Journal of Computational Science 36 (2019) 101020

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Generalization in fully-connected neural networks for time series forecasting

Anastasia Borovykh*, Cornelis W. Oosterlee, Sander M. Bohté

CWI Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 14 February 2019
Received in revised form 28 May 2019
Accepted 11 July 2019
Available online 8 August 2019

Keywords:
Neural networks
Deep learning
Generalization
Time series
Forecasting

ABSTRACT

In this paper we study the generalization capabilities of fully-connected neural networks trained in the context of time series forecasting. Time series do not satisfy the typical assumption in statistical learning theory of the data being i.i.d. samples from some data-generating distribution. We use the input and weight Hessians, that is the smoothness of the learned function with respect to the input and the width of the minimum in weight space, to quantify a network's ability to generalize to unseen data. While such generalization metrics have been studied extensively in the i.i.d. setting of for example image recognition, here we empirically validate their use in the task of time series forecasting. Furthermore we discuss how one can control the generalization capability of the network by means of the training process using the learning rate, batch size and the number of training iterations as controls. Using these hyperparameters one can efficiently control the complexity of the output function without imposing explicit constraints.
© 2019 Elsevier B.V. All rights reserved.

1. Introduction

Forecasting time series is an exceptionally difficult task due to the risk of overfitting on the dataset, in particular in the case of overparametrized networks [36,37]. In other words, when using the past to predict the future one has to be certain to have succeeded in extracting a signal from the past that will propagate to the future, and not simply fitted a complex function on the past. Neural networks, while being powerful function approximators that are relatively easy to optimize, can lead to poor extrapolation in time series forecasting due to the latter. Due to their ability to approximate almost any function it is of the essence to ensure that the network is learning the signal of interest instead of the noise. Understanding the structure of neural networks and the ability of a trained network to perform well on unseen data is therefore of utmost importance, and the main objective of this paper.

The loss surface of a neural network, defined as a function of the loss over the weights, is typically highly non-convex and can, for a deep network, depend on a large number of parameters (the weights). Even for a simple network, the number of local minima and saddle points in the loss surface may grow exponentially in the number of parameters. The general shape of this loss function, and also the differences in the loss functions of small and large neural networks, is an active topic of research [9,21]. In terms of theory, a recent line of work has related the neural network loss surface to Gaussian random fields [9,3,7,11]. Alternatively, random matrix theory has been used to obtain insight into the loss surface [27]. In more empirical lines of work, the authors of [21] found that adding more layers to a network gives rise to a more non-convex loss surface, so that adding more layers can complicate the training of the neural network by causing the optimisation methods to get stuck in sub-optimal critical points.

The above work gives insight into the structure of the loss surface on the training dataset. For noisy time series, a trained network is able to generalize well, that is perform well on unseen data, when it is not overfitting on noise in the training dataset. However, as is mentioned by the authors of [36], if the network is big enough (i.e. overparametrized) it can even fit a random noise dataset almost perfectly, but it will most certainly have bad performance out of sample. Understanding the structure of the minima so that a network will perform well on unseen data can give insight into setting up the training methods, for example to avoid convergence on noise.

There are different ways of measuring the learning capability of a neural network (see [5] for an overview). One is the output sensitivity, or the first derivatives, i.e. the Jacobian, of the error with respect to the input (see [25]) or the weights. The other measure is the statistical sensitivity which evaluates the output range variation of a node when its inputs or weights are perturbed. As is shown in [5], this is equivalent to considering second derivatives, the Hessian, of the loss function with respect to the input or weight values.

* Corresponding author.
E-mail addresses: anastasia.borovykh@cwi.nl (A. Borovykh), c.w.oosterlee@cwi.nl (C.W. Oosterlee), s.m.bohte@cwi.nl (S.M. Bohté).

https://doi.org/10.1016/j.jocs.2019.07.007
1877-7503/© 2019 Elsevier B.V. All rights reserved.

The statistical sensitivity with respect to the weights gives a measure for the smoothness of the error surface, with a small value of the statistical sensitivity implying a small output variation when weights are perturbed. The sensitivity with respect to the inputs measures the input noise immunity. Using the statistical sensitivity as a measure for generalization is intuitive in the sense that we are interested in the robustness of the network when the input is perturbed. It has also been proposed that the Hessian with respect to the weights can be used as a measure for generalization. These flat minima in the weight space correspond to simpler functions learned [16] or can be related to the Bayesian evidence [32].

Regularisation in the network can help to obtain a learned function with lower complexity such that a better generalization may be obtained. Typical explicit regularisation methods, such as L_1 or L_2 regularisation or multiplicative noise injection (such as dropout [34]), contribute to the generalizability of the trained function by restricting the function complexity in some way. For the regularisation method to work well, we need to understand how to make a trade-off between the complexity of the function and its ability to fit the data. This trade-off is known as the information bottleneck [35], and we study this in the coming sections to understand the effects of the trade-off on the learned function. Alternatively to explicit regularization, the noise in a stochastic gradient descent method (SGD) can act as an implicit regularizer. The gradient is computed over batches and, as opposed to computing the full gradient, SGD thus introduces non-isotropic noise into the optimisation scheme. It can be shown that this drives the parameters away from sharp minima towards the broader ones. In particular, the noise variance is proportional to the learning rate over the batch size, so that a large learning rate and small batch size result in a higher noise component. This has been shown in previous work, e.g. [32,8] and will be a focus of this work as well.

The novelty of our contribution consists in a thorough analysis of what generalizability means for time series forecasting with fully-connected neural networks. In particular, time series do not satisfy the typical assumption in statistical learning theory of i.i.d. data. Furthermore, while generalizibility for image datasets has been studied extensively, the problem is much more complex for time series: the dataset is typically much smaller, the signal-to-noise ratio might be low, the distribution can be non-stationary and there is little intuitive indication of what the underlying pattern in the data must be. Due to the non-i.i.d. nature the pattern might also change through time. Understanding what it means for a neural network to have good generalizibility, i.e. learning a consistently present pattern instead of overfitting on noise or on a changing pattern, and how this can be achieved through the learning algorithms will be the main task of this paper.

We assume that the reader is familiar with the general neural network concepts such as optimisation methods like stochastic gradient descent and its parameters and the neural network architectures. For a general introduction to this we refer to [6]. The rest of this paper is structured as follows: in Section 2 the loss surface structure is studied in a simplified setting; in Section 3 the input and weight Hessians are introduced as generalization metrics and the relation between the two is described; in Section 4 it is discussed how to make the trade-off between complexity and data fit and how one can influence complexity during the training of the network; finally Section 5 presents the numerical results.

2. Loss surface structure

In this section we give some background about neural networks and the properties of their loss surfaces and the implications of this structure for generalization capabilities.

2.1. Loss surface as a Gaussian random field

The loss surface of a neural network is defined as the loss function over the high-dimensional weight space. This loss surface can be related to a Gaussian process on a high-dimensional space [11,9]. With this insight, one is able to obtain theoretical results on the structure of the loss surface. We shortly repeat this derivation and discuss its implications. Let the inputs to the neural network be given by $x \in \mathbb{R}^{n_0}$. Let $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ be the weight matrix in layer l with element $w^{(l)}_{i,j}$ connecting neuron i in layer l and j in layer $l-1$. Define w as the vectorized total weights in the network, so that $w \in \mathbb{R}^d$ with $d = n_0 n_1 + n_1 n_2 + \cdots + n_{L-1} n_L$, with d thus being the dimension of the weight space. In the rest of this paper the dimension $\mathbb{R}^d$ refers to *column* vectors. The first layer output, for $l = 1$, is given by

$$a^{(1)} = f(z^{(1)}) = f\left(W^{(1)} x\right),$$

where $f(\cdot)$ is the non-linear activation function, $a^{(1)} \in \mathbb{R}^{n_1}$ is the activation of the first layer and $z^{(1)} \in \mathbb{R}^{n_1}$ is the pre-activation output. Each subsequent layer $l = 2, \ldots, L$ outputs,

$$a^{(l)} = f(z^{(l)}) = f\left(W^{(l)} a^{(l-1)}\right).$$

The final layer output is then given by,

$$\hat{y}(x, w) = q f(W^{(L)} f(W^{(L-1)} \ldots f(W^{(1)} x))) \ldots), \tag{1}$$

with q a scaling factor. Assume the data is given as a set of inputs x and outputs y generated from some data-generating distribution $\mathcal{D}$. Typical loss functions are the mean absolute error,

$$\mathcal{L}(x, w, y) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[|\hat{y}(x, w) - y|], \tag{2}$$

or the mean squared error,

$$\mathcal{L}(x, w, y) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[(\hat{y}(x, w) - y)^2].$$

where the expected values are taken over the data generating distribution.

We define a critical point w^* and its index α as follows,

Definition 1 *(A critical point and its index).* A critical point w^* of some differentiable function $g : \mathbb{R}^d \to \mathbb{R}$ is point $w^* \in \mathbb{R}^d$ where all partial derivatives of the function g are zero. In this work, we also refer to a critical point in a more loose definition as the point to which the optimization algorithm for the neural network has converged. For a function of d variables, the number of negative eigenvalues of the Hessian matrix H^w, the matrix of second-order derivatives of the loss function with respect to the parameters (defined more explicitly in (3)), at a critical point w^* is called the index α of the critical point.

Following the derivation in [9], let the non-linear activation function be the rectified linear unit defined as $f(x) = \max(x, 0)$ and replace the activation function in (1) by the term $A_{i,j} \in \{0, 1\}$, which denotes whether a path (i, j), where j labels any of the P paths from the input i to the output, is active or not. We obtain,

$$\hat{y}(x, w) = q \sum_{i=1}^{n_0} \sum_{j=1}^{P} x_i A_{i,j} \prod_{k=1}^{L} w^{(k)}_{i,j}.$$

Here x_i refers to the ith element of the input vector x and $P := n_1 n_2 \ldots n_L$ is the number of paths from a given network input to its output.

We now make the first key assumption that each path is equally likely to be active. The probability of a path being active follows a

Bernoulli distribution with probability ρ, independent of the input. Taking the expected value over the activation we obtain,

$$\mathbb{E}_A[\hat{y}(x,w)] = q\sum_{i=1}^{n_0} x_i \rho \sum_{j=1}^{P}\prod_{k=1}^{L} w_{i,j}^{(k)},$$

with i the summation over the inputs and $P = n_1, \ldots, n_L$ representing the summation over the further possible paths in the network. We remark that this expression is similar to a deep *linear* model multiplied by the factor ρ.

The second key assumption in this section is to let the input elements be sampled independently as $x_i \sim \mathcal{N}(0, \sigma_I^2)$ (and let $\sigma_I^2 = 1$ for simplicity). Due to the summation being over independent standard Gaussian random variables, $\hat{y}$ is equal to a Gaussian process on the weight space. Letting the loss function be given by the absolute loss as in (2) in which the expected value can be taken over the activations,

$$\mathcal{L}(x,w,y) = \mathbb{E}_A\left[|\hat{y}(x,w) - y|\right],$$

due to the x_i being sampled from a $\mathcal{N}(0,1)$ distribution, this loss function follows a Gaussian process distribution. For a particular value of q it is equal to the well-studied Hamiltonian of spin-glass systems [3] and previous work on Gaussian random fields can be applied [7,14] to gain insight into the structure of the critical points.

