 2\,\alpha\, G^{(3)}_{z,(r,s,t)} \left(G^{(1)}_{z,(r)} + G^{(1)}_{z,(s)} + G^{(1)}_{z,(t)} \right) \\ &= \end{aligned}$$

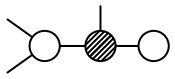
(a)

$+$



(b)

$+$



(c)

(97)

Using this truncation in each layer, we can iteratively determine theoretical values for the cumulants of the network output z^L up to order $n_{\max} = 3$ as

$$\left.\begin{aligned} \mu_{z^l,r} &= (W^l\,\mu_{y^l} + b^l)_r, \\ \mu_{y^{l+1},r} &= \mu_{z^l,r} + \alpha\,(\mu_{z^l,r})^2 + \alpha\,\Sigma_{z^l,rr}, \end{aligned}\right\} \quad \text{for } l = 0, \ldots, L-1$$

$$\mu_{z^L,r} = (W^L\,\mu_{y^L} + b^L)_r;$$

$$\left.\begin{aligned} \Sigma_{z^l,rs} &= \left(W^l\,\Sigma_{y^l}\,(W^l)^\mathsf{T}\right)_{rs}, \\ \Sigma_{y^{l+1},rs} &= \Sigma_{z^l,rs} + 2\,\alpha\,\left(\mu_{z^l,r}\,\Sigma_{z^l,rs} + \mu_{z^l,s}\,\Sigma_{z^l,sr}\right) \\ &\quad + 2\,\alpha^2\,(\Sigma_{z^l,rs})^2 + 4\,\alpha^2\,\mu_{z^l,r}\,\Sigma_{z^l,rs}\,\mu_{z^l,s} \\ &\quad + \alpha\left(G^{(3)}_{z,(r,s,s)} + G^{(3)}_{z,(s,r,r)}\right), \end{aligned}\right\} \quad \text{for } l = 0, \ldots, L-1$$

$$\Sigma_{z^L,rs} = \left(W^L\,\Sigma_{y^L}\,(W^L)^\mathsf{T}\right)_{rs};$$

$$
\left.
\begin{aligned}
G^{(3)}_{z^l,(r,s,t)} &= \sum_{u,v,w} (W^l)_{r\,u}\,(W^l)_{s\,v}\,(W^l)_{t\,w}\,G^{(n)}_{y^l,(u,v,w)}, \\
G^{(3)}_{y^{l+1},(r,s,t)} &= G^{(3)}_{z^l,(r,s,t)} \\
&\quad + \alpha\left(\Sigma_{z^l,rs}\,\Sigma_{z^l,st} + \Sigma_{z^l,sr}\,\Sigma_{z^l,rt} + \Sigma_{z^l,rt}\,\Sigma_{z^l,ts}\right) \\
&\quad + 2\,\alpha\,G^{(3)}_{z^l,(r,s,t)}\left(G^{(1)}_{z^l,(r)} + G^{(1)}_{z^l,(s)} + G^{(1)}_{z^l,(t)}\right),
\end{aligned}
\right\} \quad \text{for } l = 0,\ldots,L-1
$$

$$
G^{(3)}_{z^L,(r,s,t)} = \sum_{u,v,w} (W^L)_{r\,u}\,(W^L)_{s\,v}\,(W^L)_{t\,w}\,G^{(n)}_{y^L,(u,v,w)}.
$$

The contributions taken into account for our Gaussian description of the network mapping in Section 2.3 include terms up to $\mathcal{O}(\alpha^2)$. Accordingly, this would be a natural choice to truncate the expansion series when considering contributions from higher order cumulants beyond mean and covariance. However, the number of resulting Feynman diagrams is then neither instructive nor feasible. Thus, we here use an approximation up to linear order $\mathcal{O}(\alpha)$ in each layer.

8.3 Results for untrained networks

To assess the effect of including higher order cumulants beyond mean and covariance, we compare the theoretical results obtained using the expressions in the previous section with the theoretical results obtained when using the Gaussian description of the network mapping described in Section 2.3. We consider the case of untrained networks and use the same specifications for both the distribution of the input data and the network initialization as in Section 2.4.

Although the network input is Gaussian distributed, we get non-zero contributions to the third order cumulant in the first network layer when taking into account higher order cumulants up to linear order in α. We calculate theoretical values for the cumulants of the network output up to order $n = 3$ using the relations derived in the previous section. Based on these, we can determine theoretical curves for the probability density function of the network output for $d_{\text{out}} = 1$ by using the *Edgeworth expansion*. This expansion takes into account corrections to a Gaussian approximation resulting from higher order cumulants [29]. Up to first order, it is given by

$$
\hat{p}_{z^L,\,\text{Edgeworth}}(z^L) = \mathcal{N}(z^L|\mu_{z^L},\,\Sigma_{z^L})\left[1 - \frac{G^{(3)}_{z^L}}{6\,(\Sigma_{z^L})^{3/2}}\,\text{He}_3\!\left(\frac{z_L - \mu_L}{(\Sigma_{z^L})^{1/2}}\right)\right], \tag{98}
$$

where $\text{He}_3(z)$ denotes the *Hermite polynomial* of third order. Since this polynomial can yield negative values, we use

$$
p_{z^L,\,\text{theo., layerwise}}(z^L) = \min\left(0, \hat{p}_{z^L,\,\text{Edgeworth}}(z^L)\right) \tag{99}
$$

to ensure that the resulting theoretical curve for the probability density function is non-negative.

As the influence of higher order cumulants becomes more noticeable for deeper network architectures, we consider networks of depth $L = 4$ and width $N = 10$. We see that the theoretical curve derived from the linear approximation agrees better with the empirical distribution than the theoretical curve derived from the Gaussian description of the network output, as shown in Fig. 26(a). This decrease in deviations is also reflected in the relative Kullback-Leibler divergence, which yields

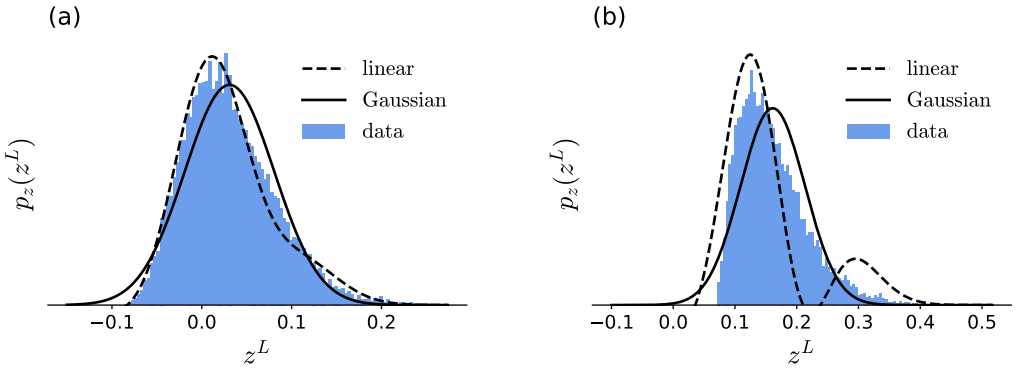


Figure 26: Distribution of the network output $z^L \in \mathbb{R}$ for networks with depth $L = 4$ and width $N = 10$ that are initialized to different network parameters in (a) and (b), respectively. A normalized histogram (blue) of the network output for a dataset of size $D = 10^4$ acts as an empirical estimate of the probability density function. We show theoretical curves for the probability density function derived from both the linear approximation described in Section 8.2 (dashed black line) and the Gaussian description of the network output described in Section 2.3 (solid black line).

$\hat{D}_{KL}(p_{z^L,\,\text{emp.}} \| p_{z^L,\,\text{theo.,\,layerwise}}) = 0.3\%$ compared to $\hat{D}_{KL}(p_{z^L,\,\text{emp.}} \| p_{z^L,\,\text{theo.,\,Gaussian}}) = 1.0\%$. In particular, we see that the third order cumulant can account for some of the skewness of the empirical distribution.

However, the improved agreement between theory and simulation cannot be observed consistently across networks initialized to different network parameters. We show an example in Fig. 26(b) for which both theoretical curves exhibit deviations from the empirical distribution. The theoretical curve obtained using the linear approximation exhibits oscillations for larger values, which results from the Hermite polynomial in the Edgeworth expansion. Accordingly, the relative Kullback-Leibler divergence is given by $\hat{D}_{KL}(p_{z^L,\,\text{emp.}} \| p_{z^L,\,\text{theo.,\,layerwise}}) = 55.6\%$ and $\hat{D}_{KL}(p_{z^L,\,\text{emp.}} \| p_{z^L,\,\text{theo.,\,Gaussian}}) = 2.3\%$, respectively.

We compare the relative Kullback-Leibler divergence for the linear approximation and the Gaussian description of the network, averaged over networks initialized on $n_{\text{seeds}} = 10^3$ different seeds. The resulting values are shown for varying network depth L and width N in Fig. 27. In all cases, the resulting values for the linear approximation are significantly larger. Bearing in mind the logarithmic axis, we also see that the corresponding standard error is significantly larger, accounting for the larger variability with regard to different network initializations. This might be due to the Edgeworth expansion only constituting an approximation of the corresponding probability density function. Additionally, it might not be sufficient to take into account higher order cumulants only up to linear order in α, as in the layerwise approximation derived in the previous section. We further investigate the influence of higher order cumulants in the following chapters.