2.2. Structure of the critical points

In this section we briefly summarize the results on the loss surface structure of Gaussian random fields in high dimensions. The works of [7] and [14] show that for Gaussian random fields on high-dimensional spaces the critical points of the surface possess a particular structure. In [7] the authors show, by means of a generalized Kac–Rice formula, a linear dependence between the index of a critical point and its loss value (the error $\mathcal{E}$). A similar result can be obtained for neural networks as is done in [3,9]. Let Λ be the number of different weights in the network, which is assumed to be the Lth root of the total number of paths from input to output in the network,

$$\Lambda := \sqrt[L]{n_0 P}.$$

Under the assumptions made in Section 2.1, the loss surface of a neural network on a *high-dimensional* parameter space, in other words for deep and wide networks or as Λ increases, has the following properties,

1. let $\mathcal{E}_0 < \mathcal{E}_{m_1} < \mathcal{E}_{m_2} < \cdots$; there exists a layered structure of critical points: critical values in a band $(\mathcal{E}_0, \mathcal{E}_{m_1})$ above the global minimum $\mathcal{E}_0$ are more likely to be local minima, the band $(\mathcal{E}_{m_1}, \mathcal{E}_{m_2})$ consists of local minima and saddle points of index 1, the band $(\mathcal{E}_{m_2}, \mathcal{E}_{m_3})$ consists of local minima and saddle points of index 1 and 2, and so on;
2. local minima dominate over saddle points in a band of values close to the global minimum;
3. high-index critical points lie at high loss levels; in other words, a high value of α corresponds to a high loss level $\mathcal{E}$.

To conclude, by making several assumptions on the activation function and the distribution of the inputs, it is possible to relate the neural network loss function to a particular kind of Gaussian random field, as commonly encountered in spin-glass systems. By an application of the Kac–Rice theorem, one is able to obtain a relationship between the index of a critical point of this Gaussian random field and its value. It can be shown that high-index saddle points lie at high loss levels, while local minima are close to the global minimum.

2.3. Loss and the entropy

As was shown in the previous section, under certain – albeit restrictive – assumptions on the deep neural network, the loss surface is given by a Gaussian random field on a high-dimensional space; this space represents the weight space and its dimension is given by the number of weights in the network as determined via the number of nodes per layer and the number of layers used. Gaussian random fields on high-dimensional spaces posess a particular structure of the locations of the critical points in the asymptotic setting. Similarly, there exists a result on the entropy of these critical points. We recall here a result on the width of the minima, as stated in [4]. To measure the width of a minimum w^*, consider the entropy which is defined as,

$$S(w^*) = -\log\det(H^w(\mathcal{L}(x,w^*,y))),$$

with H^w being the Hessian matrix. A larger entropy means larger basin volume, or a wider minimum. We then state the following Theorem on the width of the minima, as is given and proved in [4].

Theorem 2 (Expected entropy [4]). *Let $\mathcal{E}$ be some loss level. The expected entropy of the Hessian of the loss function that takes value $\Lambda\mathcal{E}$ asymptotically, has the following expected entropy*

$$\mathbb{E}\left[S(w^*)|\Lambda\mathcal{E}\right] \simeq -(\Lambda-1)\log(\rho) + \frac{\Lambda-1}{2}\log\left(\frac{\Lambda}{2(\Lambda-1)L(L-1)}\right) - \frac{\Lambda-1}{\pi}\int_{-\sqrt{2}}^{\sqrt{2}} \log|\sigma\sqrt{\frac{\Lambda}{\Lambda-1}}\frac{\mathcal{E}}{\rho} - t|\sqrt{2-t^2}dt,$$

where L is the number of layers and ρ is the probability of the Bernoulli distributed weights being one.

The above theorem gives a relation between the number of layers in the network, the loss level at a particular point in the weight space and the width at that point in the weight space. In particular, the lower the train loss the lower the entropy and thus the sharper the minima. In other words, wider minima lie at higher loss values in the loss surface. This seems intuitive: in order to obtain a low train loss, one has to fit a more complex function which passes through all the observations. A good fit to the training data is however not sufficient to obtain good out of sample performance; we will discuss the concept of generalization in the next section.

2.4. Generalization

Since the data generating distribution $\mathcal{D}$ is typically unknown, in extrapolation problems one assumes to have access to samples (x^i, y^i) drawn (i.i.d.) from this distribution. One assumes $y^i = g(x^i) + \epsilon^i$, for some unknown function $g(\cdot)$; so that the y^i are noisy observations of the true function of interest. In our setting we are interested in time series forecasting, i.e. we have $x^i = (y^{i-n}, \ldots, y^{i-1})$ where i is the time index, so that n historical points of y are used to predict its future, in this setting one-day-ahead, value. This can be extended to x containing the historical observations of multiple time series used to forecast y. Note that here the i.i.d. assumption is violated since the observations should clearly be dependent. Nevertheless, using these datapoints as input a neural network can be used to extract a meaningful repeating pattern from the dataset. Note that the y^is, and thus the x^is, are *noisy* observations. We define the sample loss function as the loss function on that dataset, i.e. for the squared loss we obtain,

$$\hat{\mathcal{L}}(x,w,y) = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}(x^i,w) - y^i\right)^2,$$

where (x^i, y^i) for $i = 1, \ldots, N$ is the train dataset and $\hat{y}(x^i, w)$ the neural network output.

Generalization is the relationship between a trained networks' performance on train data versus its performance on test data. This is a highly desirable property for neural networks, where ideally the performance on the train data should be similar to the performance on similar but unseen test data. In general, the generalization error of a neural network model $\hat{y}(x, w)$ can be defined as the failure of the hypothesis $\hat{y}(x, w)$ to explain the dataset sample. It is measured by the discrepancy between the true error and the error on the sample dataset,

$$\mathcal{L}(x, w, y) - \hat{\mathcal{L}}(x, w, y).$$

In statistical learning theory a bound on this error is typically dependent on the complexity of the hypothesis class where the hypothesis $\hat{y}(x, w)$ is in, as well as on the number of samples in the dataset. Obtaining bounds on this error is a topic of active research with recent advancements including [13,38] where the authors use PAC-Bayes theory. In the rest of the paper we will use the notation $\mathcal{L}(x, w, y)$ to denote the empirical loss function as computed on the sampled data.

Typically, a trained network is able to generalize well when it is not overfitting on noise in the train dataset. Since neural networks are known to be universal approximators and thus – when the network is large enough – are able to approximate any function, when training one aims to extract a meaningful pattern in the data instead of learning a flexible function that is able to fit all training points. In particular in the setting of overparametrized networks it is easy for the network to fit the training points, however being able to avoid this overfitting is an essential task. A somewhat straightforward way to define the generalization capability is to study the robustness of the network with respect to input perturbations. For some input perturbation $\epsilon \in \mathbb{R}^{n_0}$, the change in the loss function should be small,

$$|\mathcal{L}(x + \epsilon, w, y) - \mathcal{L}(x, w, y)| < \delta.$$

When the neural network is heavily overfitting on the noise, a small change in the input parameters might result in a large change in the neural network output. In this setting the generalization is related to a smoothness assumption on the function output. In the coming sections our goal is to understand and be able to control the generalization of neural networks in the overparametrized, deep neural networks for time series forecasting, a setting in which the signal in the series can be weak and we lack the availability of large datasets. We aim to define metrics that can be used to measure when a learning algorithm can be expected to perform well.

3. Metrics for measuring the generalization

In this section we present metrics, the input and weight Hessians, for measuring the generalization capabilities.

3.1. The weight Hessian

The Hessian with respect to the weights will be used in order to obtain insight on the noise robustness of the weights, giving a metric for measuring the networks' capability to generalize well to unseen data.

3.1.1. Definition

The Hessian $H^w(\mathcal{L}) \in \mathbb{R}^{d\times d}$ of the loss function with respect to the weights has elements

$$h^w_{ij} = \partial_{w_i}\partial_{w_j}\mathcal{L}(x, w, y), \tag{3}$$

which represents the rate of change of the derivative with respect to w_j in the direction of w_i. The Hessian thus represents the curvature of the loss surface of the neural network. The eigenvectors and eigenvalues represent the direction and curvature in that direction, respectively. For the large neural networks typically used in image processing, computing and storing the Hessian can be very time-consuming. In the case of time series forecasting the networks used will be smaller, but nevertheless the Hessian can contain thousands of elements. In the rest of this paper we will sometimes drop the dependence of the loss function on the input x in case we are interested in the effects of the weights only.

The Hessian gives insight into the flatness of the minimum, and, as we will show in Section 3.3, this can be related to input noise resistance of the output function. In this sense the Hessian can be related to the minimum description length, where a Hessian with small eigenvalues corresponds to a simpler function being learned. Alternatively, the Hessian is used in second-order optimization methods where the step size in each direction is inversely proportional to the curvature in that direction: in directions with large curvature it takes small steps, while in directions with small curvature it takes larger steps [23].

3.1.2. Learning rate, batch size and the Hessian

It has been mentioned in prior research [17,22,32,8] that a relationship exists between the test error and the learning rate and batch size used in the SGD updating scheme. In this section we obtain a similar conclusion through a slightly different derivation. Let the gradient in a mini-batch $\mathcal{S}$ be $g_{\mathcal{S}} \in \mathbb{R}^d$ and the full gradient be $g \in \mathbb{R}^d$, where d is the weight space dimension, defined respectively as,

$$g_{\mathcal{S}} := \frac{1}{M}\sum_{i\in\mathcal{S}} \nabla_w \mathcal{L}(x^i, w, y) =: \frac{1}{M}\sum_{i\in\mathcal{S}} g_i, \quad g := \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\nabla_w \mathcal{L}(x, w, y)\right].$$

The weight update rule is given by,

$$w_{t+1} = w_t - \eta g_{\mathcal{S}},$$

where η is the learning rate. By the central limit theorem, if $(x_i, y_i)\sim\mathcal{D}$ i.i.d. then,

$$(g_{\mathcal{S}} - g)\sim\mathcal{N}\left(0, \frac{1}{M}K\right), \quad K = \mathbb{E}\left[(g_i - g)^T(g_i - g)\right]$$

$$\approx \frac{1}{N-1}\sum_{i=1}^{N}(g_i - g)^T(g_i - g).$$

Note that the approximation of the noise by a Gaussian distribution holds in the limit of the sample size tending to infinity and when the gradients for the batches are not heavy-tailed. Although the sample size is typically finite and the gradient distribution can be heavy-tailed, the approximation is widely used.

The weight update rule can then be rewritten as,

$$w_{t+1} = w_t - \eta g - \frac{\eta}{\sqrt{M}}\epsilon, \tag{4}$$

where $\epsilon\sim\mathcal{N}(0, K)$ with $\epsilon \in \mathbb{R}^d$. If convergence has been reached,

$$|\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t)| < \delta. \tag{5}$$

Note that for ease of notation we have omitted the dependence of the loss function on the input data (x, y). By a Taylor expansion method for the loss function evaluated on the full training data we have,

$$\mathcal{L}(w_{t+1}) \approx \mathcal{L}(w_t) - \left(\eta g - \frac{\eta}{\sqrt{M}}\epsilon\right)^T \nabla_w \mathcal{L}(w)$$

$$+ \frac{1}{2}\left(\eta g - \frac{\eta}{\sqrt{M}}\epsilon\right)^T H^w(\mathcal{L}(w_t))\left(\eta g - \frac{\eta}{\sqrt{M}}\epsilon\right), \tag{6}$$

where $\nabla_w \mathcal{L}(w) \in \mathbb{R}^d$ denotes the gradient of the total loss and $H^w(\mathcal{L}(w)) \in \mathbb{R}^{d\times d}$ the Hessian of the total loss. Note that we thus approximate the loss surface by a quadratic function. Then, using the convergence in (5) and the Taylor expansion for the loss function in (6), we obtain,

$$H^w(\mathcal{L}(w_t)) \approx \frac{\delta + 2\nabla_w \mathcal{L}(w_t)^T \left(\eta g - \frac{\eta}{\sqrt{M}}\epsilon\right)}{|\eta g - \frac{\eta}{\sqrt{M}}\epsilon|^2}$$
$$= \frac{\delta + 2\nabla_w \mathcal{L}(w_t)^T \left(g - \frac{1}{\sqrt{M}}\epsilon\right)}{\eta |g - \frac{1}{\sqrt{M}}\epsilon|^2}.$$

From this expression we see that the Hessian at convergence is small if a large learning rate or a small batch size has been used. Our simple derivation shows that if convergence is obtained, the Hessian in that point in weight space is smaller for larger learning rates and smaller batch sizes, i.e. the Hessian is inversely proportional to the fraction $\eta/\sqrt{M}$. In case of using full batch gradient descent, thus if $\epsilon = 0$, the relationship between convergence and the learning rate remains the same: convergence achieved with a large learning rate results in a smaller Hessian which in turn corresponds to a wider minimum.

3.1.3. The weight Hessian and generalization

Generalization in our setting refers to a kind of robustness of the trained network. More specifically, particular transformations of the input do not decrease the accuracy of the classification/forecast as computed on the train set. In other words, when a network has been trained on a set of patterns, certain transformations of these patterns should still be interpreted correctly. A higher robustness to input noise, which can be measured by Jacobians/Hessians with respect to input or weights, leads to better generalization, i.e. the smaller the Hessian the wider the minimum.

Consider the eigenspectrum of the Hessian, i.e. the set of eigenvalues (μ^i), $i = 1, \ldots, d$, determined via,

$$H^w v = \mu v,$$

where v is the eigenvector corresponding to the eigenvalue μ. If the eigenvalues of H^w are positive (resp. negative) at some critical point w^*, that point is a local minimum (resp. maximum), and if the critical point has both positive and negative eigenvalues it is called a saddle point (see also Definition 1 on the index of a critical point). The eigenvector corresponding to the largest eigenvalue of H^w indicates the direction of greatest curvature of the loss function. The size of the positive eigenvalues is thus a measure for how well a minimum will generalize to unseen data. A large positive eigenvalue in the direction of the corresponding eigenvector thus means that a sharp increase in loss will occur in that direction in weight space. If a minimum is wide, and thus has small eigenvalues in many directions, the minimum is better resistant to noisy transformations of the weights, while a sharp minimum has a higher sensitivity to the noise in the weights. A sharp minimum is thus said to have overfitted on the noise in the training dataset, while a wider minimum may imply that a 'simpler' and more robust function has been learned.

In the work of [16] the authors show that flat minima correspond to a minimization of the expected description length of the neural network function induced by the weights. The authors of [32] show a relationship between the Bayesian evidence and the Hessian, showing that maximizing the evidence corresponds to a minimization of the Hessian, by approximating the evidence with a Taylor expansion of the cost function.

3.1.4. Downsides of the weight Hessian

A metric that is commonly used to measure the width of the minimum is the trace. The trace of a squared matrix H of size $d \times d$ is defined as,

$$Tr(H) := \sum_{i=1}^{d} h_{ii} = \sum_{i=1}^{d} \mu_i,$$

where h_{ii} denotes the elements on the diagonal of the Hessian and μ_i denotes the eigenvalues. While prior research has noted that a lower Hessian (in terms of trace or some other norm) leads to better generalization, there has also been contradicting evidence. One critique on using the trace of the weight Hessian for measuring generalization capabilities is that the Hessian can be scaled in such a way that the output function remains the same but the trace of the Hessian can become large or small. This is the conclusion of the work of [12]. Consider a neural network with two layers, so that the output is given by

$$\hat{y}(x, w) = f(W^{(1)}x)W^{(2)}.$$

Note that if $f(\cdot)$ is the rectified linear unit $f(x) = \max(x, 0)$, then we can scale the weights by a constant $\alpha > 0$ without changing the output as follows,

$$\hat{y}(x, w) = f(\alpha W^{(1)}x)\alpha^{-1}W^{(2)}. \tag{7}$$

The gradient and Hessian of the loss $\mathcal{L}$ with respect to the weights w can be modified by α. We have,

$$\mathcal{L}(W^{(1)}, W^{(2)}) = \mathcal{L}(\alpha W^{(1)}, \alpha^{-1}W^{(2)}).$$

Then

$$\nabla_w \mathcal{L}(\alpha W^{(1)}, \alpha^{-1}W^{(2)}) = \nabla_w \mathcal{L}(W^{(1)}, W^{(2)}) \begin{bmatrix} \alpha^{-1} & 0 \\ 0 & \alpha \end{bmatrix},$$

where the derivatives are taken with respect to $\alpha W^{(1)}$ and $\alpha^{-1}W^{(2)}$. Similarly,

$$H^w(\mathcal{L}(\alpha W^{(1)}, \alpha^{-1}W^{(2)})) = \begin{bmatrix} \alpha^{-1} & 0 \\ 0 & \alpha \end{bmatrix} H^w(\mathcal{L}(W^{(1)}, W^{(2)})) \begin{bmatrix} \alpha^{-1} & 0 \\ 0 & \alpha \end{bmatrix}.$$

For large weights, e.g. when α is large, the second derivative with respect to these weights is smaller, however if the next-layer weights are scaled by α^{-1} the output function has not changed. Therefore, due to the many symmetries existing in a neural network, there exist symmetries that will not modify the output function and the generalization capabilities, but are able to scale the Hessian with respect to particular weights to be larger or smaller. Norms like the trace norm or the Frobenius norm are thus not enough to determine the generalization capabilities *if* one compares minima that are equivalent through these symmetries. This analysis thus shows that for each individual minimum found, a large trace of the Hessian is not informative. In [17] the authors claim that while these sharp minima with a generalization similar to a wide minima exist, SGD does not converge to these minima and for the minima found by SGD the width correlates well with generalization. We obtain a similar result in the numerical experiments in Section 5. This can be explained by the fact that the Hessians' sensitivity to scaling becomes an issue for $\alpha \ll 1$ or $\alpha \gg 1$, i.e. when the scaled and unscaled minima are far away in weight space, because then a minimum with a large Hessian norm can have similar generalization as a minimum with a small norm. We claim that for an SGD algorithm started from random initializations, if these initializations are relatively close in the weight space, the algorithm will not converge to minima that are far away. In other words, in order to compare the generalization capabilities of the minima found for

a particular network initialized from a particular distribution such that the initializations are close, the Hessian should measure the generalization capability well enough. As an alternative metric to study generalization we also propose the Hessian with respect to the input as introduced in the next section. This metric does not suffer from the scaling symmetries in the weight space of the deep neural network. A way to keep the weight Hessian invariant against these kind of transformations is to consider the weight Hessian multiplied by the weight matrix; note that

$$\begin{aligned}&\begin{bmatrix}\alpha(\hat{w}^{(1)})^T & \alpha^{-1}(\hat{w}^{(2)})^T\end{bmatrix} H^w(\mathcal{L}(\alpha W^{(1)}, \alpha^{-1}W^{(2)}))\begin{bmatrix}\alpha\hat{w}^{(1)}\\ \alpha^{-1}\hat{w}^{(2)}\end{bmatrix}\\ &\quad=\begin{bmatrix}(\hat{w}^{(1)})^T & (\hat{w}^{(2)})^T\end{bmatrix} H^w(\mathcal{L}(W^{(1)}, W^{(2)}))\begin{bmatrix}\hat{w}^{(1)}\\ \hat{w}^{(2)}\end{bmatrix},\end{aligned} \tag{8}$$

where $\hat{w}^{(1)}, \hat{w}^{(2)}$ denotes the vectorized forms of $W^{(1)}, W^{(2)}$, respectively. The Hessian multiplied by the weight matrix should be resistant against the scaling from (7) and result in a Hessian of similar size as the one of the original, unscaled weights. We study the this metric as a measure for generalization in the numerical experiments in Section 5.

3.2. The input Hessian

Besides the weight Hessian, the input Hessian will also be used as a metric for out-of-sample performance. In this section we discuss the relationship between input noise resistance of the network and the input and weight Hessians.

3.2.1. Definition

The curvature of the loss surface as a function of the weights is given by the Hessian with respect to the weights, as discussed previously. This measures the flatness of particular critical points, and correlates with generalization capabilities. Alternatively, as used in, e.g. [25], the Jacobian with respect to the input can be used as a measure for generalization. This measures the smoothness of the output function with respect to the input parameters. In [33] the authors also study this Jacobian as a measure for generalization and propose to explicitly penalize the Frobenius norm of the Jacobian with respect to the training data in the training objective to find minima that generalize well. For this local sensitivity metric one can define the Jacobian $J^x(\hat{y})$ with respect to the input x as a vector with elements

$$j_i^x = \partial_{x_i}\hat{y}(x, w),$$

where the network is considered to have one output node; it is trivial to extend the definition to multiple outputs. An input Jacobian with small elements, would imply that the output function is robust against small changes in the input data, meaning that a better generalization can be achieved.

Besides the Jacobian, the Hessian with respect to the input can also be used. This measures the curvature of the output function or loss function with respect to a varying input. To be consistent with the weight Hessian, we compute the Hessian with respect to the input of the loss function. This measures the curvature of the loss function, and thus the output function, with respect to the inputs, such that a Hessian with small eigenvalues means a smoother output function. Define the elements of the input Hessian $H^x(\mathcal{L}) \in \mathbb{R}^{n_0\times n_0}$ of an input $x \in \mathbb{R}^{n_0\times n_0}$ as,

$$h_{ij}^x = \partial_{x_i}\partial_{x_j}\mathcal{L}(x, w, y).$$

The Jacobian and the Hessian are averaged over the data samples x^i, $i=1, \ldots, N$ in the train dataset in order to obtain an average sensitivity metric over the input space.

3.2.2. The learning rate, batch size and input Hessian

In Section 3.1.2 we showed the effects of the learning rate and batch size on the weight Hessian. Here we show that SGD and its hyperparameters also put restrictions on the input Jacobian and Hessian. Consider SGD where the weight update rule is given by (4). By a first-order Taylor expansion in x for some noise $\tilde{\epsilon}$ it holds,

$$\begin{aligned}\nabla_w\mathcal{L}\left(x+\frac{1}{\sqrt{M}}\tilde{\epsilon}, w, y\right) &\approx \nabla_w\mathcal{L}(x, w, y) + \frac{1}{\sqrt{M}}\nabla_w\tilde{\epsilon}^T\nabla_x\mathcal{L}(x, w, y)\\ &=: \nabla_w\mathcal{L}(x, w, y) + \frac{1}{\sqrt{M}}\epsilon.\end{aligned}$$

In other words, (4) can be rewritten as,

$$w_{t+1} \approx w_t - \eta\tilde{g}, \qquad \tilde{g} := \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\nabla_w\mathcal{L}\left(x+\frac{1}{\sqrt{M}}\tilde{\epsilon}, w, y\right)\right].$$

Therefore, the noise from the stochastic gradient descent can be related to noise in the input and SGD can be interpreted to minimize a jittered cost function. In turn, by a derivation similar to [28], taking the loss function to be the MSE, one can find,

$$\begin{aligned}\mathcal{L}\left(x+\frac{1}{\sqrt{M}}\tilde{\epsilon}, w, y\right) &\approx \mathcal{L}(x, w, y)\\ &+ \frac{||K||_2^2}{M}||\nabla_x\tilde{y}(w, x)||^2 + \frac{||K||_2^4}{2M}||H^x(\tilde{y}(w, x))||_2^2,\end{aligned} \tag{9}$$

where $H^x(\tilde{y}(w, x))$ is the Hessian with respect to the input x of the neural network output $\tilde{y}(w, x)$ and the output function in the loss term on the r.h.s. is given by $\tilde{y}(x, w) = \hat{y}(x, w) + \frac{||K||}{2M}Tr(H^x(\hat{y}(x, w)))$. In other words, a relation exists between training with SGD and the minimization of the loss function regularized with the first- and second-order derivatives of the output with respect to the input. Therefore, SGD imposes smoothness assumptions on the output function with respect to the input.