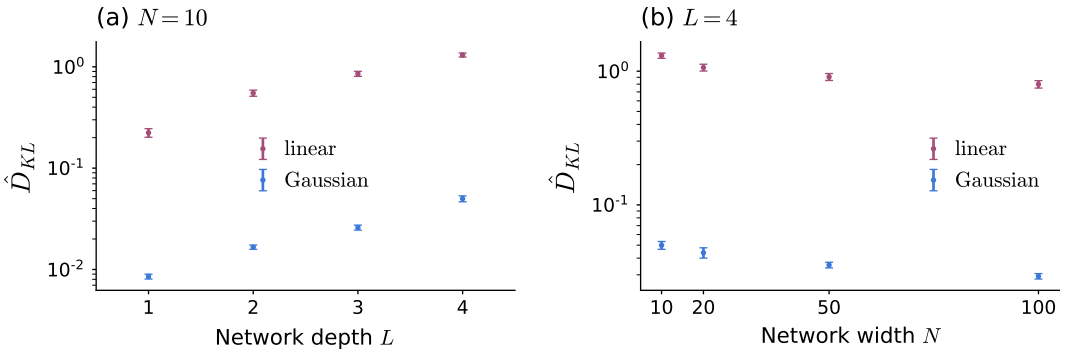


Figure 27: Relative Kullback-Leibler divergence for different network architectures, averaged over networks that are initialized on $n_{\text{seeds}} = 10^3$ different seeds. We show both mean and corresponding standard error for varying network depth L and width N, respectively. We determine theoretical values for the cumulants of the network output using either the linear approximation described in Section 8.2 (red) or the Gaussian description of the network described in Section 2.3 (blue).

9 Revisiting the performance gap

When comparing training on Gaussian data statistics and training on data samples, we discussed two classification tasks for which we observed non-negligible performance gaps between the two methods. These performance gaps probably arise because training on data samples infers information from higher order cumulants of the input distribution beyond mean and covariance.

In the previous chapter, we have adapted our description of the network mapping to include higher order cumulants. By combining the expressions derived in the previous chapter with the statistical formulation of the network loss in Section 4.1, we can include contributions from higher order cumulants when training on data statistics. With this training method, in the following we reexamine the performance gap for both the problem of alternating hills (see Section 9.1) and the MNIST database (see Section 9.2).

9.1 Classification for the problem of alternating hills

In Section 7.1, we introduced the problem of alternating hills as an example task for which the classwise means and covariances are identical and consequently, higher order cumulants encode class membership. As expected, we found that training on classwise means and covariances fails while training on data samples yields high performance values close to the corresponding upper limit. Our hypothesis is that training on data samples can infer information from higher order cumulants to find a viable solution strategy. This hypothesis is supported by the fact that we found training on componentwise means and covariances to yield similar performance values as training on data samples. In this case, using the definition of the classwise distributions as Gaussian mixture models effectively entails information encoded in higher order cumulants which can then be leveraged to find a viable solution strategy.

To further check our hypothesis, we use the derivations in Section 8.2 to include the third order cumulants of the classwise distributions when training on data statistics. The corresponding approximation of the classwise distributions is shown in Fig. 28(b). We see that this approximation can partially account for differences in the classwise distributions. However, we also see that it cannot cover the exact classwise distributions to their full extent (see Fig. 28(a)).

For a network of depth $L = 2$ and width $N = 10$, we achieve a performance of $P_\theta = 50.1\%$, which corresponds to assigning class labels by chance. Accordingly, the decision boundaries implemented by the network as well as the assignment of class labels for the corresponding decision regions do not match the optimal decision boundaries and regions as shown in Fig. 29(a). The corresponding network loss during training tends towards large negative values (see Fig. 29(b)), even though it should be constrained to non-negative values according to theory. Thus, this behavior might explain the poor performance and we therefore reexamine the network loss used during training.

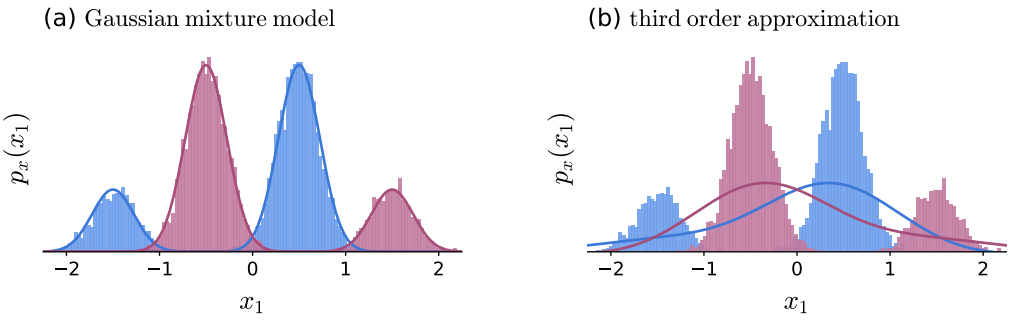


Figure 28: Distribution of data samples $x \in \mathbb{R}^2$ projected to the x_1-axis for the problem of alternating hills. Data samples are drawn from the Gaussian mixture model defined in (81)-(83) with class labels $t \in \{0, 1\}$ (blue, red) assigned accordingly. We show a histogram of the full dataset of size $D = 10^4$. We further show the classwise distributions (solid blue and red lines) given as the exact Gaussian mixture models in (a) as well as an approximation including cumulants up to the third order in (b).

When training on data samples, the network loss is given as the empirical average of the mean squared error between the network output $g_\theta(x^{(b)})$ and the corresponding target label $t^{(b)}$ with respect to a batch $\mathcal{B} = \{(x^{(b)}, t^{(b)})\}_{b=1,\dots,B}$ of data samples

$$\mathcal{L}\big(\{(x^{(b)}, t^{(b)})\}_{b=1,\dots,B};\, \theta\big) = \frac{1}{B}\sum_{b=1}^{B} \|g_\theta(x^{(b)}) - t^{(b)}\|^2. \tag{100}$$

Consequently, this quantity is non-negative.

When training on data statistics, we consider the limit of infinitely many data samples and rewrite the network loss as

$$\mathcal{L}_{\text{stat.}}\big(\{\mu_{z^L}^k,\, \Sigma_{z^L}^k\}_{k=1,\dots,K}\big) = \sum_{k=1}^{K} p_k \big(\operatorname{tr} \Sigma_{z^L}^k + \|\mu_{z^L}^k - t^k\|^2\big), \tag{101}$$

with $\mu_{z^L}^k$ and $\Sigma_{z^L}^k$ being the classwise means and covariances of the network output. Here, p_k denotes the probability of a sample to belong to class k.

Since the above expression is determined as the limit of a series of non-negative terms, it is also non-negative. To understand how the network loss used during training can become negative, we study the appearing terms. As $\|\mu_{z^L}^k - t^k\|^2$ is non-negative, this behavior is related to the trace of the covariance matrix which is given by the sum of eigenvalues in the case of a quadratic matrix. In general, the covariance matrix is positive definite which implies that its eigenvalues are all positive. Consequently, the above expression is positive as well. We also discussed that in our Gaussian description of the network mapping, the covariance matrix can become degenerate (see Section 2.1). Thus, the covariance matrix can be positive semi-definite and its eigenvalues are non-negative. Nonetheless, the above expression is then non-negative in both cases.

When training on data statistics, however, we determine an approximation $\Sigma_{z^L,\text{theo.}}^k$ based on the data statistics of the input data. If we include higher order cumulants, but only up to a certain order $n_{\max} \geq 3$, the resulting set of cumulants does not correspond to a proper distribution. Thus, truncating the number of considered cumulants at $n_{\max}$ for each network layer can result in a theoretical value $\Sigma_{z^L,\text{theo.}}^k$ of the classwise covariance

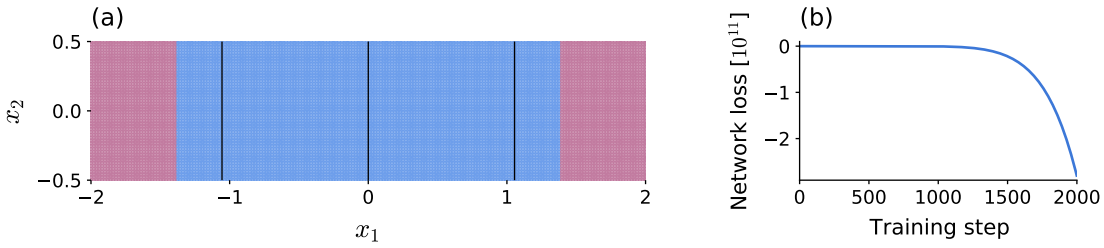


Figure 29: **(a)** Illustration of the resulting decision boundaries. All points within a region of one color (blue, red) are assigned to the respective class label ($t = 0, 1$). We show the optimal decision boundaries for solving the task (black lines) as reference. **(b)** Evolution of network loss during training for a network of depth $L = 2$ and width $N = 10$. The network was trained to solve the problem of alternating hills using the linear approximation including cumulants up to third order as described in Section 8.2.

of the network output with a negative eigenvalue. Since the network loss is minimized during training, this negative eigenvalue is subsequently amplified as it decreases the network loss in (101). Overall, this effect then leads to large negative values of the network loss during training on data statistics. In this case, the theoretical description of the network does not fit the network mapping and in consequence, training on data statistics yields poor performance values.