3.3. Relation between the input and weight Hessian

Neural networks are considered to be robust if they are resistant to noise in the input. As mentioned in Section 3.1.4, flat minima in weight space are linked to good generalization; furthermore this flatness can be controlled through the learning rate. In this section we study the relation between flatness and the input noise robustness. Previous work [29] has also studied this relation, and the authors proposed an optimal learning rate to obtain good generalization. Consider the output of a one-layer neural network,

$$\hat{y}(x, w) = W^{(2)}f\left(W^{(1)}x\right)$$

where $W^{(2)} \in \mathbb{R}^{1\times n_1}$ (assuming the output $\hat{y}(x, w) \in \mathbb{R}$) and $W^{(1)} \in \mathbb{R}^{n_0\times n_1}$. Denote by $\hat{w}^1 \in \mathbb{R}^{n_0 n_1}$, $\hat{w}^2 \in \mathbb{R}^{n_1}$ the vectorized forms of $W^{(1)}, W^{(2)}$. We have,

$$\begin{aligned}&\hat{y}(x+\epsilon, w) = W^{(2)}f(W^{(1)}(x+\epsilon)) = W^{(2)}f((W^{(1)}+\tilde{\epsilon})x),\\ &\text{only if}\quad \tilde{\epsilon} = \frac{W^{(1)}\epsilon x^T}{||x||_2^2},\end{aligned} \tag{10}$$

where $\epsilon \in \mathbb{R}^{n_0}$ and $\tilde{\epsilon} \in \mathbb{R}^{n_0\times n_1}$. Let $\hat{\epsilon} \in \mathbb{R}^{n_0 n_1}$ be the vectorized form of $\tilde{\epsilon}$. Then,

$$\mathcal{L}(x+\epsilon, w, y) = \mathcal{L}(x, w + [\hat{\epsilon}, 0I_{n_1}]^T, y),$$

so that if the output is resistant to additive noise $\hat{\epsilon}$ in the first weight matrix $W^{(1)}$ then the output is resistant to additive noise ϵ in the input. Note that additive noise resistance is just one particular type of input transformation one might be interested in; other types could be, e.g. resistance to multiplicative input noise, or more complex transformations of the input space such as translations.

Consider now the Taylor expansion around the weights,

$$\mathcal{L}(x+\epsilon,w,y)=\mathcal{L}(x,w+[\hat{\epsilon},0I_{n_1}],y)=\mathcal{L}(x,w,y)+\hat{\epsilon}^T\nabla_{\hat{w}^1}\mathcal{L}(x,w,y)+\frac{1}{2}\hat{\epsilon}^T H^{\hat{w}^1}(\mathcal{L}(x,w,y))\hat{\epsilon}.$$

In order to have a minimum such that $\mathcal{L}(x+\epsilon,w,y)\approx\mathcal{L}(x,w,y)$, i.e. the output function should be resistant to additive noise in the input, the elements of the gradient and Hessian of the first-layer weights need to be small. The smoothness of the output function with respect to the input can be controlled during training in several ways. As we showed in Section 3.1.2, the flatness in weight space can be controlled through the learning rate or batch size in the SGD updates. Biasing the optimization algorithm into wider minima in weight space results in smoother functions with lower information complexity in the input space. The width is measured by the weight Hessian, so a lower weight Hessian should give better resistance to additive input noise.

In the above derivation we thus showed that the stability of the output function with respect to input noise is equivalent to noise resistance in the first layer weights. In order to see the effect of the stability requirement on further layer weights note that the expression in (10) can be written as,

$$\hat{y}(x+\epsilon,w)=W^{(2)}f((W^{(1)}+\tilde{\epsilon})x)=(W^{(2)}+\tilde{\epsilon}^2)f((W^{(1)}+\tilde{\epsilon}^1)x),$$

for some $\tilde{\epsilon}^2\in\mathbb{R}^{1\times n_1}$ and such that $\tilde{\epsilon}^1<\tilde{\epsilon}$. Let again $\hat{\epsilon}^1$, $\hat{\epsilon}^2$ be the vectorized forms of $\tilde{\epsilon}^1$, $\tilde{\epsilon}^2$. Then,

$$\mathcal{L}(x+\epsilon,w,y)=\mathcal{L}(x,w,y)+[(\hat{\epsilon}^1)^T,(\hat{\epsilon}^2)^T]\nabla_w\mathcal{L}(x,w,y)+\frac{1}{2}[(\hat{\epsilon}^1)^T,(\hat{\epsilon}^2)^T]H^w(\mathcal{L}(x,w,y))[(\hat{\epsilon}^1)^T,(\hat{\epsilon}^2)^T]^T.$$

From this expression we see that if the Hessian with respect to $W^{(1)}$ is not sufficiently small, for a deeper network the remaining noise can be damped by $W^{(2)}$ if the loss function is sufficiently flat in weight space in the directions of the second-layer weights. This may explain why deep networks can be more robust: the noise that is not fully dampened by sufficient flatness in the first-layer weight directions due to the eigenvectors in directions tangent to the first-layer weights having large eigenvalues (i.e. sharp increases of the loss function in those directions), can still be dampened in the further layers if the eigenvalues corresponding to the eigenvectors in the directions of the further weights are sufficiently small. On the other hand, adding nodes per layer does not aid in dampening the noise; on the contrary the more nodes per layer the smaller the eigenvalues of the Hessian in all the directions of these layer weights should be.

4. How low can we go?

As mentioned before, at least theoretically under particular assumptions, the entropy of a minimum decreases with the loss value. In other words, the lower the loss at a minimum, the larger the trace of the weight Hessian. A similar property can be said to hold for the input Hessian, with functions overfitting on the noise having lower train loss but a higher trace of the input Hessian. The trade-off between optimally representing the data and the smoothness of the function, i.e. the capability to compress the function by dismissing irrelevant input data, is known as the information bottleneck. In the optimal case, a neural network should learn to extract the most informative patterns, with the most compact function possible.

4.1. The information bottleneck

In the work of, e.g. [35,1,30] the information bottleneck is studied for neural networks. The information bottleneck is used to extract the most relevant information that the input variable contains about the output variable. Let $I(x;y)$ denote the mutual information where x and y are as usual random variables sampled from some data-generating distribution $\mathcal{D}$,

$$I(x;y)=\int_x\int_y p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right)dxdy,$$

where $p(x,y)$ is the joint probability density of x and y and $p(x)$ and $p(y)$ are the marginals of x and y respectively. The mutual information measures the information that x and y share; i.e. it quantifies the amount of information obtained about one of the random variables by observing the other. Let $\hat{y}$ be a representation of x, such that the distribution of $\hat{y}$ is fully described by the conditional $p(\hat{y}|x)$. This representation $\hat{y}$ is sufficient for y if $I(\hat{y};y)=I(x;y)$, in other words if $\hat{y}$ contains all the relevant information x had about y. It is minimal if $I(x;\hat{y})$ is smallest among the sufficient representations, in other words if the complexity of the representation is the lowest. The trade-off between the sufficiency and optimality is formulated as the minimization of the information bottleneck Lagrangian,

$$L(p(\hat{y}|x))=I(\hat{y};x)-\beta I(\hat{y};y),$$

where β operates as the trade-off parameter between the complexity (first term) and the sufficiency (second term). For an overparametrized network to fit a complex dataset (e.g. memorize random noise [36]) it has to pay a price in terms of the information complexity. Bounding the information complexity can thus prevent the overfitting, but the trade-off between the two may depend on the particular dataset used. In the case of neural networks, the representation $\hat{y}$ is governed by the learned weights w, which can be viewed as a random variable depending on the data and the optimization. The mutual information of the weights and the data can be denoted by $I(w;(x,y))$. The flat minima, i.e. the ones with small eigenvalues of the weight Hessian, can be interpreted as having low information. In other words, since the minimum is flat, the weights can be stored at lower precision, requiring fewer bits and having a lower information value. This result is derived more precisely in Proposition 4.3 of [1]; here we state their main result. Let w^* denote a local minimum of the cross-entropy loss and H^w is the Hessian at that point. For the optimal choice of the posterior $p(w|x,y)=\epsilon\cdot w^*$, the following bound can be obtained,

$$I(w;(x,y))\leq\frac{1}{2}d\left[\log||w^*||_2^2+\log||H^w||_*-d\log\left(\frac{d^2\beta}{2}\right)\right],$$

where $d=dim(w)$ and $||\cdot||_*$ denotes the nuclear norm. The nuclear norm is given by $||A||_*=Tr(\sqrt{A^*A})=Tr(A)$ for square, real matrices. The trace norm of the Hessian is equivalent to the L_1-norm of the vector of eigenvalues of the Hessian; therefore minimizing the nuclear norm is the same as reducing the rank of the original matrix (fewer non-zero eigenvalues). This thus states that flat minima have low information. The authors in [1] furthermore derive that when decreasing the information in the weights (by some form of regularization), one automatically improves the minimality and thus the invariance of the function. The converse, i.e. low information implies flatness, does not need to hold; in other words, as mentioned in Section 3.1.4, there exist minima with good generalization that are not flat.

4.2. Dependence on noise

Consider a time series $y^i, i = 0, \ldots, N$ with a signal to noise ratio of $(1-\alpha):\alpha$. Suppose the neural network output should be resistant to noise in the signal.

Remark 3 *(Robustness in non-i.i.d. setting).* We remark here that in the non-i.i.d. setting, the network can overfit to not just the noise, but also to certain patterns present in one part of the series but not in another. In this paper we mostly focus on robustness to noise, and assume that the non-i.i.d. property comes from the time dependence and the noise which we assume can change in distribution between the train and the test set. Generalization then refers to finding a pattern which is present over time, and the ability to generalize across different noise distributions. A similar setting has been considered for image recognition problems in, e.g. [15].

In this case the loss function should satisfy the following objective,

$$|\mathcal{L}(x+\alpha\epsilon, w, y) - \mathcal{L}(x, w, y)| < \delta.$$

Relating this to the train and test set, we assume that $x+\alpha\epsilon$ is the test data with a noise component different from that in the train data x. In this setting, a small δ corresponds to a small difference between the error on the train data and the error on the test data, or, in other words, a small generalization error. By a Taylor expansion in the input one obtains,

$$\mathcal{L}(x+\alpha\epsilon, w, y) - \mathcal{L}(x, w, y) \approx \alpha\epsilon^T \nabla_x \mathcal{L}(x, w, y) + \frac{1}{2}\alpha^2 \epsilon^T H^x(\mathcal{L}(x, w, y))\epsilon.$$

Then, taking expected values we have,

$$\mathbb{E}[\mathcal{L}(x+\alpha\epsilon, w, y) - \mathcal{L}(x, w, y)] \approx \frac{1}{2}\alpha^2 Tr\left(H^x(\mathcal{L}(x, w, y))\right),$$

where we have used the fact that $\epsilon_i \sim \mathcal{N}(0, 1)$ i.i.d.. From this it follows that,

$$Tr\left(H^x(\mathcal{L}(x, w, y))\right) = 2\frac{\delta}{\alpha^2}.$$

In other words, the amount of noise in the input the neural network has to be resistant to is inversely proportional to the input (and thus weight) Hessian. This is intuitive in classification problems. Consider an image or a time series one would like to classify. If the output function has to be resistant to $\alpha\epsilon$ noise, i.e. the classification output is invariant to $\alpha\epsilon$ noise in the input, we require the Hessian to be small, so that the shifted input still results in the same output. The higher the noise resistance should be, the smaller the Hessian. The downside is that the requirement for the learned function to possess more resistance against noise can also decrease the performance of the classifier.