For the Gaussian description of the network mapping described in Section 2.3, this issue does not arise: In each layer, the layer input is taken to be Gaussian distributed. The expressions used to determine a theoretical value for the covariance of the layer output are exact for Gaussian distributed input data, thus again yielding a covariance matrix that is positive semi-definite. Approximating the distribution of the layer output as Gaussian at all layers corresponds to a proper distribution, in particular for the description of the network output. Consequently, the network loss used in training on Gaussian data statistics is constrained to non-negative values.

Based on these observations, we here use a different approach to include higher order cumulants: The information regarding the class membership that is encoded in the third order cumulant can be transferred to the covariance in a single network layer. Hence, in the first network layer we take into account all contributions to the covariance of the layer output that result from the third order cumulant of the classwise input distributions as derived in Section 8.1. In all subsequent network layers, we use a Gaussian description of the network mapping. Thus, we iteratively determine theoretical values for mean and covariance of the network output z^L according to

$$\left.\begin{aligned} \mu_{z^l,r} &= (W^l\,\mu_{y^l} + b^l)_r, \\ \mu_{y^{l+1},r} &= \mu_{z^l,r} + \alpha\,(\mu_{z^l,r})^2 + \alpha\,\Sigma_{z^l,rr} \end{aligned}\right\} \quad \text{for } l = 0, \ldots, L-1$$

$$\mu_{z^L,r} = (W^L\,\mu_{y^L} + b^L)_r,$$

$$
\left.
\begin{aligned}
\Sigma_{z^l,rs} &= \left(W^l\,\Sigma_{y^l}\,(W^l)^\mathsf{T}\right)_{rs},\\
\Sigma_{y^{l+1},rs} &= \Sigma_{z^l,rs} + 2\,\alpha\,\left(\mu_{z^l,r}\,\Sigma_{z^l,rs} + \mu_{z^l,s}\,\Sigma_{z^l,sr}\right)\\
&\quad + 2\,\alpha^2\,(\Sigma_{z^l,rs})^2 + 4\,\alpha^2\,\mu_{z^l,r}\,\Sigma_{z^l,rs}\,\mu_{z^l,s}\\
&\quad + \delta_{0l}\,\alpha\,\left(G^{(3)}_{z^l,(r,s,s)} + G^{(3)}_{z^l,(s,r,r)}\right)\\
&\quad + \delta_{0l}\,2\alpha^2\,\left(G^{(1)}_{z^l,(r)}\,G^{(3)}_{z^l,(r,s,s)} + G^{(1)}_{z^l,(s)}\,G^{(3)}_{z^l,(s,r,r)}\right)
\end{aligned}
\right\}
\quad \text{for } l = 0,\dots,L-1
$$

$$
\Sigma_{z^L,rs} = \left(W^L\,\Sigma_{y^L}\,(W^L)^\mathsf{T}\right)_{rs},
$$

$$
G^{(3)}_{z^0,(r,s,t)} = \sum_{u,v,w} (W^0)_{r\,u}\,(W^0)_{s\,v}\,(W^0)_{t\,w}\,G^{(n)}_{x,(u,v,w)}.
$$

We train networks based on these relations and average the achieved performance values over networks that are initialized on $n_{\text{seeds}} = 10^2$ different seeds. We compare the obtained results to the achieved performance values when training either on data samples or on classwise means and covariances. The results are collectively presented in Fig. 30 for network architectures of different depth L and width N. We see that training on classwise higher order cumulants beyond mean and covariance yields high performance values. With increasing network depth L, classification performance decreases as shown in Fig. 30(a). This probably results from the decrease in accuracy of the description of the network mapping for deeper networks. Transitioning to broader networks mitigates this effect as shown in Fig. 30(b), which is again in line with theory as discussed in Section 2.3. All in all, we see that the performance gap between training on data samples and training on data statistics can be bridged by including higher order cumulants beyond mean and covariance.

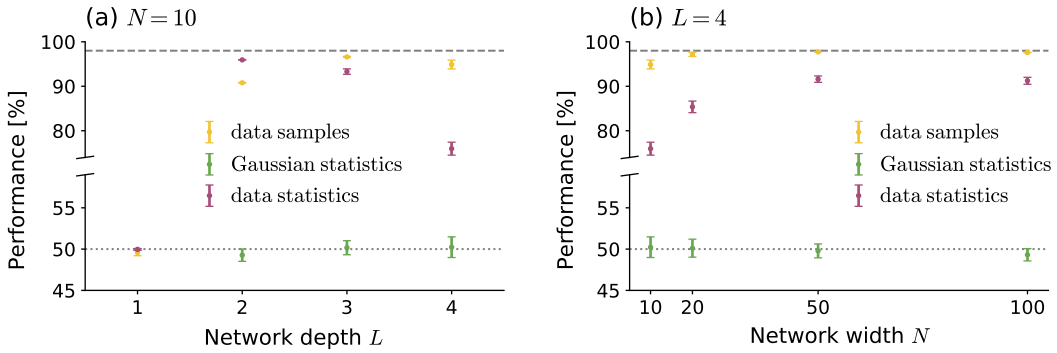


Figure 30: Achieved classification performance for different network architectures, averaged over networks that are initialized on $n_{\text{seeds}} = 10^2$ different seeds. The networks were trained using different training methods to solve the problem of alternating hills. We show both mean and corresponding standard error. The theoretical performance limit of $P_{\text{max}} = 98.0\%$ (gray dashed line) and the performance value of $P_{\text{chance}} = 50\%$ corresponding to assignment by chance (gray dotted line) are given as references.

The case $L = 1$ forms an exception with regard to these observations: Neither of the training methods discussed here yields a network mapping that solves the given task. In fact, this network architecture and in particular the quadratic activation function together with the used loss function are not powerful enough to leverage information from higher order cumulants beyond mean and covariance of each class to separate these

classes. We discuss this effect in more detail in Chapter 10.

Lastly, we point out that generally cumulants of fourth order in addition to third order contribute in the first network layer and therefore need to be taken into account to ensure that the resulting covariance matrix is positive semi-definite (see Eq. (92) in Section 8.1). However, network training did not behave as expected when using an accordingly adapted method and did not succeed at finding a solution strategy that solves the given task. This shortfall is probably due to numerical issues of the used implementation. Nonetheless, for this particular task the above description of the network mapping consistently yields proper values for mean and covariance of the network output. Therefore, in this thesis we presented the above approach including only cumulants up to third order.

9.2 Classification on MNIST

For the MNIST database, we found in Section 6.2 that, while training on Gaussian data statistics yields high performance values, there remains a non-negligible performance gap compared to training on data samples for the same reason as discussed in the previous section. We used either of the two presented methods that include the third order cumulant when training networks on the cumulants of the network input. In both cases, we faced the issue of a negative network loss which resulted in poor performance values. As discussed in the previous section, we would need to take into account the fourth order cumulant to resolve this issue.

However, higher order cumulants are in general computationally expensive with respect to the required memory: For a cumulant $G_x^{(n)}$ of order n, the total number of entries is given by d_{in}^n with d_{in} being the dimensionality of the input data $x \in \mathbb{R}^{d_{\text{in}}}$. Thus, this quantity scales exponentially with the cumulant order n which can be problematic, in particular for high-dimensional input spaces. For the MNIST database, the dimensionality of the input data is $d_{\text{in}} = 784$. Consequently, storing a single cumulant of fourth order requires approximately 1.5 TB memory and is thus not feasible.

A possible approach for reducing memory requirements is to decrease the dimensionality d_{in} of the input space. Many pixels at the image edges do not encode any information regarding the class membership, as can be seen in Fig. 31(a), and can therefore be omitted. Thus, for each image we remove a certain amount of pixels, thereby reducing the dimensionality d_{in} of the input space. The pixel positions to be excluded are selected randomly to eliminate any bias. To preferably remove pixels at the edges, we weight all pixel positions according to a two-dimensional zero-mean Gaussian distribution with variance $\sigma^2 = 0.1$ in each direction that is discretized on a 28×28 grid of the unit cell $[-1, 1]^2$. A certain fraction of pixel positions is then randomly selected according to this weighting (see Fig. 31(b)) and the corresponding pixels are removed from all images as illustrated in Fig. 31(c).