4.3. Obtaining better generalizable minima

In this section we summarize which hyperparameters can be used to control the trade-off between generalization and complexity (typically the train data fit). These hyperparameters are similar to what is used in an i.i.d. setting, however using the derivations in the previous sections and as we will show in the numerical section in 5 these hyperparameters can also control the output in the non-i.i.d. setting.

Learning rate. In Section 3.1.2 the relationship between the weight Hessian and the learning rate was discussed. It was shown that using a higher learning rate can result in wider minima if convergence is obtained. In Section 3.2.2 the same kind of relationship is derived between the learning rate and the input Hessian. In [29] the authors also make a link between a high learning rate and a wide minimum. In particular, they claim that using a high learning rate allows the training algorithm to escape from sharp minima in the weight space so that the optimization algorithm converges to smoother and wider minima that are able to generalize better to unseen data. By starting with a small learning rate the weights do not diverge in the beginning when the gradients tend to be large, but due to the increase in learning rate the weights do not converge to a sharp local minimum either. A similar learning rate schedule was proposed in [31], where the authors used a cyclic learning rate with one cycle and a large maximum learning rate. Our derivations in the previous sections thus give a theoretical explanation as to why the learning rate can be used as a control parameter for generalization. In Section 5 we study this numerically. In order to avoid the size of the gradient influencing the minima to which the optimization converges (as does happen in the previous works of [29]), we propose to normalize the gradient by its L_2 norm.

Batch size. The batch size used, similar to the learning rate, determines the size of the noise of the SGD, as discussed in Sections 3.1.2 and 3.2.2. A smaller batch size results in a larger variance of the noise, which in turn, according to (9) results in the smoothing terms, i.e. the input Jacobian and Hessian, having a larger weight in the optimization objective. This makes the trade-off between data fit and function complexity more biased towards obtaining a low function complexity. The batch size can thus determine the amount of smoothness required in the output function learned by the neural network.

Number of iterations. A small learning rate results in smaller steps taken in the network. In other words, for the same number of training iterations a smaller learning rate may give rise to a minimum with a higher loss value. By a similar argument, a smaller number of training iterations gives rise to a minimum with higher loss. By Theorem 2, at least in theory, the higher-loss minima should also have a higher entropy and thus a better generalization. Therefore, early stopping, or equivalently training with fewer iterations, terminates the algorithm at a point in the loss surface with higher entropy. Training the network for fewer iterations should avoid overfitting on the noise, and by controlling the train error and the Hessian one can stop training when the sufficient trade-off between fit and smoothness has been obtained. In this way, the number of training iterations can be used to control the smoothness of the obtained solution, i.e. the trade-off in terms of data fit and the information complexity of the learned solution.

5. Numerical results

The neural network we consider, contains N_{depth} hidden layers with N_{width} nodes per layer. The weights are initialized as $\mathcal{N}\left(0, \frac{1}{N_{width}}\right)$ and trained with SGD. The learning rate is set to λ, with mini-batches of size N_b, and N_{it} iterations are used to minimize the mean squared error (MSE). The network is trained to predict the value at time $t+1$ given the time series at times $t-n, t-n+1, \ldots, t$, i.e. n historical datapoints. We use the hyperbolic tangent as the activation function. The results are presented for 20 trained networks starting from different initializations. The typical measures of generalization based on the input and weight Hessians are particular norms of the matrices. The trace of the weight Hessian will be used, i.e.

$$Tr(H^w) = \sum_{i=1}^{d} h_{ii}^w.$$

Similarly, the trace of the input Hessian averaged over the data samples will be used as a metric for generalization,

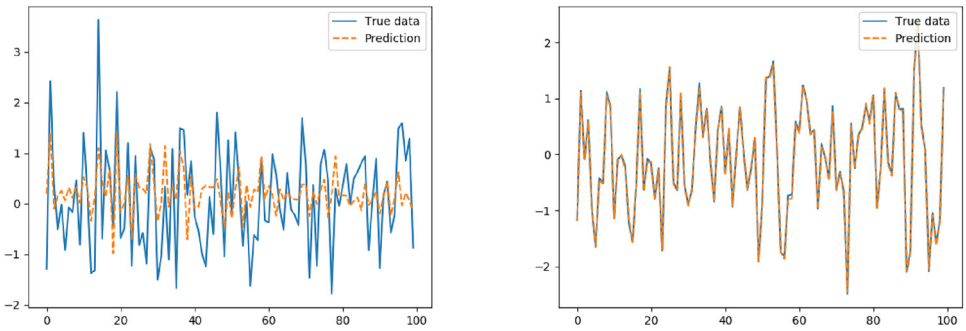

Fig. 1. The initial fit (L) and the output function (on the train data) learned (R) by a neural network of size $N_{depth} = 1$ and $N_{width} = 500$ with $\lambda = 0.05$ on random noise data. An overparametrized network trained with SGD can fit to random noise. This effect is undesired and ideally, a network will not converge on noise.

$$\mathbb{E}_x \left[Tr(H^x) \right] \approx \frac{1}{N} \sum_{i=1}^{N} Tr \left(H^{x^i} \right).$$

While in case of the Hessians we work with the trace of the matrices as a measure of generalization, for the input Jacobian we use the Frobenius norm and use as a metric of generalization the sensitivity of the output with respect to the input averaged over the data samples,

$$\mathbb{E}_x \left[||J^x||_F \right] \approx \frac{1}{N} \sum_{i=1}^{N} ||J^{x^i}||_F.$$

5.1. Artificial data

In this section we use artificial datasets to gain understanding of the neural network and its generalization capabilities. We show that as expected from the theory, a linear relation exists between the trace of the input and weight Hessians (i.e. the function complexity) and the generalization error. In our results, a higher function complexity means that the network is learning a function which is overfitting on the noise. This is clearly undesirable and results in a worse test set performance. The main task of this section is to show how to recognize when the network starts overfitting, and how to avoid this using the optimization hyperparameters as described in Section 4.3.

5.1.1. Random noise

We simulate 100 datapoints from an $\mathcal{N}(0, 1)$ distribution. An overparametrized neural network with the number of parameters larger than the sample size will be able to perfectly fit this random noise, see Fig. 1. However, in order to obtain a low loss, the network will have to significantly increase its function complexity, as can be measured by the norms of the weight and input Jacobians and Hessians. This is shown in Figs. 2 and 3 : the traces of the input and weight Hessians significantly increase to obtain a small loss value. After some number of iterations, if the MSE has converged to some value, the Hessian remains approximately constant, with small fluctuations due to the stochasticity of the optimization algorithm. When the network starts to learn the high-frequency components, here the noise, the norms of the input and weight Hessians increase significantly. In order to thus avoid convergence on noise one has to keep these norms small.

5.1.2. Noisy sine function

Consider now the function $y_i = \sin(0.1t_i) + c\epsilon_i$, with $t_i \in \{0, 1, \ldots, 100\}$ and $\epsilon_i \sim \mathcal{N}(0, 1)$. The network input consists of $(y_{i-4}, \ldots, y_i)$ and is trained to forecast y_{i+1}. Note that this time series is clearly not stationary, but contains seasonality which is a common feature of time series such as weather measurements.

Generalization and the number of iterations. Figs. 4 and 5 shows the trace norm of the input and weight Hessians plotted against the train error and the generalization error, respectively. There exists a linear relation between the trace of the input and weight Hessians and the generalization error: a smaller trace norm results in lower generalization error. This effect is slightly less significant for the deeper network. Furthermore, training longer results in a solution of higher complexity, which in this case is undesired since a higher complexity means that the function is overfitting on the noise. This is in accordance with the theoretical result on the entropy and the loss in Theorem 2, where it was claimed that the lower the train loss, the sharper the minimum, or the larger the output function complexity is. Training longer allows to access lower points in the loss surface with a lower train error as seen in Fig. 4. These lower points have a lower entropy, which results in a higher generalization error as seen from Fig. 5.

Increasing the noise amplitude. Consider now the sine function with the noise coefficient given by $c = 0.3$. In Fig. 6 we observe that in order to obtain a generalization error in the high noise case ($c = 0.3$) similar to that in the low noise case ($c = 0.1$, Fig. 5) the Hessian should be much smaller. This corresponds with the theoretical analysis in Section 4.2, where it was observed that with more noise in the signal one requires a lower Hessian in order to obtain a similar generalization error. Finding a low complexity solution on noisy data can be difficult due to the possibility of overfitting, since no explicit smoothness constraints are imposed. Deep networks are even more prone to overfitting due to the higher number of parameters and the sharper gradient descent directions. Thus, in order to obtain sufficiently smooth solutions for deep networks one needs to adapt the training method or cost function accordingly. For the training method, as seen in Fig. 6 taking fewer steps – or equivalently (not shown in the plots but discussed in Section 4.3) using smaller learning rates– results in smaller generalization error.

Generalization and the learning rate. Here we study the effects of the learning rate on generalization. Fig. 7 shows the test error plotted against the input and weight Hessians obtained by training the neural network with different learning rates. Using a larger learning rate results in wider minima while a smaller learning rate tends to converge to sharper minima, however a significant amount of outliers are found in both cases. We used batch gradient descent and scaled the gradient by its L_2 norm – i.e. the gradient in the SGD updates is given by $g/||g||_2$ – in order to avoid the gradient size influencing the minima width. We remark that the relation between a larger learning rate and wider minima seems to be more clear in the deep neural network where the usage of the higher learning rates results in more minima clustered at lower values of the trace. While the Hessian is correlated with the test error, i.e. a small test error means a smaller Hessian, the test set performance is not significantly better using the higher learning rates.

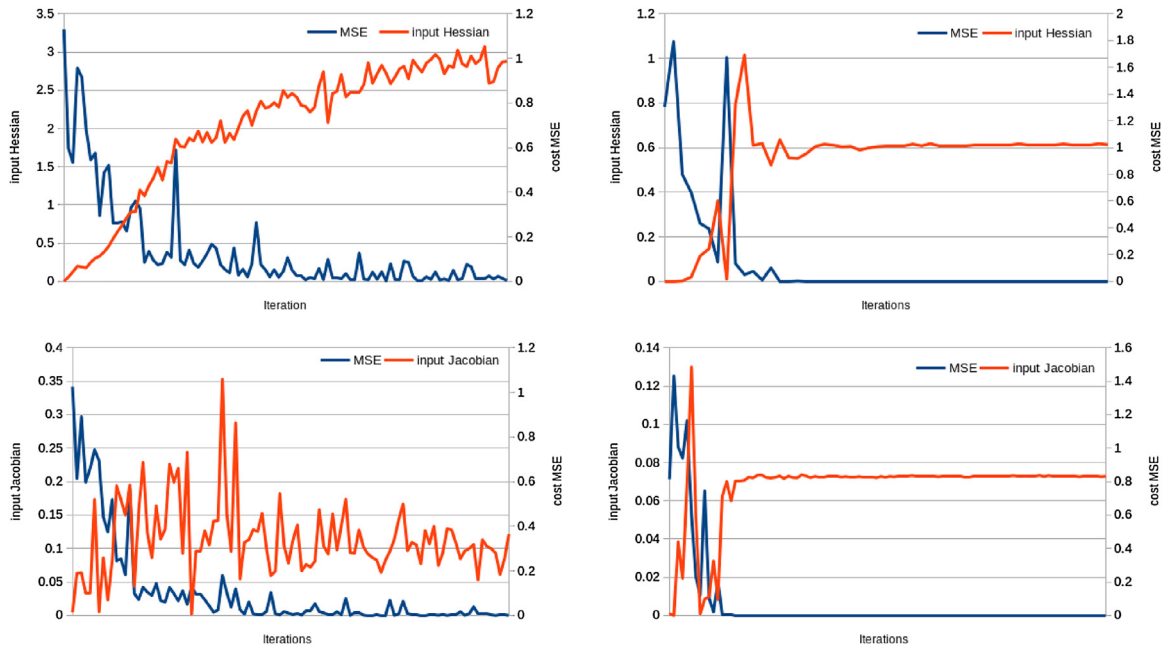

Fig. 2. The convergence of the network of size $N_{depth} = 1$ (L) and $N_{depth} = 10$ (R) and $N_{width} = 500$ with $\lambda = 0.05$ for the input Hessian (T) and the input Jacobian (B) on random noise data; the loss decreases with iterations, but the input norms of the Jacobian and Hessian increase, as the output function increases in complexity. The input Jacobian and Hessian can give indication for when a network starts overfitting on the noise. This can then be avoided by making a trade-off between complexity and train error.