We assess how the classification performance is affected when applying the above pre-processing to the data prior to network training. We train networks of depth $L = 1$ and width $N = 100$ using either training on data samples or training on Gaussian data statistics. The achieved classification performance averaged over networks that are initialized on $n_{\text{seeds}} = 10$ different seeds is shown in Fig. 32. Up to a certain fraction of removed pixels, the performance values are approximately constant for training on data samples and even increase slightly for training on Gaussian data statistics. A possible explanation for the latter effect might be that removing pixels can reduce redundancy in the input data and thus facilitate finding an optimal solution strategy. However,

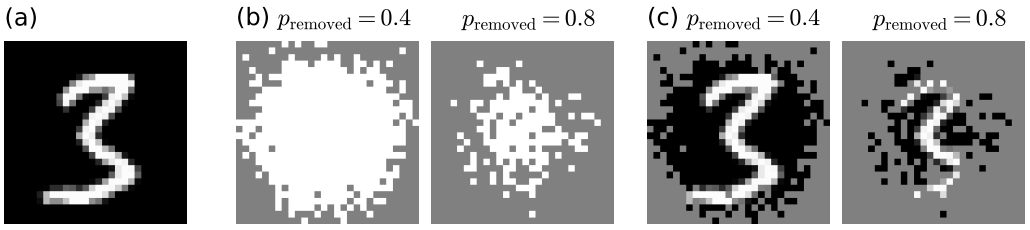


Figure 31: Illustration of data pre-processing in the case of the digit three. **(a)** Data sample from training dataset. **(b)** Selected pixel positions for different fractions p_{removed} of removed pixels. **(c)** Pre-processed data sample for different fractions p_{removed} of removed pixels.

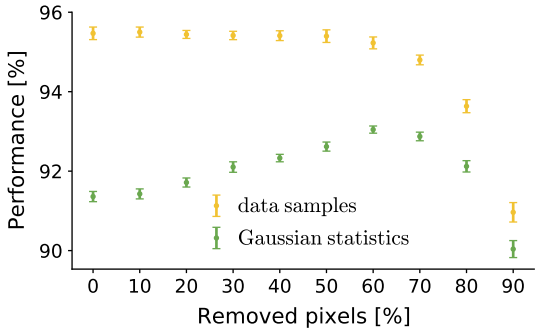


Figure 32: Achieved classification performance on pre-processed data from the MNIST database, averaged over networks of depth $L = 1$ and width $N = 100$ that are initialized on $n_{\text{seeds}} = 10$ different seeds. The networks were trained on either data samples (yellow) or the classwise means and covariances (green). We show both mean and corresponding standard error for different fractions of removed pixels.

removing 60% of image pixels or more leads to a decrease in classification performance, since in this case also pixels that contain relevant information are affected. When further increasing the fraction of removed pixels, the performance value for training on data samples even drops below the achieved performance for training on Gaussian data statistics without data pre-processing. Thus, $p_{\text{removed}} = 60\%$ is a suitable choice for training on data statistics when including higher order cumulants. Unfortunately, this choice still requires about 38.4 GB memory to store a single cumulant of fourth order, making this approach infeasible as well.

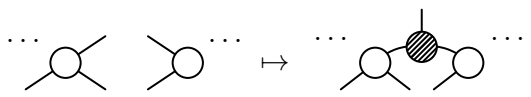
Other approaches to further reduce the dimensionality of the input space are applying a principal component analysis (PCA) [30] or auto-encoders [31] to the input data. Furthermore, we saw in the previous section that describing the classwise distribution as a Gaussian mixture model can account for higher order cumulants. Since both mean and covariance are significantly less expensive with regard to memory requirements than higher order cumulants, this offers another promising approach. Its feasibility depends strongly on the accuracy that can be obtained when fitting Gaussian mixture models to the classwise distributions. We plan to investigate the approaches discussed here in future research, but due to time constraints they are not part of this thesis.

10 Insights into the network expressivity

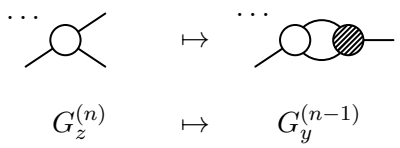
We studied the problem of alternating hills, which was introduced in Section 7.1 as an example where the classwise means and covariances are identical while higher order cumulants differ between classes. In Section 7.2, we found that training on classwise means and covariances fails, while training on data samples yields high performance values. Furthermore, we showed in Section 9.1 that including higher order cumulants when training on data statistics resolves the observed performance gap, yielding performance values close to the theoretical upper limit of $P_{\max} = 98.0\%$. This supports the notion that training on data samples can infer information contained in higher order cumulants.

However, we saw that networks of depth $L = 1$ on average yield performance values of $P_\theta \approx 50\%$ when trained with either of the above-mentioned methods, which corresponds to assigning class labels by chance. This observation raises the question whether such shallow network architectures can effectively leverage information contained in higher order cumulants beyond mean and covariances. Therefore, in this section we study which information is available to a certain network architecture with regard to the cumulants of the input data.

To this end, we reexamine the transformation of cumulants of arbitrary order by the quadratic term of the activation function $\phi(z)$ on a conceptual level. We use the graphical representation of the quadratic term as a two-point interaction vertex as described in Section 2.2. This interaction vertex can act in two ways, either joining two cumulant vertices by connecting to one internal line of each vertex, or modifying a single cumulant vertex by connecting two of its internal lines with one another. The order of the cumulant that is generated by the first operation is then given by $n+n'-1$, with n and n' being the orders of the joined cumulant vertices. In contrast, the second operation reduces the order n of the cumulant vertex by one, yielding a contribution to the cumulant of order $n-1$. In terms of Feynman diagrams, these two operations can be depicted as follows for $y = \phi(z)$:



$$G_z^{(n)}, \quad G_z^{(n')} \quad \mapsto \quad G_y^{(n+n'-1)} \tag{102}$$



$$G_z^{(n)} \quad \mapsto \quad G_y^{(n-1)} \tag{103}$$

Additionally, these operations can be combined and applied multiple times under the condition that the resulting Feynman diagram obeys the rules for composing such diagrams.

To address which information is available to the network in terms of the cumulants of the input data, we study the network loss used during training. For training on data samples, this quantity is given by the sample mean of the loss function $\ell(x, t; \theta)$ with respect to a batch of data samples $\mathcal{B} = \{(x^{(b)}, t^{(b)})\}_{b=1,\dots,B}$. In this thesis, we chose the mean squared error as loss function $\ell(x, t; \theta) = \|g_\theta(x) - t\|^2$. In Section 4.1, we derived that in the limit of infinitely many data samples the network loss can be written as

$$\mathcal{L}_{\text{stat.}}\big(\{\mu_{z^L}^k, \Sigma_{z^L}^k\}_{k=1,\dots,K}\big) = \sum_{k=1}^{K} p_k \big(\operatorname{tr} \Sigma_{z^L}^k + \|\mu_{z^L}^k - t^k\|^2\big),$$

with $\mu_{z^L}^k$ and $\Sigma_{z^L}^k$ being the classwise means and covariances of the network output z^L. Since network training aims at minimizing the network loss with respect to the network parameters, the network loss governs which information is used during training. Therefore, we effectively need to study the dependence of the appearing quantities on the order n of the classwise cumulants $G_x^{(n),\,k}$ of the input data. To keep the notation concise, in the following we drop the class index k.

For a network of depth $L = 1$, in Section 2.1 and Section 8.1 we derived the following dependencies

$$\mu_{z^1} = f_\mu\big(\{\mu_x, \Sigma_x\}; \theta\big), \tag{104}$$

$$\Sigma_{z^1} = f_\Sigma\big(\{\mu_x, \Sigma_x, G_x^{(3)}, G_x^{(4)}\}; \theta\big). \tag{105}$$

These results can also be understood in terms of the operations in (102) and (103). Since we are interested in the information available, we focus on the latter operation, which reduces the order of a cumulant by precisely one: In the case of a cumulant $G_z^{(n)}$ of order n, we can attach at most $\lfloor \frac{n}{2} \rfloor$ two-point interaction vertices in a single application of the quadratic activation function. This yields a contribution to the cumulant $G_y^{(\tilde{n})}$ of the transformed data $y = \phi(z)$ at order $\tilde{n} = \lfloor \frac{n+1}{2} \rfloor$. Furthermore, we have a contribution from $G_z^{(n)}$ to $G_y^{(n)}$ resulting from the linear term of the quadratic activation function $\phi(z)$. Taking into account all possibilities of attaching an intermediate number of two-point interaction vertices, we see that $G_z^{(n)}$ yields contributions to all cumulants $G_y^{(\tilde{n})}$ of order $\tilde{n} = \lfloor \frac{n+1}{2} \rfloor, \dots, n$. By reversing this argumentation, we get $G_y^{(n)} = f\big(\{G_z^{(n)}, \dots, G_z^{(2n)}\}; \alpha\big)$.

For arbitrary orders $n' < n$, we get a contribution to the cumulant $G_y^{(n)}$ of order n by applying the operation in (102) to $G_z^{(n')}$ and $G_z^{(n-n'+1)}$. Overall, we thus get $G_y^{(n)} = f\big(\{G_z^{(1)}, \dots, G_z^{(2n)}\}; \alpha\big)$.

Iterating this relation for a network of depth L, which corresponds to applying the quadratic activation function $\phi(z)$ exactly L times, yields

$$\mu_{z^L} = f_\mu\big(\{G_x^{(n)}\}_{n=1,\dots,2^L}; \theta\big), \tag{106}$$

$$\Sigma_{z^L} = f_\Sigma\big(\{G_x^{(n)}\}_{n=1,\ldots,2^{L+1}};\, \theta\big). \tag{107}$$

Thus, the maximal order of a cumulant $G_x^{(n)}$ of the input data that enters during training is given by $n_{\max,\,\Sigma_{z^L}} = 2^{L+1}$.

However, the method to assign predicted class labels used in this thesis requires the classwise means to be separated in the output space. Hence, it is not sufficient to have a network mapping that yields different covariances for each class. Instead, the methods used in this thesis require the classwise means to differ from one another. Consequently, a necessary condition to obtain high performance values is that at least one cumulant of order $n_{\max,\,\mu_{z^L}} = 2^L$ or lower contains sufficient information to solve the given task. In general, the fact that both $n_{\max,\,\mu_{z^L}}$ and $n_{\max,\,\Sigma_{z^L}}$ scale exponentially with the network depth L offers an explanation why adding a network layer can significantly improve the achievable classification performance.

These considerations allow us to understand the results obtained for networks of depth $L = 1$ for the problem of alternating hills in Section 7.2. In this case, the classwise means of the network output depend solely on the classwise means and covariances of the input data. Since these two quantities are identical for the two classes in this particular setup, networks of depth $L = 1$ cannot separate the two data classes in the network output and are thus not powerful enough to solve this task. This explanation is not to be confused with the argumentation for why training on classwise means and covariances cannot solve this task for any network architecture. This shortcoming results from using the description of the network as a non-linear mapping of mean and covariance for this specific training method. In this section, however, we consider the computational properties of the network mapping.

Lastly, we point out that the above relations strongly depend on various aspects: On the one hand, the chosen activation function and more precisely its polynomial order, if applicable, determine the maximal order of the input distribution's cumulants that yield contributions to a particular cumulant of the network output.

As an example, we consider the case of an additional cubic term in the activation function where $\tilde{\phi}(z) = z + \alpha\, z^2 + \beta\, z^3$. This introduces a three-point interaction vertex, which can reduce the order of a cumulant by at most two. Thus, it is $G_y^{(n)} = f\big(\{G_z^{(1)}, \ldots, G_z^{(3n)}\};\, \alpha,\, \beta\big)$ and the above results become

$$\mu_{z^L} = f_\mu\big(\{G_x^{(n)}\}_{n=1,\ldots,3^L};\, \theta\big), \tag{108}$$

$$\Sigma_{z^L} = f_\Sigma\big(\{G_x^{(n)}\}_{n=1,\ldots,3^{L+1}};\, \theta\big). \tag{109}$$

On the other hand, the chosen loss function determines the cumulants of the network output that arise and their relative dependence when evaluating the network loss in the limit of infinitely many data samples (see Section 4.1). In the case of the mean squared error, the classwise mean and covariance of the network output enter in an uncoupled manner. For other loss functions, higher order cumulants might occur and cumulants of arbitrary order may be coupled to one another in a non-linear manner.

Furthermore, the achievable performance is used as a measure for the expressivity of a particular network architecture. Since the method for performance evaluation can differ from the chosen network loss, its relations to the cumulants of the network output and to the network loss need to be taken into account.

All in all, the considerations presented here show that the joint description of network mapping and network loss that was derived in this thesis can offer explanations for differences in the network expressivity. As intuitively expected, these differences are

attributable to network architecture, training dynamics and performance evaluation as well as the interplay of these aspects.

Conclusion and Outlook

Despite many successes in recent years, deep neural networks are still lacking in transparency and interpretability. Obtaining an understanding of the mechanisms that neural networks employ to solve a given task is an important step towards making neural networks reliable tools in sensitive areas such as medical applications. Two main aspects in that regard are the functional principles of solution strategies that are learned by neural networks and the information utilized in these solution strategies. For supervised learning problems such as classification tasks, knowledge of the joint probability distribution of data samples and class labels allows the identification of a solution method that maximizes the average number of correctly classified data samples. Neural networks learn a solution strategy by inferring information from pairs of a data sample and corresponding target label so that they effectively rely on an empirical estimate of this joint distribution. Therefore, in this thesis we investigated how the information contained in the distribution of the input data, which we expressed in terms of its cumulants, enters in solution strategies learned by neural networks.

We here put our focus on classification tasks as these are common examples in machine learning. Furthermore, class membership is often to a large extent encoded in mean and covariance of the corresponding class. Thus, we can generally approximate the input distribution as a Gaussian mixture model with one component for each class. Based on this, a large part of this thesis was concerned with the transformation of Gaussian data statistics by neural networks. In a second part, we explored the influence of higher order cumulants beyond mean and covariance.

To solve a given classification task, we used feed forward networks with a quadratic activation function (see Chapter 1). These networks are hierarchically structured by network layers and therefore we traced the transformation of cumulants iteratively from layer to layer (see Chapter 2). For Gaussian distributed layer input, we derived exact expressions that describe the non-linear recombination of mean and covariance when applying a single network layer. Additionally, higher order cumulants are generated, thus rendering the resulting distribution of the layer output non-Gaussian. According to the disorder average, however, non-Gaussian terms scale down with the network width so that we can approximate the distribution at intermediate layers as Gaussian (see Appendix A.1). Thus, we obtained a description of the network as a non-linear mapping of mean and covariance. Using the relative Kullback-Leibler divergence as an accuracy measure, we found that the resulting theoretical curves for the probability density function of the network output agree well with the empirical results for both untrained and trained networks.

However, approximation errors due to neglecting higher order cumulants accumulate for deeper network architectures, leading to increased deviations between theory and simulation. This increase in deviations arises in particular for trained networks, as network training introduces dependencies among network parameters, whereas the

disorder average requires the network parameters to be independent from one another. Therefore, we adapted our description of the network mapping to include higher order cumulants consistently in each network layer (see Chapter 8). To this end, we determined corrections to the expressions for Gaussian distributed input data by using an expansion in terms of Feynman diagrams, which is a well-established method from statistical physics. We found that this adaptation can account for certain features such as skewness in the distribution of the network output, but does not consistently result in a higher accuracy of our theoretical description of the network mapping with respect to different network initializations. Furthermore, for certain choices of network parameters, the theoretical values for the covariance turned out negative because the approximation used in each layer does not necessarily correspond to a proper distribution.

As an approach to include higher order cumulants while ensuring that these belong to a proper distribution, we propose using a Gaussian mixture model in future work to describe the distribution at intermediate layers. This offers an elegant method to represent higher order cumulants [32], at least to a certain extent. Furthermore, this approach might allow us to obtain cohesive expressions to describe the transformation of cumulants by the network. An ensuing question is the number of mixture components necessary to match cumulants up to a certain order with a given accuracy. The overall number of mixture components used in the description of the network output then scales exponentially with this quantity. Thus, it might act as a complexity measure of the resulting distribution and possibly relate to the expressivity of a particular network architecture.

Throughout this thesis, we looked at three different classification tasks to study the accuracy of our network description in the case of trained networks as well as the solutions strategies implemented by these networks. As a first task, we introduced an adaptation of the XOR problem, which is a classical problem in machine learning (see Chapter 3). Since this problem is not linearly separable, it requires the use of a non-linear activation function and thus allows to study the influence of the non-linearity on the learned solution strategy. Furthermore, it constitutes an example task for which mean and covariance of each class contain sufficient information to solve this task. Thus, these quantities form a potential basis for viable solution strategies to this task.

To target the question of which information is required by the network, we developed a training method that allows us to directly train on cumulants of the input distribution (see Chapter 4): In the limit of infinitely many data samples, we derived an expression for the network loss used during training on data samples in terms of mean and covariance of the network output. Using the above results regarding the transformation of cumulants by the network, we can approximate the cumulants of the network output as functions of the cumulants of the network input and the network parameters. Combining these results allows us to train directly on cumulants of the input distribution.

For our adaptation of the XOR problem, we found that training on classwise means and covariances yields high performance values (see Chapter 5). This result confirms that the contained information is sufficient to be used for finding a viable solution strategy. Moreover, it shows that our description of the network as a non-linear mapping of mean and covariance is sufficiently accurate to serve as an adequate representation during training. Furthermore, for this task we found that there exist different information coding paradigms that can be used by the network to find a viable solution strategy. For a particular network architecture, we showed that the network finds at least two different solution strategies and studied how these relate to one another. We plan to

extend the investigation of these information coding paradigms and their dependence on different network initializations and architectures.

As a second task, we considered the MNIST database as an example of a non-synthetic dataset (see Chapter 6). In this case, a large proportion of the variability within each class can be accounted for by approximating the classwise input distributions as Gaussian. Accordingly, we found that training on classwise means and covariances already yields high performance values. However, there remained a non-negligible performance gap in comparison to training on data samples. This performance gap probably results from training on data samples being able to infer additional information contained in higher order cumulants. Taking these into account in a consistent manner necessitates cumulants up to the fourth order at the very least. However, the corresponding memory requirements scale exponentially with the cumulant order and are thus problematic for high-dimensional input spaces such as MNIST. Even after reducing the input dimensionality by removing less informative dimensions to an extent that only slightly affected classification performance, memory requirements still exceeded the resources at our disposal (see Chapter 9). We discussed above that describing the data distribution at intermediate layers as a Gaussian mixture model can account for higher order cumulants. This method can also be applied to the distribution of the input data. Since mean and covariance require significantly less memory, this offers a promising approach to take into account higher order cumulants, which we plan to investigate in the future.