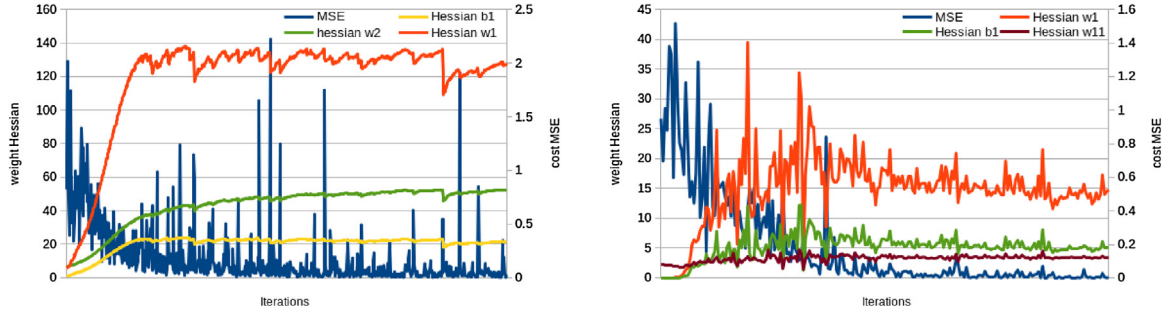

Fig. 3. The convergence of the network of size $N_{depth} = 1$ (L) and $N_{depth} = 10$ (R) and $N_{width} = 500$ with $\lambda = 0.05$ in terms of the loss function and the trace of the weight Hessians per layer on random noise data; the loss decreases with iterations, but the weight Hessians increase, showing that in order to obtain low loss the function complexity has to significantly increase. The weight Hessian norm can be used in order to bound the function complexity in order to avoid convergence on noise.

Even though convergence has been obtained, the network weights have converged to a minimum that underfits the data and therefore the output function can have a worse test set performance.

Generalization and the batch size. In Fig. 8 we plot the generalization error and the traces of the input and weight Hessians for different batch sizes. Using a smaller batch size causes the network to converge to minima with lower input and weight Hessians, which in turn correspond to minima with lower generalization error. As expected from the theory in Section 4.3 batch size has a significant influence on the smoothness of the output function with respect to input and weights, and can be used as a control for the trade-off between train and generalization error. Out of the three controls considered: number of iterations, learning rate and batch size, the number of iterations and the batch size appear to be the most effective ones for controlling the trade-off.

The scaled Hessian as a metric for generalization. Here we use the Hessian multiplied by the weights as defined in (8) as a measure for generalization. While good results were obtained with the original weight Hessian, it is of interest to see if the amount of outliers (here minima with different Hessians but similar generalization errors) decreases when using the scaled weight Hessian. In Fig. 9 we see that similar to the unscaled weight Hessian a clear linear dependence exists between the size of the Hessian and the generalization error. This dependence does not seem to be more significant as compared to the unscaled Hessian, showing that as expected from Section 3.1.3, the scaling sensitivity of the weight Hessian is not a significant problem for the minima found by SGD, and the weight Hessian is a valid metric for measuring generalization capability, despite its scaling sensitivity.

5.2. Real-world data

In this section we study generalizability for several real-world time series forecasting. We show that the norm of the input and weight Hessian is a good metric for measuring the capability of a network to generalize, and that similar to the artificial dataset, the hyperparameters defined in Section 4.3 are very effective for controlling the trade-off between smoothness – as measured by the Hessians – and data fit – as measured by the train loss.

Remark 4 *(Network architecture).* In the coming examples we consider a network architecture with $N_{depth} = 2$ and $N_{width} = 100$. Similar results, with regard to the effects of the controls and the ability of the metrics to measure generalization, hold for other architectures as long as the network is overparametrized. The hyperparameters would have to be tuned accordingly to the network size. For exam-

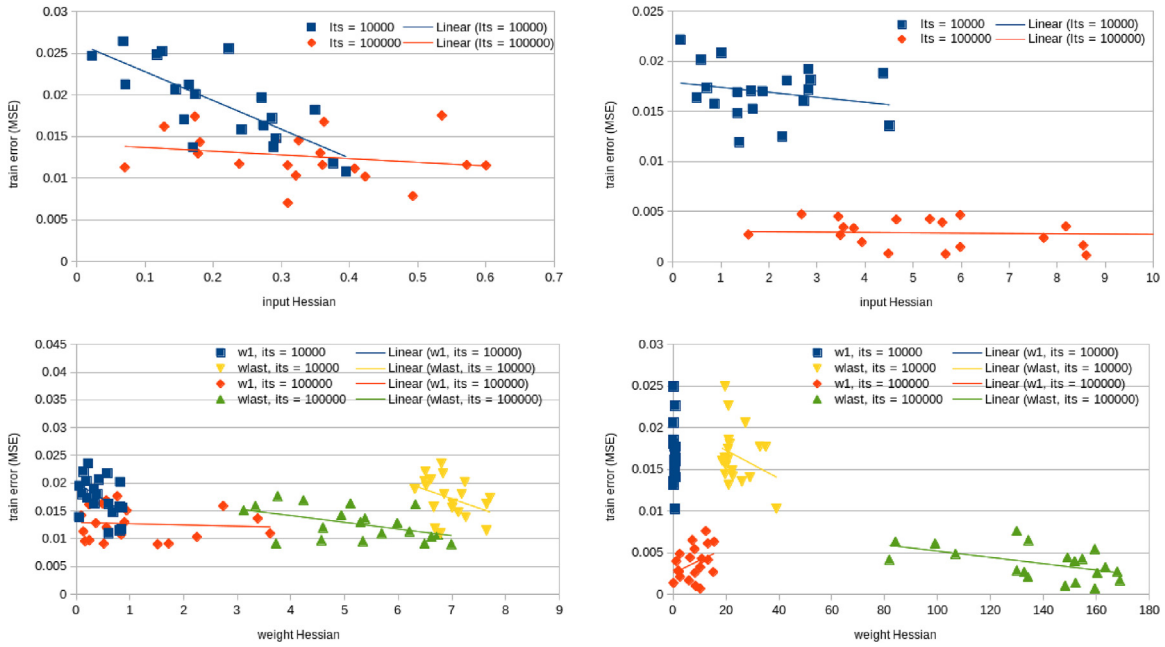

Fig. 4. The train error and trace norm of the input Hessian (T) and the weight Hessian (B) for the noisy sine with $c = 0.1$ for 20 trained networks. The neural network has 1 (L) and 10 (R) layers with 500 nodes per layer and is trained for a different number of iterations (10,000 and 100,000). Training longer results in a solution of higher complexity as measured by the norms of the input and weight Hessians. This solution has a smaller train error but will likely result in worse generalization.

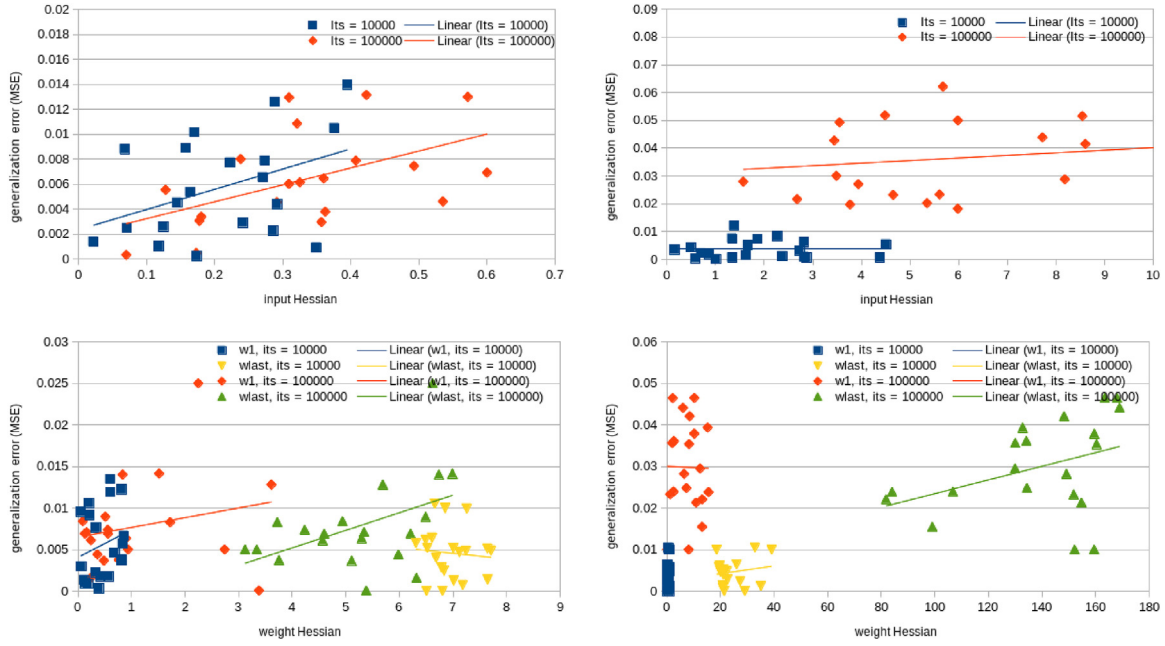

Fig. 5. The generalization error and trace norm of the input Hessian (T) and the weight Hessian (B) for the noisy sine with $c = 0.1$ for 20 trained networks. The neural network has 1 (L) and 10 (R) layers with 500 nodes per layer and is trained for a different number of iterations (10,000 and 100,000). A smaller trace of the Hessian results in a lower generalization error, and training longer increases the complexity of the learned solution.

ple, the more parameters a network has, the easier it is to overfit so that even fewer iterations might be needed to avoid the overfitting. Furthermore, as has been mentioned in, e.g. [26] the wider the neural network, the more noise it can handle, which in turn results in better generalization.