Finally, we constructed a classification task to study the influence of higher order cumulants in more detail (see Chapter 7). For this task, mean and covariance of both classes are identical and consequently, the class membership is encoded in higher order cumulants. We defined the distribution of the input data as a Gaussian mixture model and named this task the problem of alternating hills due to its geometric layout. As expected, training on classwise means and covariances does not find a suitable solution while training on data samples achieves high performance values. We were able to bridge this performance gap by including the third order cumulant when training on data statistics since the third order cumulant encodes class membership (see Chapter 9). Alternatively, we can use the definition of the input distribution as a Gaussian mixture model to train on Gaussian data statistics but nevertheless account for higher order cumulants of the classwise distributions. For the problem of alternating hills, this method yields high performance values, further supporting such an approach for MNIST.

However, networks with a single non-linear layer did not find a viable solution for the problem of alternating hills when using either of the training methods discussed beforehand. Since we saw that including the third order cumulant is sufficient to achieve high performance values, we studied whether this particular network architecture is limited with respect to inferring information contained in higher order cumulants beyond mean and covariance. Based on the functional structure of Feynman diagrams, we determined the maximal cumulant orders that affect mean and covariance of the network output (see Chapter 10). In general, these two quantities together drive network training. However, a high classification performance necessitates a separation of means for the two classes. For a network with a single non-linear layer, the third order cumulant of the network input affects the covariance of the network output, but the mean of the network output depends solely on mean and covariance of the network input. As these two quantities are identical for both classes, this particular network architecture is indeed not powerful enough to solve the problem of alternating hills. Beyond this

example, these considerations allow us to gain insights into the network expressivity in terms of the information inferable for different network architectures. In particular, we saw that the network expressivity depends on different aspects such as choice of activation function, network loss or performance evaluation. In future work, we plan to investigate how these results relate to previous research on this topic, such as Raghu et al., 2017 [33], and Safran et al., 2020 [34].

Lastly, we point out that this thesis is embedded in a project that studies invertible neural networks and possible applications to medical data [35]. One example of such applications in previous studies is the analysis of EEG data [36, 8]. Due to their invertibility, these networks do not only allow processing data, but also generating synthetic data samples based on the learned network mapping [37]. From these synthetic data samples, it is possible to discern features learned by the network [38]. Furthermore, we can prescribe the distribution of the network output during network training, which is typically chosen to be Gaussian. Thus, by using the inverse mapping, we can study how the distribution of the input data is composed. Since feed forward networks often form building blocks in such network architectures [39, 40], we hope to extend the results obtained in this thesis to invertible neural networks.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3065386

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. [Online]. Available: http://www.nature.com/articles/nature16961

[3] Y. Bahri, J. Kadmon, J. Pennington, S. S. Schoenholz, J. Sohl-Dickstein, and S. Ganguli, "Statistical Mechanics of Deep Learning," *Annual Review of Condensed Matter Physics*, vol. 11, no. 1, pp. 501–528, Mar. 2020. [Online]. Available: https://www.annualreviews.org/doi/10.1146/annurev-conmatphys-031119-050745

[4] A. Holzinger, C. Biemann, C. S. Pattichis, and D. B. Kell, "What do we need to build explainable AI systems for the medical domain?" *arXiv:1712.09923 [cs, stat]*, Dec. 2017, arXiv: 1712.09923. [Online]. Available: http://arxiv.org/abs/1712.09923

[5] M. Hofmarcher, T. Unterthiner, J. Arjona-Medina, G. Klambauer, S. Hochreiter, and B. Nessler, "Visual Scene Understanding for Autonomous Driving Using Semantic Segmentation," in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller, Eds. Cham: Springer International Publishing, 2019, pp. 285–296. [Online]. Available: https://doi.org/10.1007/978-3-030-28954-6_15

[6] C. M. Bishop, *Pattern recognition and machine learning*, ser. Information science and statistics. New York: Springer, 2006.

[7] R. Schirrmeister, L. Gemein, K. Eggensperger, F. Hutter, and T. Ball, "Deep learning with convolutional neural networks for decoding and visualization of EEG pathology," in *2017 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*. Philadelphia, PA: IEEE, Dec. 2017, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/8257015/

[8] R. T. Schirrmeister, Y. Zhou, T. Ball, and D. Zhang, "Understanding Anomaly Detection with Deep Invertible Networks through Hierarchies of Distributions and Features," *arXiv:2006.10848 [cs, stat]*, Jun. 2020, arXiv: 2006.10848. [Online]. Available: http://arxiv.org/abs/2006.10848

[9] S. L. Oh, J. Vicnesh, E. J. Ciaccio, R. Yuvaraj, and U. R. Acharya, "Deep Convolutional Neural Network Model for Automated Diagnosis of Schizophrenia Using EEG Signals," *Applied Sciences*, vol. 9, no. 14, p. 2870, Jul. 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/14/2870

[10] J. Zinn-Justin, *Quantum field theory and critical phenomena*, 3rd ed., ser. Oxford science publications. Oxford : New York: Clarendon Press ; Oxford University Press, 1996, no. 92.

[11] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989. [Online]. Available: http://link.springer.com/10.1007/BF02551274

[12] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, Jan. 1993. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0893608005801315

[13] A. Pinkus, "Approximation theory of the MLP model in neural networks," *Acta Numerica*, vol. 8, pp. 143–195, Jan. 1999. [Online]. Available: https://www.cambridge.org/core/product/identifier/S0962492900002919/type/journal_article

[14] M. Helias and D. Dahmen, "Statistical field theory for neural networks," *arXiv:1901.10416 [cond-mat]*, Jan. 2019, arXiv: 1901.10416. [Online]. Available: http://arxiv.org/abs/1901.10416

[15] E. Lukacs, "A Survey of the Theory of Characteristic Functions," *Advances in Applied Probability*, vol. 4, no. 1, pp. 1–38, 1972, publisher: Applied Probability Trust. [Online]. Available: http://www.jstor.org/stable/1425805

[16] A. Abbara, B. Aubin, F. Krzakala, and L. Zdeborová, "Rademacher complexity and spin glasses: A link between the replica and statistical theories of learning," *arXiv:1912.02729 [cond-mat, stat]*, Jun. 2020, arXiv: 1912.02729. [Online]. Available: http://arxiv.org/abs/1912.02729

[17] Radford M. Neal, "Bayesian Learning for Neural Networks," Ph.D. dissertation, Dept. of Computer Science, University of Toronto, 1994.

[18] C. K. I. Williams, "Computing with Infinite Networks," *Advances in neural information processing systems*, pp. 295–301, 1997.

[19] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, "Deep Neural Networks as Gaussian Processes," *arXiv:1711.00165 [cs, stat]*, Mar. 2018, arXiv: 1711.00165. [Online]. Available: http://arxiv.org/abs/1711.00165

[20] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," *arXiv:1811.03378 [cs]*, Nov. 2018, arXiv: 1811.03378. [Online]. Available: http://arxiv.org/abs/1811.03378

[21] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 03 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[22] T. M. Cover and J. A. Thomas, *Elements of information theory*, ser. Wiley series in telecommunications. New York: Wiley, 1991.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986. [Online]. Available: http://www.nature.com/articles/323533a0

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, conference Name: Proceedings of the IEEE.

[26] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[27] D. Yadav, "Categorical encoding using Label-Encoding and One-Hot-Encoder," Dec. 2019. [Online]. Available: https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd

[28] P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, vol. 1. Edinburgh, UK: IEEE Comput. Soc, 2003, pp. 958–963. [Online]. Available: http://ieeexplore.ieee.org/document/1227801/

[29] S. Blinnikov and R. Moessner, "Expansions for nearly Gaussian distributions," *Astronomy and Astrophysics Supplement Series*, vol. 130, no. 1, pp. 193–205, May 1998. [Online]. Available: http://aas.aanda.org/10.1051/aas:1998221

[30] H. Hotelling, "Analysis of a complex of statistical variables into principal components." *Journal of Educational Psychology*, vol. 24, no. 6, pp. 417–441, 1933. [Online]. Available: http://content.apa.org/journals/edu/24/6/417

[31] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological Cybernetics*, vol. 59, no. 4-5, pp. 291–294, Sep. 1988. [Online]. Available: http://link.springer.com/10.1007/BF00332918

[32] C. S. Withers, S. Nadarajah, and S. H. Shih, "Moments and Cumulants of a Mixture," *Methodology and Computing in Applied Probability*, vol. 17, no. 3, pp. 541–564, Sep. 2015. [Online]. Available: http://link.springer.com/10.1007/s11009-013-9379-y

[33] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the Expressive Power of Deep Neural Networks," *arXiv:1606.05336 [cs, stat]*, Jun. 2017, arXiv: 1606.05336. [Online]. Available: http://arxiv.org/abs/1606.05336

[34] I. Safran and O. Shamir, "Depth-Width Tradeoffs in Approximating Natural Functions with Neural Networks," *arXiv:1610.09887 [cs, stat]*, May 2020, arXiv: 1610.09887. [Online]. Available: http://arxiv.org/abs/1610.09887

[35] "KI-Erklärbarkeit und Transparenz." [Online]. Available: https://www.softwaresysteme.pt-dlr.de/de/ki-erkl-rbarkeit-und-transparenz.php

[36] R. T. Schirrmeister and T. Ball, "Deep Invertible Networks for EEG-based brain-signal decoding," p. 11, 2019.