5.2.1. Index data

Financial data is highly non-linear, non-stationary and has a very low signal-to-noise ratio [10]. Overfitting on the training data and not being able to generalize well to unseen data is therefore a challenge. We will use a network of size $N_{depth} = 2$, $N_{width} = 100$. The input data will consist of $n = 5$ historical daily absolute returns of the S&P500 index, $n = 5$ historical daily absolute returns of the CBOE 10 year interest rate, and $n = 5$ historical daily absolute returns of the volatility index (VIX), so that the total input into the neural network will consist of 15 nodes. The train period consists of data from 2017-01-03 until 2018-02-02, and the test data from 2018-02-03 until 2018-08-13. Given the value of the time series S_t the returns are computed as $r_t = S_t - S_{t-1}$. The returns are then normalized using the mean and variance. The network output will consist of the prediction for the next day return r_{t+1} of the S&P500 index. In Table 1 we present the MSE and hit rate (computed as the number of up or

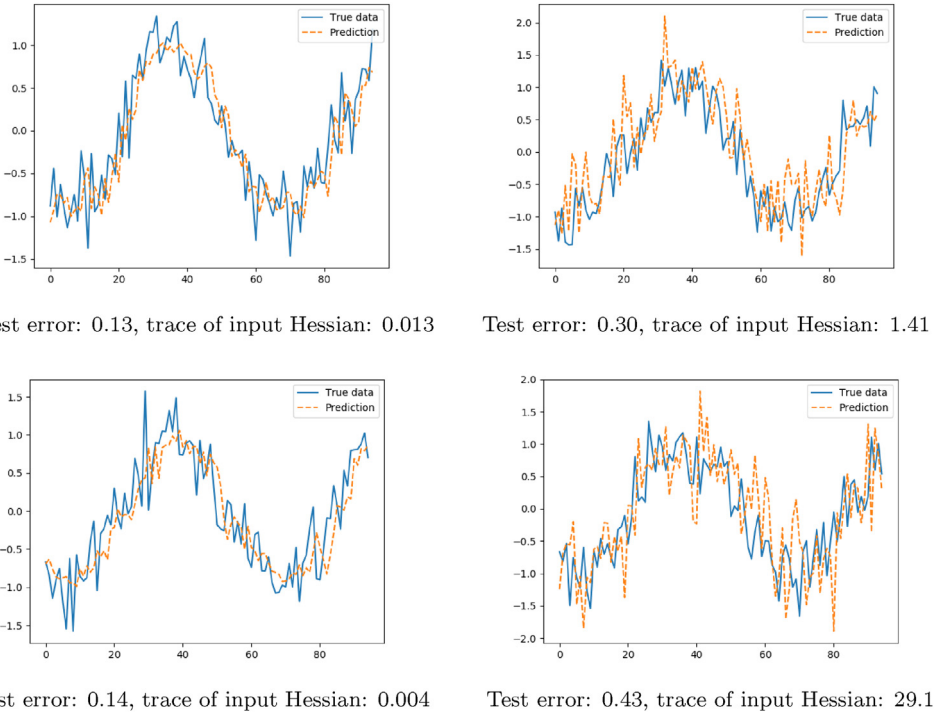

Fig. 6. The functions learned by the network on the sine data with $c = 0.3$ with 1000 (L) and 10,000 (R) iterations with a network of one layer (T) and 10 layers (B). With more noise the network is prone to overfitting, especially in the deep network. A smaller number of training iterations results in a lower Hessian which clearly results in a smoother function with lower test error.

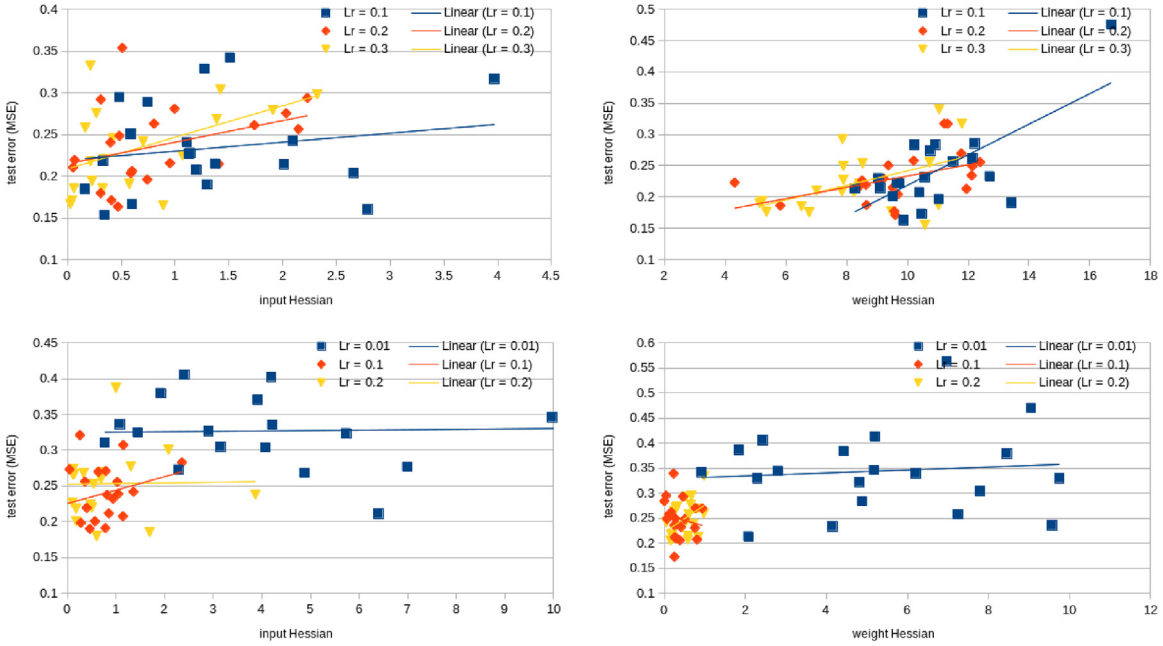

Fig. 7. The test error and the input (L) and weight (R) Hessians for the noisy sine function with $c = 0.3$ for different learning rates. The neural network has 1 (T) and 10 (B) layers with 100 nodes per layer. On average, training with a larger learning rate results in wider minima, i.e. smaller input and weight Hessians, however significant overlap between the minima found with different learning rates nevertheless exists.

down movements predicted correctly) for different hyperparameters averaged over 20 sampled networks. A larger trace of the input or weight Hessian appears to correspond to a worse performance; similarly, training longer results in overfitting. A smaller batch size corresponds to a smaller weight Hessian in the final layer, but it does not seem to result in better performance due to, e.g. underfitting the signal. Financial returns are highly noisy and non-linear and distinguishing the signal in the data from noise remains challenging. Nevertheless we showed that the techniques presented in the paper can be used to bias the algorithm into minima that have more (additive) noise resistance.

5.2.2. Temperature

In this section we train a network for predicting the daily minimum temperature in Melbourne, Australia. The dataset contains observations over the period of 1988-01-01 until 1990-12-31. We

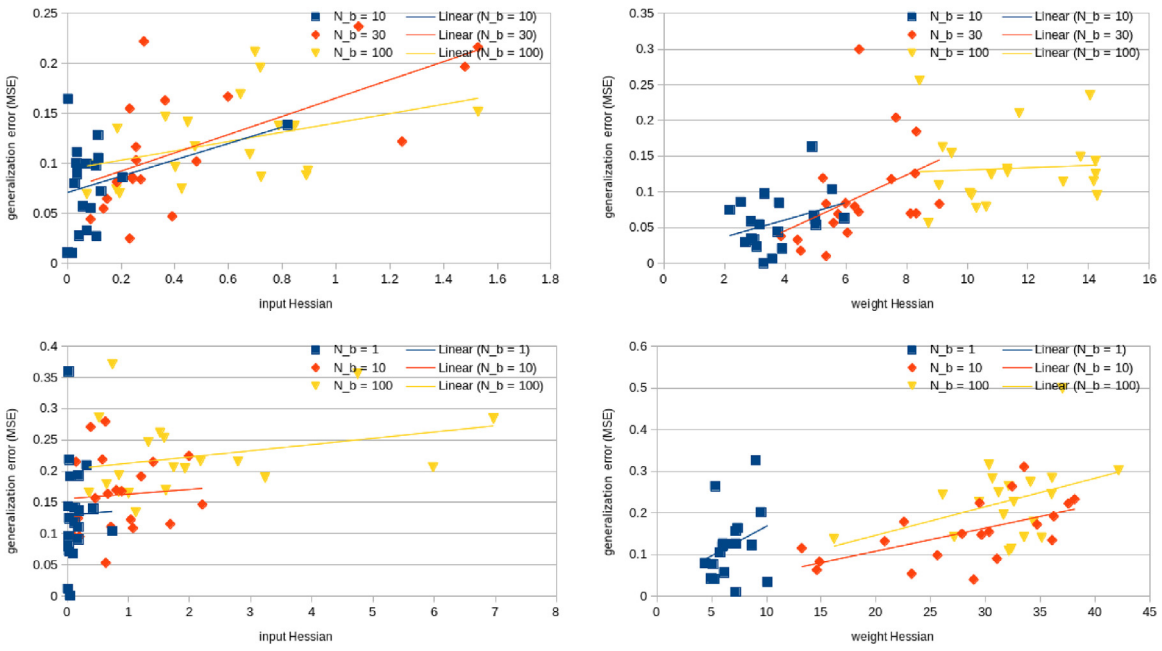

Fig. 8. The generalization error and the trace of the input (L) and weight (R) Hessians for the noisy sine function with $c = 0.3$ for different batch sizes. The neural network has 1 (T) and 10 (B) layers with 100 nodes per layer. Training with a smaller batch size results in a minimum with smaller values of its input and weight Hessian, which means a smoother output function and a wider minimum in weight space. Smaller batch sizes thus result in functions which can generalize better.

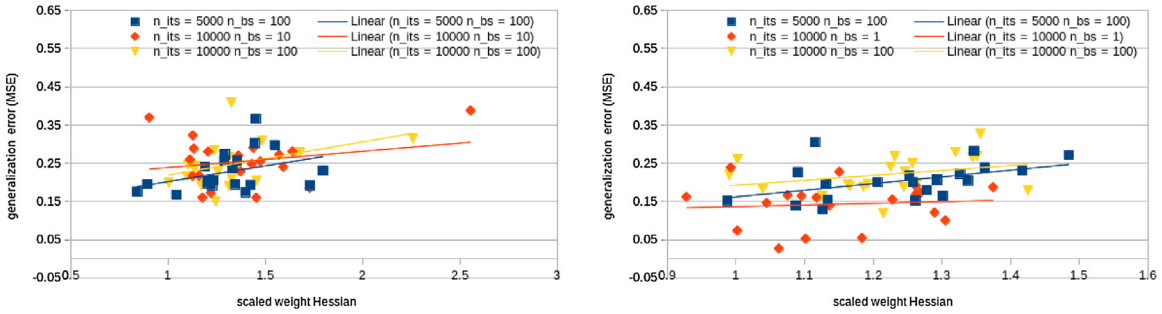

Fig. 9. The generalization error with respect to the weight Hessian multiplied by the weight vector as defined in (8) for a network with 1 (L) and 10 (R) layers for the noisy sine function with $c = 0.3$. A linear trend is observed with a smaller Hessian giving a lower generalization error. The scaled Hessian as a measure for generalization seems to be as accurate as the unscaled Hessian, showing that the scaling sensitivity is not a significant problem for the minima found with SGD.