[37] L. Dinh, D. Krueger, and Y. Bengio, "NICE: Non-linear Independent Components Estimation," *arXiv:1410.8516 [cs]*, Apr. 2015, arXiv: 1410.8516. [Online]. Available: http://arxiv.org/abs/1410.8516

[38] L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, "Guided Image Generation with Conditional Invertible Neural Networks," *arXiv:1907.02392 [cs]*, Jul. 2019, arXiv: 1907.02392. [Online]. Available: http://arxiv.org/abs/1907.02392

[39] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using Real NVP," *arXiv:1605.08803 [cs, stat]*, Feb. 2017, arXiv: 1605.08803. [Online]. Available: http://arxiv.org/abs/1605.08803

[40] D. P. Kingma and P. Dhariwal, "Glow: Generative Flow with Invertible 1×1 Convolutions," p. 10.

[41] C. W. Gardiner, *Handbook of stochastic methods: for physics, chemistry and the natural sciences*, study ed., 2. ed., 6. print ed., ser. Springer series in synergetics. Berlin: Springer, 2002, no. 13, oCLC: 174775208.

[42] E. W. Weisstein, "Hyperbolic Tangent," publisher: Wolfram Research, Inc. [Online]. Available: https://mathworld.wolfram.com/HyperbolicTangent.html

[43] ——, "Bernoulli Number," publisher: Wolfram Research, Inc. [Online]. Available: https://mathworld.wolfram.com/BernoulliNumber.html

A Appendix

A.1 Disorder average for broad networks

The application of a quadratic activation function $\phi(z)$ in each network layer generates higher order cumulants from Gaussian distributed layer input, as derived in Section 2.1. For broad networks, we can however perform a disorder average with regard to the network parameters θ, by which follows that higher order cumulants of the pre-activations z beyond mean and covariance scale down with the layer width N_l. Consequently, these become negligible for broad networks where $N_l \gg 1$.

We assume the network parameters $\theta = \{W^l, b^l\}_{l=0,\dots,L}$ to be Gaussian distributed with

$$w_{rs}^l \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(0, \frac{\sigma_w^2}{N_{l-1}}\right), \tag{110}$$

$$b_r^l \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(0, \sigma_b^2\right). \tag{111}$$

This choice of initialization ensures that the magnitude of the covariance is preserved within the network as it does not scale with the layer width

$$\langle \Sigma_{z^l, rs} \rangle_\theta = \left\langle \left(W^l \, \Sigma_{y^l} \, (W^l)^\mathsf{T} \right)_{rs} \right\rangle_\theta \tag{112}$$

$$= \left\langle \sum_{t,u=1}^{N_{l-1}} w_{rt}^l \, w_{su}^l \, \Sigma_{y^l, tu} \right\rangle_\theta \tag{113}$$

$$= \delta_{rs} \, \frac{\sigma_w^2}{N_{l-1}} \sum_{t=1}^{N_{l-1}} \Sigma_{y^l, tt} \tag{114}$$

$$= \delta_{rs} \, \mathcal{O}_{N_{l-1}}(1). \tag{115}$$

According to the central limit theorem, a properly normalized sum of independent random variables tends toward a Gaussian distribution [41]. We apply this idea to the affine linear transformation in each layer, which is given by

$$z_r^l = \sum_{s=1}^{N_{l-1}} w_{rs}^l \, y_s^l + b_r^l. \tag{116}$$

We investigate the dependence of the corresponding cumulants $G_{z^l}^{(n)}$ on the layer width N_{l-1} by taking the average with regard to the network parameters θ. As these are independently distributed, we can perform the average layerwise. For Gaussian input data, we have in the first network layer

$$\langle \mu_{z^0, rs} \rangle_\theta \propto \langle w_{rs}^0 \rangle_{W^0} + \langle b_r^0 \rangle_{b^0} = 0, \tag{117}$$

$$\langle \Sigma_{z^0,\,rs} \rangle_\theta = \delta_{rs}\, \mathcal{O}_{d_{\text{in}}}(1). \tag{118}$$

In consequence, the expressions in (20) (see Section 2.1) yield

$$\left\langle G^{(n)}_{y^l,\,(r_1,\ldots,r_n)} \right\rangle_\theta \propto \delta_{r_1,\ldots r_n}. \tag{119}$$

After applying the affine linear transformation in the next layer, cumulants of odd order vanish since

$$\left\langle G^{(n)}_{z^l,\,(r_1,\ldots,r_n)} \right\rangle_\theta = \left\langle \sum_{s_1,\ldots,s_n=1}^{N_{l-1}} w^l_{r_1\,s_1} \ldots w^l_{r_n\,s_n}\, G^{(n)}_{y^l,\,(s_1,\ldots,s_n)} + \delta_{1n} b^l_{r_1} \right\rangle_\theta \tag{120}$$

$$\propto \langle w^l_{r_1 s_1} \rangle_{W^l} + \delta_{1n} \langle b^l_{r_1} \rangle_{b^l} = 0. \tag{121}$$

For even orders, it follows with (119) that

$$\left\langle G^{(n)}_{z^l,\,(r_1,\ldots,r_n)} \right\rangle_\theta = \left\langle \sum_{s_1,\ldots,s_n=1}^{N_{l-1}} w^l_{r_1\,s_1} \ldots w^l_{r_n\,s_n}\, G^{(n)}_{y^l,\,(s_1,\ldots,s_n)} \right\rangle_\theta \tag{122}$$

$$= \sum_{s=1}^{N_{l-1}} \left\langle w^l_{r_1\,s} \ldots w^l_{r_n\,s} \right\rangle_{W^l} \left\langle G^{(n)}_{y^l,\,(s,\ldots,s)} \right\rangle_{\theta'} \tag{123}$$

$$= \mathcal{O}\left(\left(\frac{\sigma_w^2}{N_{l-1}} \right)^{\frac{n}{2}} N_{l-1} \right) \tag{124}$$

$$= \mathcal{O}\left((N_{l-1})^{-\frac{n}{2}+1} \right). \tag{125}$$

Thus, cumulants beyond second order become negligible for $N_{l-1} \gg 1$ and the distribution at intermediate layers can be approximated as a Gaussian.

A.2 Approximate series expansion of the ReLU function

In this section, we investigate the relationship between the ReLU function and the quadratic activation function used in this thesis by deriving an approximate series expansion of the former. As $\text{ReLU}(z) = \max(0, z)$, we approximate the maximum function and get

$$\text{ReLU}\,(z) \approx \frac{z}{1 + \exp\left(-\gamma\, z\right)} \qquad \text{for } \gamma \gg 1. \tag{126}$$

The logistic function which appears in the expression above can be rewritten as

$$\frac{1}{1+\exp\left(-z\right)} = \frac{1}{2}\left[1 + \tanh\left(\frac{z}{2}\right)\right] \tag{127}$$

$$= \frac{1}{2} + \sum_{n=1}^{\infty} \frac{(2^n - 1)\; B_{2n}}{(2n)!}\, z^{2n-1}, \tag{128}$$

where we used the Taylor series of $\tanh(z)$ which holds for $|z| < \frac{\pi}{2}$ [42]. Here, B_n refers to the n^{th} Bernoulli number [43]. Thus, we get

$$\frac{z}{1 + \exp\left(-\gamma\, z\right)} = \frac{1}{2}\, z + \sum_{n=1}^{\infty} \frac{(2^n - 1)\; B_{2n}}{(2n)!}\, \gamma^{2n-1}\, x^{2n}. \tag{129}$$

Taking into account only leading order terms, we have

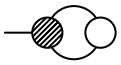
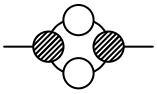
$$\mathrm{ReLU}\,(z) \approx \frac{1}{2}\,z + \frac{\gamma}{4}\,z^2 - \frac{\gamma^3}{48}\,z^4 + \mathcal{O}\left(\gamma^5\,z^6\right). \tag{130}$$

Since we assume $\gamma \gg 1$ in (126), the range for which this expression gives a reasonable approximation is constrained to values close to zero and highly dependent on the choice of γ. Nonetheless, we see that $\mathrm{ReLU}(z)$ can be described by a quadratic function in leading orders.

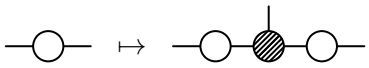
A.3 Diagrammatic resummation for Gaussian input data

In Section 2.1, we studied the transformation of Gaussian distributed input data by a single network layer with quadratic activation function. We derived exact expressions for the cumulants of the layer output by analytically calculating the corresponding cumulant-generating function. An alternative approach to determine the cumulants of the layer output is an expansion of the cumulant-generating function in terms of Feynman diagrams as described in Section 2.2. We here present how the expressions in Section 2.1 for cumulants of arbitrary order can be derived by using Feynman diagrams.