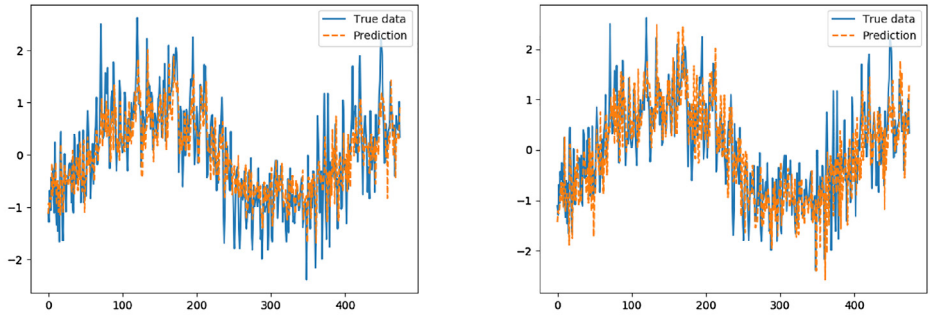

Fig. 10. The temperature forecast for a network trained for 5000 iterations (L) and 10,000 iterations (R). There is a clear seasonality in the dataset, however the network that was trained longer overfits on the noise in the observations and therefore generalizes worse.

will use a network of size $N_{depth} = 2$, $N_{width} = 100$. The input data will consist of $n = 20$ historical daily observations of the temperature. The results for the MSE for different training hyperparameters are presented in Table 2 and Fig. 10. As expected, training with fewer iterations and using smaller batch sizes results in a smoother output function with respect to the input and causes the training algorithm to converge to wider minima. The temperature data has a clear seasonal pattern but the daily observations vary due to noise; using smaller batch sizes or training shorter has a regularising effect on the output function, so that the network does not

Table 1
The MSE and hit rate for different batch sizes and iteration numbers. A higher trace of the input or weight Hessian results in a lower hit rate and a higher MSE. Training longer results in a higher error and using a smaller batch size results in a smaller weight Hessian in the final layer, but does not seem to correspond to a better performance due to, e.g. underfitting.

N_{it}	N_b	MSE	Hit rate	$Tr(H^x)$	$Tr(H^{W^{(1)}})$	$Tr(H^{W^{(3)}})$
10,000	100	2.80	0.49	0.73	5.60	45.16
5000	100	2.65	0.513	0.73	4.86	45.48
10,000	300	2.76	0.500	0.83	4.81	47.26
5000	300	2.71	0.505	0.74	4.50	46.60

Table 2
The MSE for different batch sizes and iteration numbers. A higher trace of the input or weight Hessian corresponds to a worse test set MSE due to overfitting on the noise. As usual, training longer and using larger batch sizes results in more overfitting.

N_{it}	N_b	MSE	$Tr(H^x)$	$Tr(H^{W^{(1)}})$	$Tr(H^{W^{(3)}})$
10,000	10	0.44	0.078	6.05	34.9
5000	10	0.36	0.023	5.85	27.5
10,000	100	0.49	0.11	36.7	40.2
5000	100	0.36	0.030	38.4	31.9
10,000	200	0.50	0.10	42.6	40.9
5000	200	0.37	0.032	56.7	31.7

overfit on the noise in the data, but continues to follow the main trend.

6. Conclusion

In this work we studied generalization capabilities of neural networks trained for the purpose of time series forecasting. We showed that there is a correspondence between good generalization capability and small input and weight Hessians of the loss function at the minima found after training. A small input or weight Hessian corresponds to the smoothness of the trained function, or, in other words, the resistance of the output function to noise in the input or weights, respectively. The challenge lies in finding the optimal tradeoff between fit of the data and smoothness of the learned function, so as to avoid overfitting on the noise and underfitting on the signal of interest. We showed how to use the learning rate, the batch size and the number of iterations used in the training algorithm to bias the network into minima that possess a certain structure. Other aspects that may influence generalization capabilities are the kind of activation function used: while not reported we noticed that the network is prone to overfitting when using the piecewise linear ReLU compared to the hyperbolic tangent or the sigmoid function. Furthermore, the network size itself also matters: deep networks, due to the larger amount of parameters, will more easily overfit on the noise, obtaining a low training error but a bad out-of-sample performance.

While this work provided some insight into obtaining good generalization for time series forecasting, forecasting remains a challenging task due to the non-linear and non-stationary distribution of the data. The typical assumption in statistical learning theory of having i.i.d. samples from some data-generating distribution does not hold in time series: there is a dependence between the observations through time and the underlying distribution may change due to unobserved variables so that the train and test data might not be identically distributed. A relaxation that has become standard to deal with the independence assumption is to assume that the observations are drawn from a stationary mixing distribution (see, e.g. [2,24]). The authors of, e.g. [18–20] provide bounds that also hold for non-stationary time series. A related issue is that of generalization in neural networks across different noise distributions. As has been mentioned in the work of [15], neural networks have trouble generalizing when the noise distribution in the data they were trained on differs from the noise distribution in the test dataset. Understanding and solving this issue will prove valuable in time series forecasting, where the distribution of the noise in the observations could change over time. Obtaining theoretical results on, e.g. the link between the generalization error and the Hessian, and understanding how to make machine learning algorithms work in order to generalize in a non-i.i.d. setting is still a relevant and active topic of research which we aim to address in future work.

Conflict of interest

None declared.

References

[1] A. Achille, S. Soatto, Emergence of invariance and disentanglement in deep representations, J. Mach. Learn. Res. 19 (2018) 1–34.
[2] A. Agarwal, J.C. Duchi, The generalization ability of online algorithms for dependent data, IEEE Trans. Inform. Theory 59 (2013) 573–587.
[3] A. Auffinger, G.B. Arous, J. Černý, Random matrices and complexity of spin glasses, Commun. Pure Appl. Math. 66 (2013) 165–201.
[4] S. Becker, Y. Zhang, A.A. Lee, Geometry of Energy Landscapes and the Optimizability of Deep Neural Networks, 2018 arXiv:1808.00408.
[5] J.L. Bernier, J. Ortega, E. Ros, I. Rojas, A. Prieto, A quantitative study of fault tolerance, noise immunity, and generalization ability of MLPs, Neural Comput. 12 (2000) 2941–2964.
[6] C. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), 1st ed. 2006. corr. 2nd printing ed., Springer, New York, 2007.
[7] A.J. Bray, D.S. Dean, Statistics of critical points of Gaussian fields on large-dimensional spaces, Phys. Rev. Lett. 98 (2007) 150201.
[8] P. Chaudhari, S. Soatto, Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks, in: 2018 Information Theory and Applications Workshop (ITA), IEEE, 2018, pp. 1–10.
[9] A. Choromanska, M. Henaff, M. Mathieu, G.B. Arous, Y. LeCun, The Loss Surfaces of Multilayer Networks, 2015, pp. 192–204.
[10] R. Cont, Empirical Properties of Asset Returns: Stylized Facts and Statistical Issues, 2001.
[11] Y.N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, Advances in Neural Information Processing Systems (2014) 2933–2941.
[12] L. Dinh, R. Pascanu, S. Bengio, Y. Bengio, Sharp Minima Can Generalize for Deep Nets, 2017 arXiv:1703.04933.
[13] G.K. Dziugaite, D.M. Roy, Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters Than Training Data, 2017 arXiv:1703.11008.
[14] Y.V. Fyodorov, I. Williams, Replica symmetry breaking condition exposed by random matrix calculation of landscape complexity, J. Stat. Phys. 129 (2007) 1081–1116.
[15] R. Geirhos, C.R. Temme, J. Rauber, H.H. Schütt, M. Bethge, F.A. Wichmann, Generalisation in humans and deep neural networks, Advances in Neural Information Processing Systems (2018) 7549–7561.
[16] S. Hochreiter, J. Schmidhuber, Flat minima, Neural Comput. 9 (1997) 1–42.
[17] S. Jastrzkebski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, A. Storkey, Three Factors Influencing Minima in SGD, 2017 arXiv:1711.04623.
[18] V. Kuznetsov, M. Mohri, Generalization bounds for time series prediction with non-stationary processes, in: International Conference on Algorithmic Learning Theory, Springer, 2014, pp. 260–274.
[19] V. Kuznetsov, M. Mohri, Generalization bounds for non-stationary mixing processes, Mach. Learn. 106 (2017) 93–117.
[20] V. Kuznetsov, M. Mohri, Theory and Algorithms for Forecasting Time Series, 2018 arXiv:1803.05814.
[21] H. Li, Z. Xu, G. Taylor, T. Goldstein, Visualizing the Loss Landscape of Neural Nets, NIPS, 2018.
[22] S. Mandt, M.D. Hoffman, D.M. Blei, Stochastic Gradient Descent as Approximate Bayesian Inference, 2017 arXiv:1704.04289.
[23] J. Martens, Deep learning via Hessian-free optimization, ICML, vol. 27 (2010) 735–742.
[24] D.J. McDonald, C.R. Shalizi, M. Schervish, Nonparametric risk bounds for time-series forecasting, J. Mach. Learn. Res. 18 (2017) 1–40.
[25] R. Novak, Y. Bahri, D.A. Abolafia, J. Pennington, J. Sohl-Dickstein, Sensitivity and Generalization in Neural Networks: An Empirical Study, 2018 arXiv:1802.08760.
[26] D.S. Park, J. Sohl-Dickstein, Q.V. Le, S.L. Smith, The Effect of Network Width on Stochastic Gradient Descent and Generalization: An Empirical Study, 2019 arXiv:1905.03776.
[27] J. Pennington, Y. Bahri, Geometry of neural network loss surfaces via random matrix theory, International Conference on Machine Learning (2017) 2798–2806.

[28] R. Reed, S. Oh, R. Marks, Regularization using jittered training data, in: International Joint Conference on Neural Networks, 1992. IJCNN, vol. 3, IEEE, 1992, pp. 147–152.
[29] S. Seong, Y. Lee, Y. Kee, D. Han, J. Kim, Towards Flatter Loss Surface Via Nonmonotonic Learning Rate Scheduling, UAI, 2018.
[30] R. Shwartz-Ziv, N. Tishby, Opening the Black Box of Deep Neural Networks Via Information, 2017 arXiv:1703.00810.
[31] L.N. Smith, N. Topin, Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates, 2017 arXiv:1708.07120.
[32] S.L. Smith, Q.V. Le, A Bayesian Perspective on Generalization and Stochastic Gradient Descent, 2018.
[33] J. Sokolić, R. Giryes, G. Sapiro, M.R. Rodrigues, Robust large margin deep neural networks, IEEE Trans. Signal Process. 65 (2017) 4265–4280.
[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR, 2014, pp. 1929–1958.
[35] N. Tishby, N. Zaslavsky, Deep learning and the information bottleneck principle, in: 2015 IEEE Information Theory Workshop (ITW), IEEE, 2015, pp. 1–5.
[36] C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals, Understanding Deep Learning Requires Rethinking Generalization, 2016 arXiv:1611.03530.
[37] G. Zhang, B.E. Patuwo, M.Y. Hu, Forecasting with artificial neural networks: the state of the art, Int. J. Forecast. 14 (1998) 35–62.
[38] W. Zhou, V. Veitch, M. Austern, R.P. Adams, P. Orbanz, Compressibility and Generalization in Large-Scale Deep Learning, 2018 arXiv:1804.05862.

Anastasia Borovykh is a postdoctoral researcher at the CWI (Centrum Wiskunde & Informatica) in Amsterdam. Her research interests are in the domain of machine learning, applied probability, and partial differential equations with a current focus on the theoretical aspects of neural networks. She obtained her Ph.D. cum laude from the University of Bologna in Financial Mathematics as part of a Marie-Curie Industrial Doctorates and Horizon2020 project, a Master's degree in Quantitative Finance at the VU Amsterdam and a Bachelor's degree in Applied Mathematics from the Delft University of Technology.

Cornelis W. Oosterlee is a senior scientist and group leader at the CWI – National Research Center for Mathematics and Computer Science in Amsterdam. He is also a full professor at the Delft University of Technology in Delft, the Netherlands, where he teaches Computational Finance courses. His expertise and interests include numerical methods and computational finance, like Fourier pricing techniques, partial differential equations for derivative pricing, stochastic models for hybrid derivatives, numerical techniques in risk management and Monte Carlo simulation.

Sander M. Bohté is a senior researcher at the Netherlands Centrum Wiskunde & Informatica (CWI), a part-time full professor of Bio-inspired Deep learning at the Rijksuniversiteit Groningen and a part-time full professor of Cognitive Computational Neuroscience at the University of Amsterdam. He specialises in the development of computational models to understand information processing in neural networks.