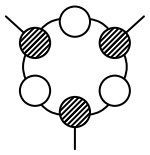
To this end, we compare the diagrammatic contributions in (23)(c) and (24)(c) to mean and covariance of the layer output, respectively, which are given by:

$\mu_{y,r}$: $\alpha\,\Sigma_{z,rr}$ $\quad=$  $\qquad$ $\Sigma_{y,rs}$: $2\,\alpha^2\,(\Sigma_{z,rs})^2$ $\quad=$  (131)

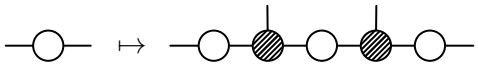
We see that these diagrams exhibit the same structure. More precisely, the diagram to the right can be generated by replacing:

 (132)

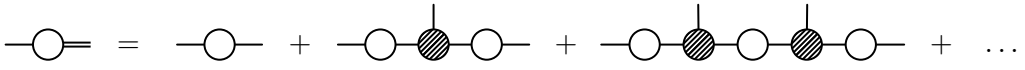
By repeating this operation, we obtain a Feynman diagram that yields a contribution to the cumulant of third order:

 (133)

Alternatively, this diagram can be created from the left diagram in (131) by directly substituting:

 (134)

We see that we can generate Feynman diagrams contributing to cumulants of arbitrary order by iterating this operation. We can collectively denote this operation by introducing an effective second order cumulant vertex corresponding to $\tilde{\Sigma}_z$:

 (135)

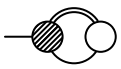
The resummation of the above diagrammatic components exactly corresponds to one of the equations obtained earlier (see Eq. (19) in Section 2.1)

$$\tilde{\Sigma}_z = \Sigma_z + \Sigma_z\,J\,\Sigma_z + \Sigma_z\,J\,\Sigma_z\,J\,\Sigma_z + \ldots \tag{136}$$

$$= \Sigma_z \sum_{k=0}^{\infty} (J\,\Sigma_z)^k \tag{137}$$

$$= \left(\Sigma_z^{-1} - J\right)^{-1}. \tag{138}$$

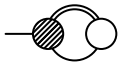
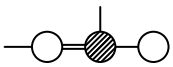
With the above notation, we can represent all Feynman diagrams that are generated in this way from the left diagram in (131) as

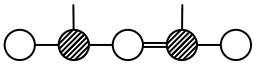


(139)

yielding $\frac{1}{2n}(2\alpha)^n\,\Sigma_{z,\,r_1r_2}\ldots\Sigma_{z,\,r_nr_1}$ as a contribution to $G^{(n)}_{y,\,(r_1,\ldots,r_n)}$ at order n. The factor $\frac{1}{n}$ results from the rotational symmetry of the diagram. In Section 2.1, this contribution originates from the term $-\frac{1}{2}\operatorname{tr}\ln\left(1-\Sigma_z J\right)$ where the trace operator accounts for the ring structure of the diagram.

In the case of Gaussian distributed layer input, all diagrammatic contributions to higher order cumulants can be represented by the following four diagrams:

(a)  (b) 

(c)  (d) 

(140)

For $G^{(n)}_{y,\,(r_1,\ldots,r_n)}$ with $n \geq 2$, these diagrams yield the expressions we derived earlier (see Eq. (20) in Section 2.1):

$$\begin{array}{llll} \text{(a)} & \frac{1}{2n}(2\alpha)^n\,\Sigma_{z,\,r_1r_2}\ldots\Sigma_{z,\,r_nr_1} & \text{(b)} & (2\alpha)^{n-1}\,\mu_{z,\,r_1}\,\Sigma_{z,\,r_1r_2}\ldots\Sigma_{z,\,r_{n-1}r_n} \\ \text{(c)} & \frac{1}{2}\,(2\alpha)^{n-2}\,\Sigma_{z,\,r_1r_2}\ldots\Sigma_{z,\,r_{n-1}r_n} & \text{(d)} & \frac{1}{2}\,(2\alpha)^{n}\,\mu_{z,\,r_1}\,\Sigma_{z,\,r_1r_2}\ldots\Sigma_{z,\,r_{n-1}r_n}\,\mu_{z,\,r_n} \end{array} \tag{141}$$

The case of $n = 1$ cannot completely be accounted for as there does not appear a second order cumulant vertex in two of the three appearing Feynman diagrams (see Eq. (23) in Section 2.1).

Overall, we obtain a compact representation of all contributing diagrams for higher order cumulants, which demonstrates the relation between Feynman diagrams and algebraic expressions.

A.4 Detailed calculations for the variance of the network loss for finite datasets

In this section, we present the detailed derivation for the variance of the network loss for finite datasets in the case of a single target label t:

$$\begin{aligned} &\operatorname{Var}\Big(\left(z^L\right)^2 - 2\,z^L\,t\Big) \\ &= \operatorname{Var}\Big(\left(z^L\right)^2\Big) + \operatorname{Var}\Big(-2\,z^L\,t\Big) + 2\operatorname{Cov}\Big(\left(z^L\right)^2, -2\,z^L\,t\Big) \\ &= \Big\langle \left(z^L\right)^4 \Big\rangle_{z^L\sim p(z^L|t)} - \Big\langle \left(z^L\right)^2 \Big\rangle^2_{z^L\sim p(z^L|t)} + 4t^2\operatorname{Var}\left(z^L\right) \\ &\quad - 4t\,\Big\langle \left(z^L\right)^3 \Big\rangle_{z^L\sim p(z^L|t)} + 4t\,G^{(1)}_{z^L}\,\Big\langle \left(z^L\right)^2 \Big\rangle_{z^L\sim p(z^L|t)} \end{aligned}$$

$$
\begin{aligned}
&= G_{z^L}^{(4)} + 4\, G_{z^L}^{(3)}\, G_{z^L}^{(1)} + 3\big(G_{z^L}^{(2)}\big)^2 + 6\, G_{z^L}^{(2)}\, \big(G_{z^L}^{(1)}\big)^2 + \big(G_{z^L}^{(1)}\big)^4 - \Big[G_{z^L}^{(2)} + \big(G_{z^L}^{(1)}\big)^2\Big]^2 \\
&\quad + 4t^2\, G_{z^L}^{(2)} - 4t\, \Big[G_{z^L}^{(3)} + 3\, G_{z^L}^{(2)}\, G_{z^L}^{(1)} + \big(G_{z^L}^{(1)}\big)^3\Big] + 4t\, G_{z^L}^{(1)}\, \Big[G_{z^L}^{(2)} + \big(G_{z^L}^{(1)}\big)^2\Big] \\
&= G_{z^L}^{(4)} + 2\big(G_{z^L}^{(2)}\big)^2 + 4\, G_{z^L}^{(3)}\, G_{z^L}^{(1)} - 4\, G_{z^L}^{(3)}\, t \\
&\quad + 4\, G_{z^L}^{(2)}\, \big(G_{z^L}^{(1)}\big)^2 + 4t^2\, G_{z^L}^{(2)} - 8t\, G_{z^L}^{(2)}\, G_{z^L}^{(1)} \\
&= G_{z^L}^{(4)} + 2\big(G_{z^L}^{(2)}\big)^2 + 4\, G_{z^L}^{(3)}\, \big[G_{z^L}^{(1)} - t\big] + 4\, G_{z^L}^{(2)}\, \big[G_{z^L}^{(1)} - t\big]^2 \\
&= G_{z^L}^{(4)} + 2\big(\Sigma_{z^L}\big)^2 + 4\, G_{z^L}^{(3)}\, \big[\mu_{z^L} - t\big] + 4\, \Sigma_{z^L}\, \big[\mu_{z^L} - t\big]^2 .
\end{aligned}
$$

A.5 Additional plots for binary classification on XOR

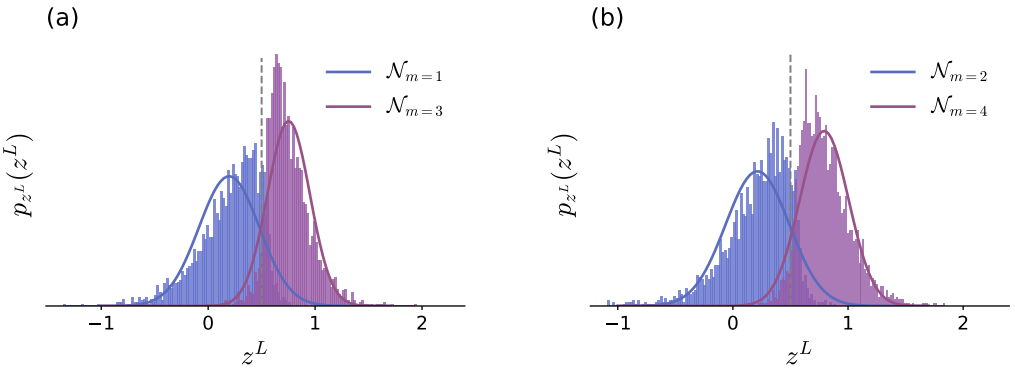


Figure 33: Distribution of the network output $z^L \in \mathbb{R}$ for a network of depth $L = 1$ and width $N = 10$ that is trained to solve the XOR problem. Normalized histograms of the network output for a test dataset of size $D_{\text{eval}} = 10^4$ act as empirical estimates of the probability density function. Their theoretical counterparts are given by Gaussian distributions (solid lines) with mean and covariance calculated according to Section 2.3. We show the individual distributions for all four components.

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)
Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature