A mathematical framework for improved weight initialization of neural networks using Lagrange multipliers

de Pater, Ingeborg; Mitici, Mihaela

DOI
10.1016/j.neunet.2023.07.035

Publication date
2023

Document Version
Final published version

Published in
Neural networks : the official journal of the International Neural Network Society

Neural Networks 166 (2023) 579–594

Contents lists available at ScienceDirect

Neural Networks

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

A mathematical framework for improved weight initialization of neural networks using Lagrange multipliers

Ingeborg de Pater [a,*], Mihaela Mitici [b]

[a] Faculty of Aerospace Engineering, Delft University of Technology, HS 2926 Delft, The Netherlands
[b] Faculty of Science, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

ARTICLE INFO

Article history:
Received 11 January 2023
Received in revised form 12 June 2023
Accepted 26 July 2023
Available online 3 August 2023

Keywords:
Weight initialization
Neural network training
Linear regression
Lagrange function
Remaining useful life

ABSTRACT

A good weight initialization is crucial to accelerate the convergence of the weights in a neural network. However, training a neural network is still time-consuming, despite recent advances in weight initialization approaches. In this paper, we propose a mathematical framework for the weight initialization in the last layer of a neural network. We first derive analytically a tight constraint on the weights that accelerates the convergence of the weights during the back-propagation algorithm. We then use linear regression and Lagrange multipliers to analytically derive the optimal initial weights and initial bias of the last layer, that minimize the initial training loss given the derived tight constraint. We also show that the restrictive assumption of traditional weight initialization algorithms that the expected value of the weights is zero is redundant for our approach. We first apply our proposed weight initialization approach to a Convolutional Neural Network that predicts the Remaining Useful Life of aircraft engines. The initial training and validation loss are relatively small, the weights do not get stuck in a local optimum, and the convergence of the weights is accelerated. We compare our approach with several benchmark strategies. Compared to the best performing state-of-the-art initialization strategy (Kaiming initialization), our approach needs 34% less epochs to reach the same validation loss. We also apply our approach to ResNets for the CIFAR-100 dataset, combined with transfer learning. Here, the initial accuracy is already at least 53%. This gives a faster weight convergence and a higher test accuracy than the benchmark strategies.

1. Introduction

Neural networks have become increasingly popular in the last few decades, with applications in a wide range of domains (Li, Liu, Yang, Peng, & Zhou, 2021), such as image classification (Pan et al., 2022), time series prediction (Martínez, Charte, Frías, & Martínez-Rodríguez, 2022), object detection (Zhao, Zheng, Xu, & Wu, 2019) and natural language processing (Roh, Park, Kim, Oh, & Lee, 2021). The weights of these neural networks are usually optimized using gradient descent, until the weights converge such that the loss is close to a minimum (Vasilev, 2019). In the last years, many improvements have been proposed to accelerate the convergence of the weights as, for instance, improved optimizers (Kingma & Ba, 2014; Xie, Pu, & Wang, 2023), good weight initialization strategies (Glorot & Bengio, 2010; He, Zhang, Ren, & Sun, 2015) and the use of specialized hardware such as GPUs. However, training an accurate neural network is still computationally intensive, consumes a large amount of energy and takes a long time. This is especially problematic for deep neural networks trained with large data sets. Methods that accelerate the convergence of the weights in a neural network are therefore still needed.

One way to speed up the training of a neural network is a good weight initialization method. Most papers on weight initialization focus on initializing the weights such that the convergence of the weights during the back-propagation algorithm is accelerated. Here, the goal is to mitigate the vanishing or exploding gradient problem, and to prevent that the weights get stuck in a local optimum (Narkhede, Bartakke, & Sutaone, 2022). This is also the aim of the most popular approaches for weights initialization, namely "Xavier" (Glorot & Bengio, 2010) and "Kaiming" initialization (He et al., 2015). In Glorot and Bengio (2010), it is shown that the variance of both the forward-propagated outputs and the backward-propagated gradients should be equal for all layers in the neural network after the weight initialization to prevent the vanishing/exploding gradient problem. From this requirement, the required variance of the weights is derived for each layer. In Glorot and Bengio (2010), this required variance is derived for a neural network with the sigmoid or tanh activation function, while in He et al. (2015), this required variance is derived for a neural network with the ReLU activation function. In both papers, the weights are then randomly chosen from a distribution, such

* Corresponding author.
E-mail address: i.i.depater@tudelft.nl (I. de Pater).

https://doi.org/10.1016/j.neunet.2023.07.035
0893-6080/

as uniform or normal, with a mean of zero and the required variance. These weight initialization strategies are very successful when training deep neural networks. In this paper, we therefore also consider the requirement that the variance of the outputs and gradients of each layer of the neural network are equal after the weight initialization.

Other studies on weight initialization use the characteristics of the training data instead. In Saxe, McClelland, and Ganguli (2013), the weights are initialized with the orthogonal projection of the input correlation matrix using the singular value decomposition. In Mishkin and Matas (2015), this method is extended by using layer sequential unit variance: Random weights are first orthonormalized, and then the variance of the output of each layer is normalized to one. In Adam, Karras, Magoulas, and Vrahatis (2014), the weights are sampled from an interval that is determined based on the characteristics of the data. In this paper, we also use the characteristics of the training data to initialize the weights.

Overall, most existing studies on weight initialization do not consider the initial loss (Glorot & Bengio, 2010; He et al., 2015; Mishkin & Matas, 2015; Saxe et al., 2013). Due to the random starting point of the weights, however, many updating steps and thus many epochs might be necessary to achieve convergence. In contrast, there are few studies that focus on finding initial weights that minimize the initial training loss, i.e., on weights close to an optimum, instead. For example, in Aguirre and Fuentes (2019) and Yam, Chow, and Leung (1997) linear regression is used to initialize the weights of the last layer, the authors of Chumachenko, Iosifidis, and Gabbouj (2022) use linear discriminant analysis to initialize the weights such that the difference between classes is maximized and in Yam and Chow (1995), the initial training loss is minimized using the singular value decomposition. However, these studies in turn do not consider the convergence of the weights when initializing the weights of the last layer. To address this, we propose a weight initialization strategy that combines both goals: we initialize the weights and bias of the last layer of a neural network such that (i) the weight convergence during the back-propagation algorithm is accelerated and (ii) given that the weight convergence is accelerated, we initialize the weights close to an optimum point by minimizing the initial training loss.

Our approach is inspired by techniques from the field of neural networks with random weights (NNRW), such as extreme learning machines. In NNRW, the weights are optimized without gradient descent or other iterative methods (Cao, Wang, Ming, & Gao, 2018). Instead, the weights of the first layers of the neural network are randomly chosen. The weights of the last layer are then optimized with the objective to minimize the loss function using, for instance, Ridge regression (Cao et al., 2018), the matrix inverse (Kim, 2021) or inequality constrained least-squares (Fernández-Navarro, Riccardi, & Carloni, 2014). Inspired by NNRW, we optimize the initial weights and the initial bias of the last layer of the neural network using a regression model.

However, in contrast with NNRWs, we further optimize the weights with the back-propagation algorithm. To also mitigate the vanishing/exploding gradient problem, we analytically derive a novel tight constraint on the weights of the last layer to ensure that the variance of the output/gradients of the last layer is equal to the variance of the output/gradients of the other layers (Glorot & Bengio, 2010). This novel tight constraint holds without assuming that the mean of the weights and bias is zero, as commonly done in most weight initialization strategies. Given this constraint, we next derive analytically the optimal initial weights and bias of the last layer, i.e., the weights and bias that give the lowest initial training loss, using Lagrange. We derive this constraint, and the corresponding optimal initial weights and bias, for neural networks that solve a regression problem and that use a specific activation function.

We first apply our proposed approach to predict the Remaining Useful Life (RUL) of aircraft engines using a Convolutional Neural Network, i.e., a regression problem. Compared to the random Xavier weight initialization method (Glorot & Bengio, 2010), the weights indeed start at point that gives a relatively small initial training and validation loss: The initial validation loss is 2.4 times smaller with our approach. Moreover, the weights quickly converge further from this small initial training loss, due to the novel constraint derived on the weights. Last, the weights do not get stuck in a local optimum with our approach. Overall, the training of the neural network is therefore much faster when using our approach: The smallest validation loss obtained after 198 epochs following Xavier (Glorot & Bengio, 2010), is already obtained after 49 epochs with our approach. We thus need 75% epochs less to reach the same result. The best benchmark strategy is Kaiming weight initialization (He et al., 2015). Here, the initial validation loss is 2.4 times smaller with our approach as well. Moreover, the minimum validation loss reached after 148 epochs using Kaiming initialization, is already reached after 97 epochs with our approach, i.e., we need 34% less epochs to reach the same validation loss. We also show that with the new initialization technique, we can relax the assumption that the mean of the weights is zero (as assumed in Glorot and Bengio (2010) and He et al. (2015)). Last, we find that only a small part of the training set can be used to initialize the weights. This makes the weight initialization 42 times faster, while the weights converge at the same rate.

We next adjust our approach so that it can be applied to neural networks that solve classification problems as well, with any type of activation function. We then illustrate our approach using ResNet-18 and ResNet-34 (He, Zhang, Ren, & Sun, 2016) to classify the images in the CIFAR-100 dataset (Krizhevsky et al., 2009). When training ResNet-18 and ResNet-34 from scratch, we achieve a slightly faster convergence of the weights. However, we obtain the best results when combining our approach with transfer learning: Combined with transfer learning, the initial accuracy of the validation set after applying our approach is already 53% and 55% for ResNet-18 and ResNet-34 respectively. This leads to a faster convergence of the weights: Using ResNet-18 and the best benchmark method (LeCun initialization (LeCun, Bottou, Orr, & Müller, 2012)), the highest validation accuracy of 81.32% is obtained after 49 epochs. Using our approach, the same accuracy is already obtained after just 13 epochs. Similarly, for ResNet-34, the highest validation accuracy of 84.18% is obtained after 50 epochs using LeCun initialization, while it is already obtained after 14 epochs with our approach.

The remainder of this paper is structured as follows. We first introduce our proposed weight initialization methodology for neural networks in Section 2. We apply this methodology to a case study with a regression problem in Section 3, and compare the performance of our method with other benchmark strategies in Section 4. We also analyze our weight initialization procedure when only a part (10%) of the training set is used in Section 4.3, and we add an extra assumption on the mean of the weights in Section 4.4. Last, we apply our approach to classification neural networks, namely ResNet-18 and ResNet-34, in Section 5. We then discuss the conclusions, the limitations of this research and future research directions in Section 6. The Python code for the proposed weight initialization procedure is included in supplementary appendix.

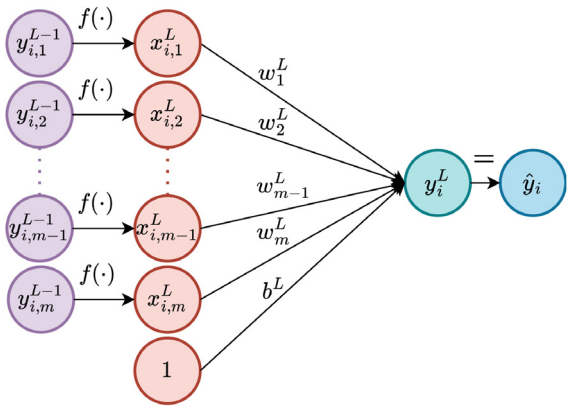


Fig. 1. Schematic overview of the last layer of the assumed neural network for a training sample $i \in S$.

2. Methodology – Weight initialization in the last layer of the neural network

In Section 2.1 we discuss the layout of the considered neural network. We derive the constraints on the weights in the last layer in Section 2.2, and we integrate these constraints in the linear regression problem using the Lagrange multiplier in Section 2.3. We summarize the full weight initialization procedure in Section 2.4. The Python code for the weight initialization procedure in Pytorch is included in supplementary appendix.

2.1. Neural network for a regression problem

We use a neural network with L layers to solve a regression problem, i.e., the true label of each sample is one numerical value. Let S be the training set for the neural network, with N training samples. The true label of a training sample $i \in S$ is denoted by y_i, and the vector with all true labels is denoted by $\mathbf{y} = [y_1, y_2, \ldots, y_N]$. We assume that the activation function $f(\cdot)$ used throughout the neural network has a unit derivative at 0, i.e., $f'(0) = 1$. For example, this could be the hyperbolic tangent activation function tanh. Moreover, we assume that the last layer L of this neural network is a fully connected layer. This is often the case for neural networks that solve a regression problem, e.g., Kim et al. (2020), Vural, Ilhan, Yilmaz, Ergüt, and Kozat (2021) and Yan (2012). A schematic overview of this last layer is in Fig. 1.

Let $\mathbf{y}^{L-1} = [\mathbf{y}_1^{L-1}, \mathbf{y}_2^{L-1}, \ldots, \mathbf{y}_m^{L-1}]$ be the matrix with the output of the $(L-1)$th layer, with m the number of output nodes. Here, $\mathbf{y}_j^{L-1} = [y_{1,j}^{L-1}, y_{2,j}^{L-1}, \ldots, y_{N,j}^{L-1}]^T$ is the vector with the output of the jth node of the $(L-1)$th layer for all training samples $i \in S$, and T denotes the transpose. Then, $\mathbf{x}^L = f(\mathbf{y}^{L-1}) = [\mathbf{x}_1^L, \mathbf{x}_2^L, \ldots, \mathbf{x}_m^L]$ is the matrix with the activated input of layer L. Here, $\mathbf{x}_j^L = [x_{1,j}^L, x_{2,j}^L, \ldots, x_{N,j}^L]^T$ is the vector with the jth hidden input state of layer L for all training samples $i \in S$. Last, the weights of layer L are denoted by $\mathbf{w}^L = [w_1^L, w_2^L, \ldots, w_m^L]$, while b^L denotes the bias. The output y_i^L of layer L for training sample i is then $y_i^L = \sum_{j=1}^m w_j^L x_{i,j}^L + b^L$.

We assume that the considered neural network applies a linear activation function to the output, i.e., the output y_i^L of the last layer directly is the estimated label $\hat{y}_i$ of training sample i. This linear activation function is also often applied in neural networks that solve a regression problem, e.g., Kim et al. (2020), Vural et al. (2021) and Yan (2012). The vector with estimated labels for the training set S is denoted by $\hat{\mathbf{y}} = \mathbf{y}^L = [y_1^L, y_2^L, \ldots, y_N^L]$. The objective of the regression task is to minimize the loss function. As loss function, we take the squared error:

$$\text{Loss} = \sum_{i \in S} \left(y_i - \hat{y}_i\right)^2. \tag{1}$$

In this paper, we randomly initialize the weights of the first $L-1$ layers from a normal distribution following Glorot and Bengio (2010) (i.e., Xavier initialization). Given these random weights, the aim of this study is to initialize the weights $\mathbf{w}^L$ and bias b^L of the last layer such that the loss is minimized. We therefore perform one forward pass with all training samples in S, and obtain the hidden input states $\mathbf{x}^L$ of layer L. Now, the weights $\mathbf{w}^L$ and bias b^L that minimize the loss function can easily be obtained with the least squares solution of a linear regression of the actual labels $\mathbf{y}$ on the hidden states $\mathbf{x}^L$. The objective of this linear regression is:

$$\begin{aligned} &\min_{b^L, \mathbf{w}^L} \sum_{i \in S} \left(y_i - \hat{y}_i\right)^2 \\ &= \min_{b^L, w_j^L, j=1,2,\ldots,m} \sum_{i \in S} \left(y_i - \left(\sum_{j=1}^m w_j^L x_{i,j}^L + b^L\right)\right)^2. \end{aligned} \tag{2}$$

However, initializing the weights in this way might cause the gradients to vanish or explode during the back-propagation algorithm. This in turn hinders the convergence of the weights. To avoid the vanishing or exploding gradient problem, the authors of Glorot and Bengio (2010) found that the variance of the outputs and gradients of each layer in the neural network should be equal after weight initialization. In the next sections, we rewrite this requirement in a constraint on the weights of the last layer, and integrate this constraint in the linear regression problem.

2.2. Constraints on the weights of the last layer of the neural network

To avoid the vanishing or the exploding gradient problem, the variance (Var) of (i) the forward-propagated output and (ii) the backward-propagated gradients of each layer should be equal. For the last layer L, this means that Glorot and Bengio (2010):

$$\text{Var}\left(\hat{y}\right) = \text{Var}\left(x^L\right) \tag{3}$$

$$\text{Var}\left(\frac{\partial \text{Loss}}{\partial y^L}\right) = \text{Var}\left(\frac{\partial \text{Loss}}{\partial y^{L-1}}\right), \tag{4}$$

where $\hat{y}$, x^L, $\frac{\partial \text{Loss}}{\partial y^L}$ and $\frac{\partial \text{Loss}}{\partial y^{L-1}}$ represent the random variable of any element in $\hat{\mathbf{y}}$, $\mathbf{x}^L$, $\frac{\partial \text{Loss}}{\partial \mathbf{y}^L}$ and $\frac{\partial \text{Loss}}{\partial \mathbf{y}^{L-1}}$ respectively (He et al., 2015). In Glorot and Bengio (2010), these two requirements are used to derive the desired variance for the weights in each layer. The authors assume in this derivation that the initial weights are independent and identically distributed random variables. The weights are then randomly sampled from a distribution, usually normal or uniform, with mean zero and the desired variance. This strategy is commonly called "Xavier initialization".

In this study, we cannot directly use the same derivation as in Glorot and Bengio (2010), since we do not assume that the mean of the weights is zero. Moreover, regarding the weights as independent and identically distributed random variables, as in Glorot and Bengio (2010), becomes problematic for the considered approach. We therefore use the two requirements to derive a constraint on the weights in the last layer using the rules of the variance of a linear function instead. An advantage of this method is that less assumptions are made, and that it suits a linear regression approach more naturally. To verify our approach, we show in Appendix C how the same constraints can be derived using the same derivation as in Glorot and Bengio (2010).

Requirement 1 : Var $(\hat{y}) = Var\,(x^L)$

As in Glorot and Bengio (2010) and He et al. (2015), we assume that the hidden states in $\mathbf{x}^L$ are independently and identically distributed. Specifically, the variance of x_j^L, representing a random variable of any element in $\mathbf{x}_j^L$, is equal for all nodes $j \in \{1, 2, \ldots, m\}$. This assumption from Glorot and Bengio (2010) still holds, since these hidden states come from the randomly initialized weights. In contrast with Glorot and Bengio (2010) and He et al. (2015), however, we treat the initialized weights and the initialized bias of the last layer as constant numbers instead of random variables, i.e., given the initialized bias and weights, we assume that the variance of $\hat{y}$ comes from the variance of x^L only. This interpretation fits a linear regression approach well, since the weights are not sampled from a distribution. Since we impose that $\hat{y} = y^L$, it follows that Heij et al. (2004):

$$\begin{aligned}\text{Var}(\hat{y}) &= \text{Var}\left(b^L + \sum_{j=1}^{m} w_j^L x_j^L\right) \\ &= \sum_{j=1}^{m} \left(w_j^L\right)^2 \text{Var}\left(x^L\right)\end{aligned} \tag{5}$$

The first requirement states that $\text{Var}\left(\hat{y}\right) = \text{Var}\left(x^L\right)$. This gives the following constraint on the sum of the squared weights:

$$\sum_{j=1}^{m} \left(w_j^L\right)^2 = 1 \tag{6}$$

In Glorot and Bengio (2010), it is assumed that the expected value of the weights is zero. If we would also assume that the mean of the weights is zero, then Eq. (6) states that the empirical variance of the weights should equal $\frac{1}{m}$ (i.e., $\frac{1}{m}\sum_{j=1}^{m}\left(w_j^L\right)^2 = \frac{1}{m}$). This is then the same as the constraint on the variance of the weights in Eq. (10) in Glorot and Bengio (2010). However, we do not use that the mean of the weights is zero in our derivation. In this study, we therefore first consider the case where the mean of the weight is not restricted. In Section 4.4, we instead follow (Glorot & Bengio, 2010; He et al., 2015) and assume that the mean of the weights is zero.

Requirement 2 : $Var\left(\frac{\partial Loss}{\partial y^L}\right) = Var\left(\frac{\partial Loss}{\partial y^{L-1}}\right)$

The second requirement states that the variance of the gradients is equal throughout the neural network. To derive a constraint on the weights from this requirement, we first write out $\frac{\partial \text{Loss}}{\partial \mathbf{y}_j^{L-1}}$ of one hidden state $j, j \in \{1, 2, \ldots, m\}$:

$$\frac{\partial \text{Loss}}{\partial \mathbf{y}_j^{L-1}} = \frac{\partial \text{Loss}}{\partial \mathbf{y}^L}\frac{\partial \mathbf{y}^L}{\partial \mathbf{x}^L}\frac{\partial \mathbf{x}^L}{\partial \mathbf{y}_j^{L-1}} = \frac{\partial \text{Loss}}{\partial \mathbf{y}^L} w_j^L f'\left(\mathbf{y}_j^{L-1}\right), \tag{7}$$

with $f'(\cdot)$ the derivative of the considered activation function. As in Glorot and Bengio (2010), we assume that this activation function has a unit derivative at 0 (see Section 2.1). Since the weight initialization in all layers before layer L is still random, we follow (Glorot & Bengio, 2010) and assume that $\mathbf{y}_j^{L-1} \approx \mathbf{0}$, and thus that $f'(\mathbf{y}_j^{L-1}) \approx \mathbf{1}$. This gives:

$$\frac{\partial \text{Loss}}{\partial \mathbf{y}_j^{L-1}} \approx \frac{\partial \text{Loss}}{\partial \mathbf{y}^L} w_j^L \tag{8}$$

Using this, we derive the variance:

$$\text{Var}\left(\frac{\partial \text{Loss}}{\partial y_j^{L-1}}\right) = \text{Var}\left(\frac{\partial \text{Loss}}{\partial y^L} w_j^L\right) = \left(w_j^L\right)^2 \text{Var}\left(\frac{\partial \text{Loss}}{\partial y^L}\right), \tag{9}$$

where $\frac{\partial \text{Loss}}{\partial y_j^{L-1}}$ represent the random variable of any element in $\frac{\partial \text{Loss}}{\partial \mathbf{y}_j^{L-1}}$. We are, however, interested in the variance of $\frac{\partial \text{Loss}}{\partial y^{L-1}}$ for the random variable of any element in $\frac{\partial \text{Loss}}{\partial \mathbf{y}^{L-1}}$, over all hidden states j. Since w_j^L usually does not equal w_i^L if $i \neq j$, we cannot assume that the variance of $\frac{\partial \text{Loss}}{\partial y_j^{L-1}}$ is the same for all $j \in \{1, 2, \ldots, m\}$. Instead, we use that the variance of $\frac{\partial \text{Loss}}{\partial y^{L-1}}$ is the variance of a mixture distribution of all random variables $\frac{\partial \text{Loss}}{\partial y_j^{L-1}}$, for $j \in \{1, 2, \ldots, m\}$. The variance of m mixture distributions, where each distribution has weight $\frac{1}{m}$, mean $\mu_j = \mathbb{E}[\frac{\partial \text{Loss}}{\partial y_j^{L-1}}]$, and variance $(w_j^L)^2 \text{Var}\left(\frac{\partial \text{Loss}}{\partial y^L}\right)$, and where the total mean is $\mu = \mathbb{E}[\frac{\partial \text{Loss}}{\partial y^{L-1}}]$, equals (Frühwirth-Schnatter & Frèuhwirth-Schnatter, 2006):

$$\sum_{j=1}^{m} \frac{1}{m}\left(\left(w_j^L\right)^2 \text{Var}\left(\frac{\partial \text{Loss}}{\partial y^L}\right) + \mu_j^2\right) - \mu^2$$

The expected value μ_j of $\frac{\partial \text{Loss}}{\partial y_j^{L-1}}$ is:

$$\begin{aligned}\mathbb{E}\left[\frac{\partial \text{Loss}}{\partial y_j^{L-1}}\right] &= \mathbb{E}\left[\frac{\partial \text{Loss}}{\partial y^L} w_j^L\right] \\ &= \mathbb{E}\left[-2(y - \hat{y}) w_j^L\right] \\ &= -2 w_j^L \left(\mathbb{E}[y] - \mathbb{E}[\hat{y}]\right).\end{aligned} \tag{10}$$

Here, y represent the random variable of any element in $\mathbf{y}$. In Section 2.3, we show that the expected value $\mathbb{E}[\hat{y}]$, given the optimal bias and weights in the last layer, equals $\frac{1}{N}\sum_{i \in S} y_i = \mathbb{E}[y]$. The expected value μ_j of $\frac{\partial \text{Loss}}{\partial y_j^{L-1}}$ is thus zero, and the total mean μ is zero as well. This gives:

$$\text{Var}\left(\frac{\partial \text{Loss}}{\partial y^{L-1}}\right) = \sum_{j=1}^{m} \frac{1}{m}(w_j^L)^2 \text{Var}\left(\frac{\partial \text{Loss}}{\partial y^L}\right) \tag{11}$$

We therefore derive the constraint that:

$$\frac{1}{m}\sum_{j=1}^{m}\left(w_j^L\right)^2 = 1. \tag{12}$$

If we would also assume that the mean of the weights is zero, then Eq. (12) states that the empirical variance of the weights should equal 1. This is then the same as the constraint on the variance of the weights in Eq. (11) of Glorot and Bengio (2010).

Final constraint. As in Glorot and Bengio (2010), we derive two different, conflicting constraints on the weights. As a compromise, we therefore average the two constraints (Glorot & Bengio, 2010):

$$\sum_{j=1}^{m}\left(w_j^L\right)^2 = \frac{1+m}{2} \tag{13}$$

2.3. Lagrange relaxation of the constrained linear regression problem

To initialize the weights in the last layer, we thus solve the following constrained linear regression problem:

$$\begin{aligned}\min_{b^L, w_j^L, j=1,2,\ldots,m} \quad & \sum_{i \in S}\left(y_i - b^L - \sum_{j=1}^{m} w_j^L x_{i,j}^L\right)^2 \\ \text{such that} \quad & \sum_{j=1}^{m}\left(w_j^L\right)^2 = \frac{1+m}{2}.\end{aligned} \tag{14}$$

This constrained regression problem can be solved exactly using the Lagrange multiplier. The Lagrange function $\mathcal{L}(\lambda, b^L, w_j^L, j = 1, 2, \ldots, m)$ is:

$$\mathcal{L}\left(\lambda, b^L, w_j^L, j = 1, 2, \ldots, m\right) = \sum_{i \in S} \left(y_i - b^L - \sum_{j=1}^{m} w_j^L x_{i,j}^L \right)^2 + \lambda \left(\sum_{j=1}^{m} \left(w_j^L\right)^2 - \frac{1+m}{2} \right). \tag{15}$$

This minimization problem is similar to the minimization problem in Ridge linear regression (Hastie, Tibshirani, Friedman, & Friedman, 2009). In Ridge regression, however, the value of the Lagrange multiplier λ is often chosen directly by the user instead.

The derivation of the solution of the Lagrange function in terms of λ, $\mathbf{w}^L$ and b^L is well-known (see Hastie et al., 2009). We therefore only give the final solution here. For completeness, we give the full derivation of this solution in Appendix A. In Appendix A, we first use the singular value decomposition of the centered hidden state to derive the optimal value for λ. Given this value for λ, the optimal value of the weights is (see Appendix A):

$$\mathbf{w}^L = \left(\left(\mathbf{x}^c\right)^T \mathbf{x}^c + \lambda \mathbf{I} \right)^{-1} \left(\mathbf{x}^c\right)^T \mathbf{y}^c, \tag{16}$$

with $\mathbf{y}^c$ a $N \times 1$ vector with the centered true label of all training samples in S, and $\mathbf{x}^c$ a $N \times m$ matrix with the centered hidden states for each training sample and each input node. Here, for one sample $i \in S$, we define the jth centered hidden state as:

$$x_{i,j}^c = x_{i,j}^L - \bar{x}_j^L, \tag{17}$$

with $\bar{x}_j^L$ the mean value of the jth hidden state over all training samples $i \in S$, i.e., $\bar{x}_j^L = \frac{1}{N} \sum_{i \in S} x_{i,j}^L$. The centered true label of a sample i is:

$$y_i^c = y_i - \frac{1}{N} \sum_{i \in S} y_i. \tag{18}$$

Last, given these weights, the optimal value of the bias is (see Appendix A):

$$b^L = \frac{1}{N} \sum_{i \in S} y_i - \sum_{j=1}^{m} w_j^L \bar{x}_j^L. \tag{19}$$

Note that with this value for b^L, the expected value of $\frac{\partial \text{Loss}}{\partial y_j^{L-1}}$ in Eq. (10) is zero since $\mathbb{E}[\hat{y}] = \mathbb{E}[y]$:

$$\begin{aligned} \mathbb{E}[\hat{y}] &= \mathbb{E}\left[b^L + \sum_{j=1}^{m} w_j^L x_j^L \right] \\ &= \mathbb{E}\left[\frac{1}{N} \sum_{i \in S} y_i - \sum_{j=1}^{m} w_j^L \bar{x}_j^L + \sum_{j=1}^{m} w_j^L x_j^L \right] \\ &= \frac{1}{N} \sum_{i \in S} y_i - \sum_{j=1}^{m} w_j^L \bar{x}_j^L + \sum_{j=1}^{m} w_j^L \mathbb{E}[x_j^L] \\ &= \frac{1}{N} \sum_{i \in S} y_i \\ &= \mathbb{E}[y], \end{aligned} \tag{20}$$

where we use that $\mathbb{E}[x_j^L] = \frac{1}{N} \sum_{i \in S} x_{i,j}^L = \bar{x}_j^L$.

2.4. Procedure for the weight initialization of a neural network

To initialize the weights of the neural network, we follow the steps below:

1. Randomly initialize the weights of the first $L-1$ layers using Xavier initialization: Sample the weights from an uniform or normal distribution with a mean of zero and the variance as derived in Glorot and Bengio (2010).
2. Perform one forward pass to compute the hidden states $x_{i,j}^L$ for all $j \in \{1, 2, \ldots, m\}$, $i \in S$.
3. Solve the constrained minimization problem in Eq. (14):
   (a) Center the hidden input states $x_{i,j}^L$ following Eq. (17) and center the true labels y_i following Eq. (18).
   (b) Calculate the optimal value of the Lagrange multiplier λ using the singular value decomposition of the centered input values $\mathbf{x}^c$ (see Appendix A).
   (c) Initialize the weights of the last layer as in Eq. (16).
   (d) Initialize the bias of the last layer as in Eq. (19).

The Python code for the weight initialization procedure with PyTorch is given in supplementary appendix.

2.5. Assuming the weights must have zero mean

In Xavier (Glorot & Bengio, 2010) and Kaiming (He et al., 2015) initialization, it is assumed that the expected value of the initialized weights is zero. Most studies therefore initialize the weights from a normal or uniform distribution with a mean of zero. To analyze the potential benefits of this restriction for our approach, we also impose here that the mean of the weights is zero. With this assumption, the final constraint on the sum of the squared weights in Eq. (13) becomes a constraint on the empirical variance of the weights instead. This is the same as the constraint on the variance of the weights in Eq. (12) of Glorot and Bengio (2010).

With this extra assumption, our constrained linear regression problem becomes:

$$\begin{aligned} \min_{b^L, w_j^L, j=1,2,\ldots,m} \quad & \sum_{i \in S} \left(y_i - b^L - \sum_{j=1}^{m} w_j^L x_{i,j}^L \right)^2 \\ \text{such that} \quad & \sum_{j=1}^{m} \left(w_j^L\right)^2 = \frac{1+m}{2}, \\ & \sum_{j=1}^{m} w_j^L = 0, \end{aligned} \tag{21}$$

with the Lagrange function, $\mathcal{L}(\lambda_1, \lambda_2, b^L, w_j^L, j = 1, \ldots, m)$:

$$\mathcal{L}\left(\lambda_1, \lambda_2, b^L, w_j^L, j = 1, \ldots, m\right) = \sum_{i \in S} \left(y_i - b^L - \sum_{j=1}^{m} w_j^L x_{i,j}^L \right)^2 + \lambda_1 \left(\sum_{j=1}^{m} \left(w_j^L\right)^2 - \frac{1+m}{2} \right) + \lambda_2 \sum_{j=1}^{m} w_j^L. \tag{22}$$

We solve this Lagrange function for λ_1, λ_2, $\mathbf{w}^L$ and b^L in Appendix B. We first derive the optimal value of λ_1 and λ_2 using the singular value decomposition of $\mathbf{x}^c$. Given these optimal values for λ_1 and λ_2, we derive the following optimal value for the weights:

$$\mathbf{w}^L = \left(\left(\mathbf{x}^c\right)^T \mathbf{x}^c + \lambda_1 \mathbf{I} \right)^{-1} \left(\left(\mathbf{x}^c\right)^T \mathbf{y}^c - \frac{1}{2} \lambda_2 \mathbf{1} \right), \tag{23}$$

with $\mathbf{x}^c$ and $\mathbf{y}^c$ as defined before in Eq. (17) and (18). Given these weights, the bias is calculated as in Eq. (19).

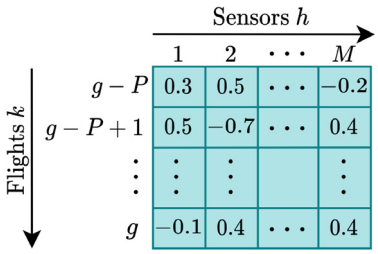


Fig. 2. A schematic example of a data sample $\mathbf{Z}$ that is used as input to the CNN.

3. Case study for regression problems

We apply our proposed methodology to predict the Remaining Useful Life (RUL, time left until failure) of aircraft engines in the C-MAPSS dataset (Saxena & Goebel, 2008). This dataset contains simulated sensor measurements of aircraft turbofan engines. In total, the measurements of 21 sensors around the engine are considered, such as the pressure at the High Pressure Combustor or the physical fan speed of an engine. For each sensor, one measurement per engine per flight is simulated by the NASA Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) simulator. Over time, the health of each engine degrades. This degradation is captured by the sensor measurements. Our goal is to develop a model that uses these sensor measurements to predict the RUL of the engines in the C-MAPPS dataset. The C-MAPPS dataset is widely used in literature to develop RUL prediction models, with 250 papers published on this dataset (de Pater & Mitici, 2022; Lee & Mitici, 2023; Vollert & Theissler, 2021). More information on this dataset can be found in Ramasso and Saxena (2014).

In this paper, we consider subset FD001 of the C-MAPSS dataset. The training set of FD001 contains 100 engines. For each engine, the sensor measurements are simulated for each flight from the installation of this engine until failure. Subset FD001 also contains a test set with 100 test engines. For each test engine, the sensor measurements are terminated somewhere before the failure of the engine. The goal is to predict the RUL of the test engine at this point.

We use a Convolutional Neural Network (CNN) to predict the RUL. Seven of the 21 sensors in the C-MAPSS dataset have constant measurements over time. We thus use the remaining $M = 14$ sensors as input to the CNN. We first normalize the measurements of these 14 sensors using min–max normalization (Li, Ding, & Sun, 2018). After a flight g of an engine, we then use a data sample $\mathbf{Z}$ with these normalized sensor measurements of the past P flights as input to the CNN, i.e.,:

$$\mathbf{Z} = [\mathbf{z}_{g-P}, \mathbf{z}_{g-P+1}, \ldots, \mathbf{z}_g], \tag{24}$$

where P is the window size, and z_k are the normalized sensor measurements of flight k:

$$\mathbf{z}_k = [\hat{z}_{k1}, \hat{z}_{k2}, \ldots, \hat{z}_{kM}], \tag{25}$$

with $\hat{z}_{kh}$ the normalized sensor measurement of sensor h and flight k. Fig. 2 shows an example of such an input sample $\mathbf{Z}$. The true label of this data sample is the RUL of the considered engine after flight g.

The considered CNN has been proposed in de Pater, Reijns, and Mitici (2022) and Li et al. (2018). This CNN consists of 5 convolutional layers. The first 4 convolutional layers each have 10 one-dimensional kernels of size 10×1. The last convolutional layer has one one-dimensional kernel of size 3×1. Same padding is applied to all convolutional layers. After the 5 convolutional layers, two fully connected layers are added. The first fully connected layer has 100 nodes as output, and uses a dropout rate of 0.5. The last connected layer uses these $m = 100$ nodes as input, and outputs the RUL prediction. This CNN has 45.372 parameters. All layers use the tanh activation function, except the last layer, which uses a linear activation function. Last, following de Pater et al. (2022) and Li et al. (2018), we use the common piece-wise linear RUL target function, where we aim to predict a RUL of 125 flights when the actual RUL is larger than 125 flights.

We split the engines in the training set in 80 engines for training the neural network, and 20 engines for the validation. Moreover, we use a window size of $P = 30$ in Eq. (24). With this window size, we create $N = 13890$ data samples for the training set S, and 3841 data samples for the validation set. The weights are further optimized using the Adam optimizer (Kingma & Ba, 2014), with a batch size of 512 samples, 200 epochs and a learning rate of 0.001. The considered loss function is the (Mean) Squared Error (as in Eq. (1)). When estimating the RUL of the test engines in the given test set, we use the weights of the neural network that give the lowest validation loss.

4. Results – Convergence of the weights for regression problems

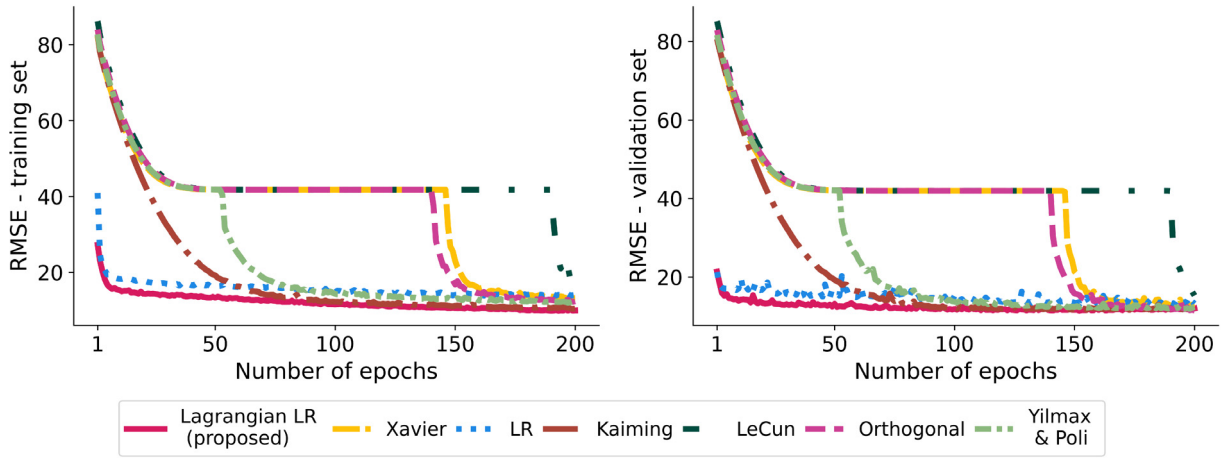
4.1. Benchmark strategies

In this section, we compare the convergence of the weights of the neural network for several weight initialization strategies. We refer to our proposed approach to initialize the weights of the last layer, that combines linear regression with Lagrange, as the "Lagrangian LR" strategy. To avoid the vanishing/exploding gradient problem, we impose a constraint on the weights in the proposed strategy. To evaluate the effectiveness of this constraint, we also initialize the weights and biases following our proposed strategy, but without any constraint on the weights of the last layer in Eq. (14). Instead, we initialize the weights and bias of the last layer with the least squares solution of the linear regression of the true label y_i on the estimated label $\hat{y}_i = b^L + \sum_{j=1}^{m} w_j^L x_{i,j}^L$ (see Eq. (2)). We refer to this benchmark strategy as "LR" (Linear Regression). As other benchmark strategies, we use Xavier initialization (Glorot & Bengio, 2010) in all layers, including the last layer, Kaiming initialization (He et al., 2015), LeCun initialization (LeCun et al., 2012), Yilmaz & Poli initialization (Yilmaz & Poli, 2022) and orthogonal initialization (Saxe et al., 2013).

4.2. Training of the neural network under different weight initialization strategies

Fig. 3 shows the Root Mean Squared Error (RMSE) of the training and validation set after each epoch for the considered strategies. Table 1 shows the corresponding minimum value of the RMSE of the training and validation set after various number of epochs. We implement the neural network in Python with Pytorch, and train the neural network on a computer with 4 Intel Core i7 CPU cores. With this computer, it takes 32 s to calculate the singular value decomposition of $\mathbf{x}^c$, and it takes less than 1 s to find an optimal value of λ. In total, the weight initialization of the last layer with the proposed approach takes 35 s. Training the neural network for one epoch takes on average 10 s. Thus, initializing the weights with the proposed approach takes the same time as training the neural network for roughly 4 epochs.

After the initialization of the weights using the proposed Lagrangian LR approach, the RMSE of the training set is only 39.0 with and without dropout, while the RMSE of the validation set is only 38.9. The training data is thus not overfitted when initializing the weights with the proposed approach. The weights



(a) RMSE of the training set for all epochs. (b) RMSE of the validation set for all epochs.

Fig. 3. RMSE of the training and validation set after each epoch, for the proposed strategy and the benchmark strategies.

Table 1
The minimum (Min.) RMSE obtained after a various number of epochs for the training (Train.) and validation (Valid.) set, for the proposed strategy and the benchmark strategies. The RMSE of the test set is calculated with the weights that give the lowest validation loss. The lowest RMSE is denoted in bold.

Initialization strategy	Min. RMSE	Number of epochs 1	10	25	50	75	100	125	150	175	200	Test RMSE
Proposed strategy												
Lagrangian LR (Proposed)	Train.	**27.2**	**15.3**	**14.3**	**13.3**	**12.5**	**11.5**	**11.0**	**10.6**	**10.2**	**9.9**	**12.5**
	Valid.	21.4	**14.0**	**13.2**	**12.5**	**12.1**	**11.7**	**11.6**	**11.5**	**11.4**	**11.4**	
Benchmark strategies												
Xavier (Glorot & Bengio, 2010)	Train.	82.1	61.4	45.5	41.8	41.8	41.8	41.8	23.0	14.0	13.0	13.0
	Valid.	80.6	61.8	46.0	42.1	42.0	42.0	42.0	21.4	13.2	12.6	
LR – Linear Regression	Train.	40.9	18.3	17.0	16.3	15.9	14.8	14.2	13.9	13.9	13.7	13.2
	Valid.	**21.3**	16.7	15.8	14.8	14.8	14.0	13.5	13.4	13.1	13.0	
Kaiming (He et al., 2015)	Train.	82.5	60.5	37.6	19.2	13.7	12.2	11.6	11.1	10.7	10.4	12.7
	Valid.	80.6	60.8	38.0	19.3	13.9	12.1	11.9	11.7	11.7	11.7	
LeCun (LeCun et al., 2012)	Train.	86.2	65.0	47.0	41.8	41.8	41.8	41.8	41.8	41.8	16.9	14.2
	Valid.	85.1	65.4	47.6	42.1	42.0	42.0	42.0	42.0	42.0	15.3	
Orthogonal (Saxe et al., 2013)	Train.	84.0	63.3	46.3	41.8	41.8	41.8	41.8	16.8	13.2	12.2	12.6
	Valid.	82.9	63.7	46.8	42.0	42.0	42.0	42.0	15.8	12.2	11.6	
Yilmaz & Poli (Yilmaz & Poli, 2022)	Train.	82.7	62.3	45.8	41.8	17.2	14.3	13.5	12.8	12.3	12.0	**12.5**
	Valid.	81.6	62.7	46.4	42.1	16.9	13.7	12.8	12.3	12.1	11.9	

of the neural network subsequently quickly convergence during the back-propagation algorithm. The lowest validation RMSE of 11.4 is therefore obtained after 152 epochs, after which the validation RMSE does not decrease any further, and even slightly increases. At this point, the weights of the neural network have thus converged. Using the weights that give the lowest validation loss, the RMSE obtained in the test set is only 12.5. Overall, the weights of the neural network thus start at a good point with the considered approach, i.e., with a small initial training and validation loss, and quickly converge during the back-propagation algorithm.

When we initialize the weights of the last layer following the Xavier strategy as well, the RMSE of the training set is 90.5 with and without dropout, while the RMSE of the validation set is 92.0. The weights of the neural network thus start at a worse point compared to the proposed strategy: The initial validation loss is 2.4 times larger. During the back-propagation algorithm, the weights initially quickly converge, until the weights of the neural network get stuck in a local optimum at epoch 46, when the RMSE of the training set is 41.8. At epoch 147, the neural network escapes the local optimum and the weights of the neural network then quickly further converge, obtaining a minimum validation RMSE of 12.6 after 198 epochs, with a corresponding test RMSE of 13.0. In contrast, our proposed LR Lagrangian strategy already obtains a validation RMSE of 12.6 after only 49 epochs. Thus, we need 75% less epochs to reach the same validation loss when using the LR Lagrangian strategy. Comparing the Xavier initialization (Glorot & Bengio, 2010) to the Lagrangian LR strategy proposed in this paper, we conclude that by using the LR Lagrangian strategy (i) the initial starting point of the weights provide a lower training and validation loss, (ii) we avoid getting stuck in a local minimum, and (iii) we obtain a quicker converge of the weights.

With the LR strategy, the RMSE of the training set after the weight initialization is 48.3 with and 20.8 without dropout, while the RMSE of the validation set is 19.0. Moreover, the weights of the neural network quickly converge during the first few epochs, leading to a considerable decrease in the RMSE of the training and validation set. However, this convergence is slow compared to the convergence of the weights with the proposed strategy: After 200 epochs, the validation RMSE is 11.4 with the proposed strategy, while it still is 13.0 with the LR strategy. The results with the LR strategy are therefore even worse than with Xavier initialization, where a lower loss for both the training and the validation set is obtained within 200 epochs. It is thus important to not only focus on initializing the weights close to an optimum, but also on initializing the weights such that the vanishing/exploding gradient problem is mitigated.

Table 2
The minimum RMSE obtained after various number of epochs for the training (Train.) and validation (Valid.) set using our proposed strategy. Here, we report the results when using the full training set to initialize the weights, and the results of the 10-fold cross validation, where only 10% of the training set is used for weight initialization in each validation split. For the 10-fold cross validation, we report the mean, minimum (min) and maximum (max) value of the minimum RMSE over the 10 performed validation splits.

			Number of epochs						
			1	10	25	50	100	150	200
10% of the training data set used for weight initialization with 10-fold cross validation	Mean RMSE	Train.	27.2	15.4	14.3	13.3	11.6	10.5	9.9
		Valid.	21.4	14.0	13.3	12.5	11.7	11.5	11.5
	Min RMSE	Train.	27.1	15.2	14.1	13.3	11.5	10.5	9.8
		Valid.	21.0	13.9	13.1	12.5	11.6	11.4	11.3
	Max RMSE	Train.	27.4	15.5	14.4	13.3	11.6	10.6	10.0
		Valid.	21.7	14.3	13.4	12.6	11.8	11.6	11.6
Full training set used for weight initialization	RMSE	Train.	27.2	15.3	14.3	13.3	11.5	10.6	9.9
		Valid.	21.4	14.0	13.2	12.5	11.7	11.5	11.4

The initial loss of the other three benchmark strategies is large, with a validation RMSE around 90 after the weight initialization. Moreover, just as with the Xavier initialization strategy, the neural network gets stuck in a local optimum with the Lecun (LeCun et al., 2012), orthogonal (Saxe et al., 2013) and Yilmaz & Poli (Yilmaz & Poli, 2022) initialization strategies. However, in the end, the neural network converges to a similar final training, validation and test accuracy for all considered benchmark strategies: Yilmax & Poli initialization even has the same final test accuracy. The best performing benchmark strategy is the Kaiming weight initialization (He et al., 2015). For this strategy, the initial RMSE of the training set is 90.4 with and without dropout, while the initial validation RMSE is 91.8. The initial validation RMSE is thus 2.4 times larger than with the proposed strategy. Using Kaiming initialization, however, the weights quickly converge to an optimum without getting stuck in a local optimum. After 148 epochs, the minimum validation RMSE of 11.7 is already reached. However, this validation RMSE is already reached after 97 epochs with the proposed approach, i.e., we need 34% less epochs to reach the same validation loss.

4.3. Results – initialization of the weights with only a part of the training set

Calculating the singular value decomposition of large training sets is time-consuming. For large training sets, it might therefore be beneficial to initialize the weights with only a part of the training set instead. In this section, we therefore analyze the convergence of the weights if we only employ 10% of the training set to initialize the weights of the last layer. Here, we use 10-fold cross validation by splitting the training set randomly in 10 non-overlapping subsets. For each validation split, we use one subset to initialize the weights of the neural network. We then subsequently train the neural network with the full training set.

The singular value decomposition with 10% of the training set takes on average only 0.41 s, while initializing the weights takes on average only 0.83 s. This is 42 times faster than when considering the full dataset (see Section 4). Table 2 shows the mean, minimum and maximum value of the minimum RMSE of the validation and training set after a various number of epochs. Here, the mean, minimum and maximum are taken over the results of the 10 validation splits. We also show the minimum RMSE of the validation and training set when considering the full training set to initialize the weights. The mean training and validation RMSE from the cross-validation is very close to the training and validation RMSE when using the full training set for the weight initialization (a maximum of 0.1 difference). Moreover, after the first 10 epochs, the minimum training and validation RMSE are close together for all 10 validation splits (a maximum of 0.3 difference after the first 10 epochs). For the considered dataset, the results are therefore very similar when using only 10% of the training dataset or the full dataset to initialize the weights, while initializing the weights with only 10% of the training set is 42 times faster.

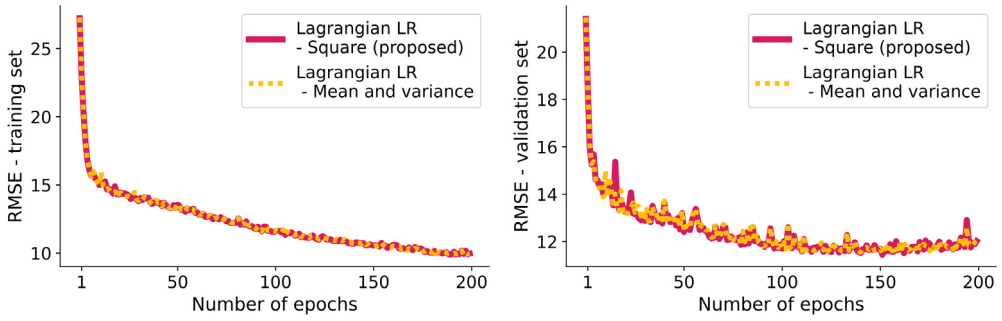
4.4. Results with a mean weight of zero

In this section we analyze the benefits of assuming that the mean of the weights has to be zero. Fig. 4 shows the RMSE of the training and validation set after each epoch for the proposed approach (Lagrangian LR – Square), and for the proposed approach with the additional restriction that the mean of the weights is zero (Lagrangian LR – Mean and variance, see Section 2.5). The RMSE of the training and validation set is nearly equal for both strategies. For our approach, there is thus no additional benefit in assuming that the mean of the weights is zero for the last layer, in contrast with the Xavier (Glorot & Bengio, 2010) and Kaiming (He et al., 2015) weight initialization strategies.

5. Results for classification problems

The focus of this paper is on neural networks that solve a regression problem. For completeness, we apply in this Section a variant of our method to neural networks for classification problems. Neural networks that solve a classification problem have a different activation function in the last layer, a different loss function and often a different activation function throughout the neural network than considered in the methodology in Section 2. Thus, we cannot directly employ the mathematical derivations from Section 2 to these neural networks as well. Instead, we adjust our methodology for classification neural networks as follows.

We first introduce the notation for the last layer of a classification neural network. An overview of this notation is in Fig. 5. Let $y_i \in \{1, 2, \ldots, n\}$ be the true label for a sample $i \in S$. Here, n denotes the number of classes. We assume that the last layer L of a classification neural network still is a fully connected layer. This layer still has m input nodes, but now has n, instead of one, output nodes. As before, let $\mathbf{x}_i^L$ denote the vector with the activated input of layer L for sample $i \in S$. Here, $\mathbf{x}_i^L = [x_{i,1}^L, x_{i,2}^L, \ldots, x_{i,m}^L]$, with $x_{i,j}^L$ the jth hidden input state of layer L for sample i. Let $\mathbf{W}^L = [\mathbf{w}_1^L, \mathbf{w}_2^L, \ldots, \mathbf{w}_n^L]$ denote the matrix with the weights of the last layer L. Here, $\mathbf{w}_h^L = [w_{1,h}^L, w_{2,h}^L, \ldots, w_{m,h}^L]$ ($h \in \{1, 2, \ldots, n\}$) denotes the weights connecting the hidden input state $\mathbf{x}_i^L$ of a sample i to the hth output node of this last layer L. Similarly, let $\mathbf{b}^L = [b_1^L, \ldots, b_n^L]$ denote the vector with the biases of the last layer L, where b_h^L denotes the bias belonging to the hth output node of layer L. For a sample i, the value $y_{i,h}^L$ of this hth output node is calculated as $y_{i,h}^L = \sum_{j=1}^{m} w_{j,h}^L x_{i,j}^L + b_h^L$. The total output of layer L for a sample i is denoted by $\mathbf{y}_i^L = [y_{i,1}^L, y_{i,2}^L, \ldots, y_{i,n}^L]$.



(a) RMSE of the training set for all epochs (b) RMSE of the validation set for all epochs

Fig. 4. The RMSE of the training set after each epoch, for the proposed strategy with only a restriction on the sum of the squared weights (Lagrangian LR – Square), and for a Lagrangian regression with a restriction on the empirical variance and mean of the weights (Lagrangian LR – Mean and variance).

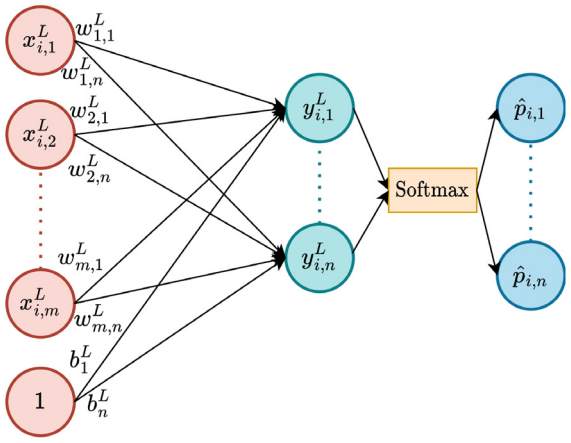


Fig. 5. Schematic overview of the last layer of assumed neural network for classification problems for a training sample $i \in S$.

We use the Softmax activation function to convert the output values $\mathbf{y}_i^L$ to probabilities. The estimated probability $\hat{p}_{i,h}$ that sample i belongs to class h is:

$$\hat{p}_{i,h} = \frac{\exp\left(y_{i,h}^L\right)}{\sum_{g=1}^{n} \exp\left(y_{i,g}^L\right)} \tag{26}$$

The loss of these estimated probabilities is calculated by the cross entropy:

$$\text{Loss} = -\sum_{i \in S} \sum_{h=1}^{n} \mathcal{I}\left(y_i = h\right) \log\left(\hat{p}_{i,h}\right), \tag{27}$$

where $\mathcal{I}\left(y_i = h\right)$ is 1 if $y_i = h$, and zero otherwise. Last, we assign each sample i to the class h with the largest estimated probability $\hat{p}_{i,h}$. Let $\hat{y}_i$ denote the class to which sample i is assigned. From this, we calculate the accuracy:

$$\text{Accuracy} = 100 \cdot \frac{\sum_{i \in S} \mathcal{I}\left(y_i = \hat{y}_i\right)}{N} \tag{28}$$

As before, we initialize the weights of the first $L-1$ layers of the neural network randomly, using several weight initialization methods. The aim is now to find the weights $\mathbf{W}^L$ and the bias $\mathbf{b}^L$ of the last layer L such that the cross entropy loss is minimized. However, we again want to prevent the exploding/vanishing gradient problem. With a logistic regression, we do not know analytically the optimal value of the sum of the squared weights. We also do not know how to analytically derive a value for λ given such a sum. However, we can still apply Ridge logistic regression with a randomly chosen λ to regularize the weights. In Ridge logistic regression, we minimize the following function (Hastie et al., 2009):

$$\min_{\mathbf{W}^L, \mathbf{b}^L} = -\sum_{i \in S} \sum_{h=1}^{n} \mathcal{I}\left(y_i = h\right) \log\left(\hat{p}_{i,h}\right) + \lambda \left(\sum_{h=1}^{n} \sum_{j=1}^{m} \left(w_{j,h}^L\right)^2 \right). \tag{29}$$

This is very similar to the Lagrangian function in Eq. (15). The main difference is that we now consider another loss function and that we choose λ ourselves. Future research should be conducted to derive a good value for λ (either empirically or analytically) in classification problems. To initialize the weights in the last layer L, we thus perform a forward pass to obtain the hidden input states $\mathbf{x}_i^L$ for all samples $i \in S$. We then use the Ridge logistic regression of the actual labels y_i on the hidden states $\mathbf{x}_i^L$ (for all samples $i \in S$) to find the weights $\mathbf{W}^L$ and the biases $\mathbf{b}^L$ that minimize the loss given λ.

5.1. Implementation - CIFAR-100 dataset

We test the above approach on the images of the CIFAR-100 dataset (Krizhevsky et al., 2009). Each image in this dataset has a size of 32 by 32 pixels, and belongs to one out of a hundred classes, such as "bicycle" or "camel". The objective is to correctly classify the images. The dataset is divided into 10.000 test images, and 50.000 training images. We further divide the training images into 45.000 images for training the neural network, and 5.000 images for validation. We preprocess the data by scaling the input images using z-score standardization. Moreover, to prevent overfitting, we apply Random Augmentation (Cubuk, Zoph, Shlens, & Le, 2020) on the 45.000 training images, with 3 operations of a magnitude of 15.

With this training images, we train two well-known neural networks to classify the images, namely ResNet-18 and ResNet-34 (He et al., 2016). These neural networks are both variants on the general ResNet developed in He et al. (2016). ResNet-18 is the smallest variant, with 18 layers and over 11 million parameters, while ResNet-34 is a larger variant with 34 layers and over 21 million parameters. The last layer of both neural networks is a single fully connected layer, to which we apply our approach. ResNets are initially developed for the ImageNet dataset, which has relatively large images. In He et al. (2016), all images in this dataset are cropped to have a size of 224 by 224 pixels. Following this, we therefore resize the images in the CIFAR-100 dataset to a size of 224 to 224 pixels as well, using bilinear interpolation. We train both neural networks for 100 epochs using stochastic gradient descent with a momentum of 0.9, a batch size of 256 and weight decay of 1^{-4}, to prevent overfitting. The initial learning rate is 0.01, and is multiplied by 0.1 after every 25 epochs.

Table 3
The maximum accuracy (max. acc.) in percent obtained after a various number of epochs for the training (Train.) and validation (Valid.) set. Here, we consider several weight initialization strategies, with two different ways to initialize the weights in the last layer; According to the considered weight initialization strategy, or with the proposed strategy of Ridge logistic regression (Proposed). The accuracy of the test set is calculated with the weights that give the lowest validation loss. The highest accuracy per weight initialization strategy is denoted in bold.

Weight initialization	Initialization last layer	Max. acc. (%)	Number of epochs 0	10	25	50	75	100	Test acc. (%)
ResNet-18 (He et al., 2016)									
Kaiming (original) (He et al., 2015)	He et al. (2015)	Train	1.01	22.44	45.59	55.38	56.93	57.14	62.20
		Valid.	1.00	30.42	49.98	61.30	62.16	62.20	
	Proposed	Train	**3.72**	**24.82**	**46.64**	**56.92**	**58.28**	**58.64**	**63.47**
		Valid.	**9.34**	**34.72**	**52.26**	**62.46**	**63.30**	**63.30**	
Xavier (Glorot & Bengio, 2010)	Glorot and Bengio (2010)	Train.	1.01	27.73	51.46	63.90	66.13	66.13	67.24
		Valid.	1.00	38.83	54.66	66.10	67.14	67.14	
	Proposed	Train.	**3.73**	**31.14**	**52.84**	**65.38**	**66.82**	**67.36**	**67.83**
		Valid.	**5.43**	**39.00**	**55.58**	**67.46**	**68.20**	**68.32**	
LeCun (LeCun et al., 2012)	LeCun et al. (2012)	Train.	1.01	28.51	51.71	63.83	65.67	66.00	67.06
		Valid.	1.22	37.26	53.44	66.52	67.68	67.84	
	Proposed	Train.	**3.72**	**31.06**	**52.65**	**65.01**	**66.75**	**67.09**	**67.47**
		Valid.	**9.34**	**40.26**	**55.94**	**67.02**	**68.08**	**68.32**	
Orthogonal (Saxe et al., 2013)	Saxe et al. (2013)	Train.	1.05	28.13	51.87	64.17	66.14	66.42	**67.50**
		Valid.	1.12	34.32	55.44	66.08	67.06	67.08	
	Proposed	Train.	**4.30**	**30.82**	**53.13**	**65.29**	**66.94**	**67.43**	67.20
		Valid.	**5.96**	**37.30**	**56.48**	**67.04**	**68.04**	**68.04**	
Yilmax & Poli (Yilmaz & Poli, 2022)	Yilmaz and Poli (2022)	Train	1.03	3.00	10.00	12.94	13.35	13.48	21.33
		Valid.	1.12	5.40	13.80	20.54	21.12	21.12	
	Proposed	Train.	**2.05**	**3.54**	**11.38**	**14.62**	**14.89**	**15.16**	**22.79**
		Valid.	**4.12**	**5.58**	**17.80**	**22.76**	**22.96**	**23.02**	
ResNet-34 (He et al., 2016)									
Kaiming (original) (He et al., 2015)	He et al. (2015)	Train.	0.94	24.23	51.59	65.04	67.62	67.90	66.04
		Valid.	0.86	33.18	54.42	65.00	65.66	65.66	
	Proposed	Train.	**2.79**	**28.50**	**54.00**	**68.64**	**70.42**	**70.90**	**67.23**
		Valid.	**7.22**	**38.76**	**56.24**	**66.14**	**67.12**	**67.26**	
Xavier (Glorot & Bengio, 2010)	Glorot and Bengio (2010)	Train.	0.94	30.38	56.87	72.77	75.53	76.00	69.52
		Valid.	0.86	37.94	57.70	69.28	70.16	70.16	
	Proposed	Train.	**2.78**	**34.37**	**58.93**	**75.65**	**77.88**	**78.73**	**69.84**
		Valid.	**7.20**	**40.96**	**59.64**	**69.62**	**70.38**	**70.38**	
LeCun (LeCun et al., 2012)	LeCun et al. (2012)	Train.	0.94	31.38	57.45	73.25	75.80	76.09	68.56
		Valid.	0.86	38.38	60.14	69.46	70.22	70.26	
	Proposed	Train.	**2.79**	**34.30**	**58.96**	**75.53**	**77.69**	**78.04**	**69.86**
		Valid.	**7.22**	**40.42**	**60.56**	**70.16**	**70.42**	**70.50**	
Orthogonal (Saxe et al., 2013)	Saxe et al. (2013)	Train.	1.00	31.23	57.85	74.12	76.71	77.35	69.13
		Valid.	1.02	39.08	**59.54**	69.40	70.06	70.24	
	Proposed	Train.	**2.37**	**34.95**	**59.12**	**76.19**	**78.13**	**78.66**	**69.99**
		Valid.	**6.84**	**40.60**	58.12	**69.42**	**70.34**	**70.38**	
Yilmax & Poli (Yilmaz & Poli, 2022)	Yilmaz and Poli (2022)	Train	0.95	2.71	8.10	11.51	12.16	12.34	19.02
		Valid.	0.92	4.64	11.32	17.86	18.70	18.70	
	Proposed	Train.	**1.82**	**3.01**	**8.58**	**12.55**	**13.24**	**13.36**	**20.25**
		Valid.	**3.02**	**4.96**	**13.10**	**19.28**	**19.68**	**19.76**	

We implement the logistic regression with Ridge using the Scikit-learn package (Pedregosa et al., 2011), with the "lbfgs" solver. This solver applies a quasi-Newton method (Hastie et al., 2009) to optimize the weights of the logistic regression. We select λ from {1, 10, 100, 1000, 10000}. Specifically, we first calculate the sum of the squared weights in the last layer L when we initialize all weights of the neural network with Xavier initialization. Let the value of this sum be denoted by α. We then select the smallest value of $\lambda \in \{1, 10, 100, 1000, 10000\}$ for which the sum of the squared weights in the last layer is equal to or smaller than this value α. For both ResNet-18 and ResNet-34, this procedure gives $\lambda = 1000$. Last, we perform the logistic regression on all training images without any random augmentation. With this, it takes between 105 and 143 s to initialize the weights in the last layer with Ridge logistic regression for ResNet-18, and between 135 and 211 s for ResNet-34. Training the neural network for one epoch on 1 NVIDIA Tesla V100S GPU takes between 80 and 90 s for ResNet-18, and between 90 and 100 s for ResNet-34. The logistic regression thus takes roughly as long as training the neural network for two epochs.

Since we do not analytically derive the optimal value for λ, we are not restricted to applying the proposed methodology after Xavier weight initialization only. Instead, we initialize the weights in the last layer with Ridge logistic regression in combination with each benchmark weight initialization method. For simplicity, we use the same value of λ for all weight initialization methods.

5.2. Results – convergence of the weights for classification problems

Table 3 shows the maximum accuracy after training ResNet-18 and ResNet-34 for several number of epochs. The accuracy after epoch 0 is the initial accuracy, i.e., the accuracy after the weight initialization without any training of the neural network. The initial validation accuracy is between 4.12% and 9.34% for ResNet-18, and between 3.02% and 7.22% for ResNet-34. This initial accuracy is relatively small compared to the initial loss of the regression problem, which is already halfway between the loss with random weight initialization and the final obtained loss after training. This might be because we consider a more complicated problem, with

100 instead of only one possible output. Moreover, we consider a more complicated neural network: In the regression problem, the neural network has 45.372 parameters, while ResNet-18 and ResNet-34 have over 11 million and 21 million parameters respectively. With many more randomly initialized parameters, the characteristics of the input might have largely disappeared in the hidden state of the last layer for both ResNets, compared to the regression problem. This also might be the reason that ResNet-34 has a lower initial validation accuracy than ResNet-18.

However, for both neural networks, the initial accuracy with the proposed weight initialization method is still higher than without, for all considered initialization methods. This leads to a slightly higher training and validation accuracy throughout the training process: For both ResNet-18 and ResNet-34 and with all initialization methods, the training and validation accuracy is slightly higher when applying Ridge logistic regression to initialize the weights of the last layer. The only exception is the validation accuracy after 25 epochs with orthogonal initialization for ResNet-34, which is higher when not using Ridge logistic regression in the beginning. However, also in this case, applying Ridge logistic regression gives a higher validation accuracy after 0, 10, 50, and 100 epochs. With the proposed methodology, we thus obtain a slightly faster convergence of the weights.

For all initialization methods, the final training and validation accuracy are slightly higher with the proposed methodology. The test accuracy is also higher when the weights are initialized with Ridge logistic regression, except when we use orthogonal initialization in ResNet-18: In this case, the test accuracy is 67.50% when using the orthogonal initialization without Ridge logistic regression, while it is 67.20% when using orthogonal initialization with Ridge logistic regression. The highest test accuracy of 67.83% for ResNet-18 is obtained when combining Xavier initialization with our approach, while the highest test accuracy of 69.99% for ResNet-34 is obtained when combining orthogonal initialization with our approach.

5.3. Results – combination of ridge logistic regression with transfer learning

Classification neural networks are often trained using transfer learning., i.e., the weights of the neural network are (partly) initialized with the weights from another neural network with (partly) the same structure, trained on another dataset (the source dataset). The weights are then fine-tuned for the target dataset using a gradient descent method (Vasilev, 2019).

In this section, we test how our approach works performs combined with transfer learning. We therefore initialize all weights, except the weights of the last layer, in ResNet-18 and ResNet-34 with the pre-trained weights of Torchvision in Pytorch. These weights are obtained by training the neural networks to classify the images in the ImageNet-1K dataset (Russakovsky et al., 2015). This dataset contains 1000 classes, instead of 100. Because there are different classes in the ImageNet-1K dataset and in the CIFAR-100 dataset, we cannot initialize the weights of the last layer with the weights from Torchvision. Instead, we initialize the weights in the last layer with the proposed methodology (Ridge logistic regression), and with all benchmark methods. For the Ridge logistic regression, we use the same λ as in Section 5.2. We then train the neural network for 50 epochs using stochastic gradient descent with a momentum of 0.9, a batch size of 256 and a weight decay of 1^{-4}. The initial learning rate is still 0.01, but we now multiply this learning rate with 0.1 after 10 and 30 epochs.

The results are in Table 4. The initial validation accuracy after epoch 0, i.e., without training the neural network, is already 53% and 55% for ResNet-18 and ResNet-34 respectively. Because of this, the weights converge fast to a (local) optimum: Throughout the training process, both the training and the validation accuracy are higher with the proposed weight initialization method than with the benchmark methods. For both ResNets, LeCun initialization (LeCun et al., 2012) is the best benchmark method. For ResNet-18, LeCun initialization obtains its highest validation accuracy of 81.32% after 49 epochs, while we already obtain this validation accuracy in 13 epochs. For ResNet-34, LeCun initialization obtains its highest validation accuracy of 84.18% after 50 epochs, while we already obtain this validation accuracy after 14 epochs. Moreover, with the proposed methodology, we obtain a higher final training, validation and test accuracy within 50 epochs than with all benchmark methods. The convergence of the weights is thus accelerated by combining Ridge logistic regression with transfer learning.

6. Conclusions

In this paper, we introduce a new initialization method for the weights in the last layer of a neural network. We assume that this neural network solves a regression problem and that it uses an activation function that has a unit derivative of 1 at 0. Here we focus both on (i) accelerating the convergence of the weights during the back-propagation algorithm by mitigating the vanishing/exploding gradient problem, and (ii) initializing the weights close to an optimum point by minimizing the initial training loss. To accelerate the convergence of the weights, we impose that the variance of the outputs and the gradients of each layer in the neural network should be equal after the weight initialization, following Glorot and Bengio (2010). From this requirement, we analytically derive a constraint on the weights in the last layer. We then analytically derive the optimal weights and bias of the last layer, i.e., the weights and bias that minimize the initial training loss, while fulfilling this derived constraint using the Lagrange function.

We apply this initialization strategy to a CNN that predicts the RUL of aircraft engines. Our proposed strategy initializes the weights such that the initial training and validation loss are relatively small. Moreover, the proposed strategy prevents the weights of the CNN to get stuck in a local optimum. The weights therefore converge very fast. The minimum validation loss obtained with Xavier initialization (Glorot & Bengio, 2010) after 198 epochs is already obtained after only 49 epochs with our approach. Moreover, compared to the best benchmark strategy (Kaiming initialization (He et al., 2015)), we need 34% less epochs to reach the same validation loss. To further analyze our proposed methodology, we also show that it is sufficient for the considered data set to use only a part of the training set for the weight initialization. Moreover, we show that with our proposed initialization approach, it is not necessary to assume that the mean of the weights is zero, as in Glorot and Bengio (2010) and He et al. (2015).

Last, we adjust our proposed methodology to apply it to a neural network with any type of activation function that solves a classification problem, by using logistic regression with Ridge regularization. We apply this to ResNet-18 and ResNet-34 (He et al., 2016), and classify the images in the CIFAR-100 dataset (Krizhevsky et al., 2009). When training ResNet from scratch, we obtain a slightly higher initial accuracy and a slightly faster convergence of the weights with our approach. However, our approach works best when combined with transfer learning. In this case, the initial validation accuracy is already 53% and 55% for ResNet-18 and ResNet-34 respectively. This leads to a faster weight convergence and a higher test accuracy than with the benchmark methods.

Table 4
The maximum accuracy (max. acc.) in percent obtained after a various number of epochs for the training (Train.) and validation (Valid.) set when using transfer learning. For the weights in the last layer, we consider several benchmark weight initialization strategies and the proposed strategy of Ridge logistic regression (Proposed). The accuracy of the test set is calculated with the weights that give the lowest validation loss. The highest accuracy is denoted in bold.

Weight initialization last layer	Max acc. (%)	Number of epochs 0	10	20	30	40	50	Test acc. (%)
ResNet-18 (He et al., 2016)								
Proposed	Train.	**20.70**	**73.04**	**81.59**	**83.46**	**84.27**	**84.76**	**81.76**
	Valid.	**53.68**	**78.24**	**81.86**	**81.86**	**81.86**	**81.86**	
Kaiming (He et al., 2015)	Train.	1.01	70.55	79.34	81.50	82.43	82.93	80.25
	Valid.	1.16	75.98	80.84	80.84	81.06	81.06	
Xavier (Glorot & Bengio, 2010)	Train.	1.02	70.96	79.71	81.92	83.00	83.27	79.97
	Valid.	0.98	76.02	80.52	80.74	80.80	80.98	
LeCun (LeCun et al., 2012)	Train.	0.95	71.38	80.00	82.04	83.14	83.50	80.54
	Valid.	1.38	76.48	80.86	81.06	81.24	81.32	
Orthogonal (Saxe et al., 2013)	Train.	1.06	71.54	80.22	82.09	82.99	83.51	80.92
	Valid.	1.20	76.48	80.88	81.28	81.36	81.44	
Yilmax & Poli (Yilmaz & Poli, 2022)	Train.	0.94	61.79	72.56	75.62	77.00	77.56	78.28
	Valid.	1.36	71.16	77.60	77.94	78.06	78.32	
ResNet-34 (He et al., 2016)								
Proposed	Train.	**22.04**	**78.67**	**88.07**	**90.09**	**90.93**	**90.96**	**84.57**
	Valid.	**55.12**	**80.60**	**84.86**	**85.24**	**85.24**	**85.28**	
Kaiming (He et al., 2015)	Train.	0.86	71.63	83.81	86.24	87.37	87.76	82.34
	Valid.	0.94	76.56	83.04	83.04	83.26	83.26	
Xavier (Glorot & Bengio, 2010)	Train.	0.86	73.27	85.16	86.99	88.39	88.59	82.91
	Valid.	0.94	77.62	82.78	82.78	82.98	82.98	
LeCun (LeCun et al., 2012)	Train.	0.86	76.51	87.12	88.96	90.13	90.38	83.03
	Valid.	0.94	78.14	83.54	84.12	84.12	84.18	
Orthogonal (Saxe et al., 2013)	Train.	0.90	76.73	87.36	89.03	90.02	90.34	83.10
	Valid.	0.88	78.22	83.98	83.98	84.06	84.10	
Yilmax & Poli (Yilmaz & Poli, 2022)	Train.	0.87	64.42	78.36	81.97	83.56	83.86	80.35
	Valid.	0.94	73.12	79.64	80.66	80.66	80.66	

Limitations and future research

In this paper, we consider a specific type of problem (i.e., regression problem), with a specific type of neural network (i.e., specific type of activation function). Many problems in machine learning, however, do not satisfy these assumptions: They either do not solve a regression problem and/or use other activation functions. For future research, we therefore plan to extend both the mathematical analysis and the experiments to other types of problems, neural networks and datasets. Specifically, we would like to consider other activation functions, and to empirically find a good value for λ instead of analytically.

Moreover, we have also discussed applying the considered methodology to classification neural networks with any type of activation function. Here, we choose the value of the Lagrangian multiplier λ ourselves. Future work could develop the approach for these classification neural networks further. We find it particularly interesting to determine a good value for λ, either analytically or empirically, and to further research the combination of the proposed approach with transfer learning.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data is publicly and freely available at the NASA Prognostics Center of Excellence Data Set Repository. The code is attached to the paper.

Appendix A. Solution of the minimization problem (Eq. (14)) for λ

In this Appendix, we solve the constrained linear regression problem of Eq. (14) to find the optimal value of λ following Hastie et al. (2009). The Lagrange function $\mathcal{L}(\lambda, b^L, w_j^L, j = 1, 2, \ldots, m)$ of the minimization problem is:

$$\mathcal{L}\left(\lambda, b^L, w_j^L, j = 1, 2, \ldots, m\right) = \sum_{i \in S} \left(y_i - b^L - \sum_{j=1}^{m} w_j^L x_{i,j}^L \right)^2 + \lambda \left(\sum_{j=1}^{m} \left(w_j^L\right)^2 - \frac{1+m}{2} \right). \tag{30}$$

A.0.1. Scaling of the inputs

The solution of this constrained minimization problem is not equivalent to scaling the inputs or the outputs (Hastie et al., 2009). Following Hastie et al. (2009), we therefore first normalize the variables:

$$\begin{aligned} &\mathcal{L}\left(\lambda, b^L, w_j^L, j = 1, 2, \ldots, m\right) \\ &= \sum_{i \in S} \left(y_i - b^L - \sum_{j=1}^{m} w_j^L \bar{x}_j^L + \sum_{j=1}^{m} w_j^L \bar{x}_j^L - \sum_{j=1}^{m} w_j^L x_{i,j}^L \right)^2 \\ &+ \lambda \left(\sum_{j=1}^{m} \left(w_j^L\right)^2 - \frac{1+m}{2} \right) \\ &= \sum_{i \in S} \left(y_i - b^L - \sum_{j=1}^{m} w_j^L \bar{x}_j^L - \sum_{j=1}^{m} w_j^L \left(x_{i,j}^L - \bar{x}_j^L \right) \right)^2 \\ &+ \lambda \left(\sum_{j=1}^{m} \left(w_j^L\right)^2 - \frac{1+m}{2} \right), \end{aligned} \tag{31}$$

with $\bar{x}_j^L$ the mean value of the jth hidden state over all training samples $i \in S$, i.e., $\bar{x}_j^L = \frac{1}{N}\sum_{i\in S} x_{i,j}^L$. We define the centered weight w_j^c and the centered bias b^c as:

$$w_j^c = w_j^L, \tag{32}$$

$$b^c = b^L + \sum_{j=1}^{m} w_j^L \bar{x}_j^L. \tag{33}$$

We then obtain that:

$$\begin{aligned}\mathcal{L}\left(\lambda, b^L, w_j^L, j=1,2,\ldots,m\right) &= \mathcal{L}\left(\lambda, b^c, w_j^c, j=1,2,\ldots,m\right) \\ &= \sum_{i\in S}\left(y_i - b^c - \sum_{j=1}^{m} w_j^c\left(x_{i,j}^L - \bar{x}_j^L\right)\right)^2 \\ &\quad + \lambda\left(\sum_{j=1}^{m}(w_j^c)^2 - \frac{1+m}{2}\right).\end{aligned} \tag{34}$$

Following Hastie et al. (2009), we first analyze the optimal value of b^c. The derivative of the loss with respect to b^c is:

$$\begin{aligned}&\frac{\partial}{\partial b^c}\mathcal{L}\left(\lambda, b^c, w_j^c, j=1,2,\ldots,m\right) \\ &= -2\sum_{i\in S}\left(y_i - b^c - \sum_{j=1}^{m} w_j^c\left(x_{i,j}^L - \bar{x}_j^L\right)\right)\end{aligned} \tag{35}$$

To find the optimum, we set this derivative equal to zero:

$$\sum_{i\in S}\left(y_i - b^c\right) - \sum_{i\in S}\sum_{j=1}^{m} w_j^c\left(x_{i,j}^L - \bar{x}_j^L\right) = 0 \tag{36}$$

First, let us analyze $\sum_{i\in S}\sum_{j=1}^{m} w_j^c\left(x_{i,j}^L - \bar{x}_j^L\right)$:

$$\begin{aligned}\sum_{i\in S}\sum_{j=1}^{m} w_j^c\left(x_{i,j}^L - \bar{x}_j^L\right) &= \sum_{j=1}^{m}\sum_{i\in S} w_j^c x_{i,j}^L - \sum_{j=1}^{m}\sum_{i\in S} w_j^c \bar{x}_j^L \\ &= \sum_{j=1}^{m} w_j^c N \bar{x}_j^L - \sum_{j=1}^{m} N w_j^c \bar{x}_j^L \\ &= 0.\end{aligned} \tag{37}$$

Using this in Eq. (36) gives the optimal value for b^c (Hastie et al., 2009):

$$b^c = \frac{1}{N}\sum_{i\in S} y_i. \tag{38}$$

We therefore center the input and output values of the linear regression as:

$$x_{i,j}^c = x_{i,j}^L - \bar{x}_j^L, \tag{39}$$

$$y_i^c = y_i - \frac{1}{N}\sum_{i\in S} y_i. \tag{40}$$

This gives the following Lagrange function:

$$\begin{aligned}\mathcal{L}\left(\lambda, b^c, w_j^c, j=1,2,\ldots,m\right) &= \mathcal{L}\left(\lambda, w_j^c, j=1,2,\ldots,m\right) \\ &= \sum_{i\in S}\left(y_i^c - \sum_{j=1}^{m} w_j^c x_{i,j}^c\right)^2 \\ &\quad + \lambda\left(\sum_{j=1}^{m}\left(w_j^c\right)^2 - \frac{1+m}{2}\right).\end{aligned} \tag{41}$$

In matrix form, this is:

$$\mathcal{L}\left(\lambda, \mathbf{w}^c\right) = \left(\mathbf{y}^c - \mathbf{x}^c\mathbf{w}^c\right)^T\left(\mathbf{y}^c - \mathbf{x}^c\mathbf{w}^c\right) + \lambda\left(\left(\mathbf{w}^c\right)^T\left(\mathbf{w}^c\right) - \frac{1+m}{2}\right), \tag{42}$$

with $\mathbf{y}^c$ a $N \times 1$ vector with the centered true label of all training samples in S, $\mathbf{w}^c$ a $m \times 1$ vector with the centered weights of the last layer L, and $\mathbf{x}^c$ a $N \times m$ matrix with the centered hidden states for each training sample and each input node.

A.0.2. Solution of the Lagrange function using the singular value decomposition

To solve the Lagrange function, we solve the system of equations:

$$\nabla_{\mathbf{w}^c}\mathcal{L}(\lambda, \mathbf{w}^c) = \mathbf{0}, \tag{43}$$

$$\nabla_{\lambda}\mathcal{L}(\lambda, \mathbf{w}^c) = 0, \tag{44}$$

where $\mathbf{0}$ is a $m \times 1$ vector with zeros. Given λ, we solve the first gradient $\nabla_{\mathbf{w}^c}\mathcal{L}(\lambda, \mathbf{w}^c) = \mathbf{0}$ with respect to $\mathbf{w}^c$. Solving this gradient gives the well-known solution of Ridge linear regression (Hastie et al., 2009):

$$\begin{aligned}&\nabla_{\mathbf{w}^c}\mathcal{L}(\lambda, \mathbf{w}^c) = \mathbf{0} \\ \Rightarrow &-2\left(\mathbf{x}^c\right)^T\left(\mathbf{y}^c - \mathbf{x}^c\mathbf{w}^c\right) + 2\lambda\mathbf{w}^c = \mathbf{0} \\ \Rightarrow &\mathbf{w}^c = \left(\left(\mathbf{x}^c\right)^T\mathbf{x}^c + \lambda\mathbf{I}\right)^{-1}\left(\mathbf{x}^c\right)^T\mathbf{y}^c,\end{aligned} \tag{45}$$

with $\mathbf{I}$ a $m \times m$ identity matrix.

We then use the singular value decomposition to solve the gradient with respect to λ as well. The singular value decomposition of $\mathbf{x}^c$ is (Poole, 2011):

$$\mathbf{x}^c = \mathbf{U}\mathbf{D}\mathbf{V}^T, \tag{46}$$

where $\mathbf{U}$ is an orthogonal $N \times N$ matrix (so $\mathbf{U}^{-1} = \mathbf{U}^T$) and $\mathbf{V}$ is an orthogonal $m \times m$ matrix (so $\mathbf{V}^{-1} = \mathbf{V}^T$). Moreover, $\mathbf{D}$ is a $N \times m$ "diagonal" matrix, with the singular values s of $\mathbf{x}^c$ on the diagonal (Poole, 2011). Using this decomposition, the optimal value for the centered weights w^c becomes (Hastie et al., 2009):

$$\begin{aligned}\mathbf{w}^c &= \left(\left(\mathbf{x}^c\right)^T\mathbf{x}^c + \lambda\mathbf{I}\right)^{-1}\left(\mathbf{x}^c\right)^T\mathbf{y}^c \\ &= \left(\mathbf{V}\mathbf{D}^T\mathbf{D}\mathbf{V}^T + \lambda\mathbf{V}\mathbf{V}^T\right)^{-1}\left(\mathbf{U}\mathbf{D}\mathbf{V}^T\right)^T\mathbf{y}^c \\ &= \left(\mathbf{V}\left(\mathbf{D}^T\mathbf{D} + \lambda\mathbf{I}\right)\mathbf{V}^T\right)^{-1}\mathbf{V}\mathbf{D}^T\mathbf{U}^T\mathbf{y}^c \\ &= \mathbf{V}\left(\mathbf{D}^T\mathbf{D} + \lambda I\right)^{-1}\mathbf{D}^T\mathbf{U}^T\mathbf{y}^c\end{aligned} \tag{47}$$

The sum of the centered weights $\left(\mathbf{w}^c\right)^T\mathbf{w}^c$ thus becomes:

$$\begin{aligned}\left(\mathbf{w}^c\right)^T\mathbf{w}^c &= \left(\mathbf{V}\left(\mathbf{D}^T\mathbf{D} + \lambda I\right)^{-1}\mathbf{D}^T\mathbf{U}^T\mathbf{y}^c\right)^T\mathbf{V}\left(\mathbf{D}^T\mathbf{D} + \lambda I\right)^{-1}\mathbf{D}^T\mathbf{U}^T\mathbf{y}^c \\ &= \left(\mathbf{y}^c\right)^T\mathbf{U}\mathbf{D}\left(\mathbf{D}^T\mathbf{D} + \lambda I\right)^{-2}\mathbf{D}^T\mathbf{U}^T\mathbf{y}^c.\end{aligned} \tag{48}$$

Let $\mathbf{b} = \mathbf{U}^T\mathbf{y}^c$, a vector of size $N \times 1$. Let p_i denote the ith element of a vector p, and let s_i be the ith singular value of $\mathbf{x}^c$. Then, it follows that:

$$\begin{aligned}\left(\mathbf{w}^c\right)^T\mathbf{w}^c &= \mathbf{b}^T\mathbf{D}\left(\mathbf{D}^T\mathbf{D} + \lambda I\right)^{-2}\mathbf{D}^T\mathbf{b} \\ &= \sum_{j=1}^{m}\frac{b_j^2 s_j^2}{(s_j^2 + \lambda)^2}.\end{aligned} \tag{49}$$

Now,

$$\sum_{j=1}^{m}\frac{b_j^2 s_j^2}{(s_j^2 + \lambda)^2} = \frac{1+m}{2}, \tag{50}$$

which can be solved numerically.

Appendix B. Solution of the minimization problem with a mean weight of zero (Eq. (21)) for λ

In this Appendix, we solve the constrained linear regression problem of Eq. (21) to find the optimal value of λ. The Lagrange function $\mathcal{L}(\lambda_1, \lambda_2, b^L, w_j^L, j = 1, 2, \ldots, m)$ of this problem is:

$$\mathcal{L}\left(\lambda_1, \lambda_2, b^L, w_j^L, j = 1, \ldots, m\right) = \sum_{i \in S} \left(y_i - b^L - \sum_{j=1}^{m} w_j^L x_{i,j}^L \right)^2 + \lambda_1 \left(\sum_{j=1}^{m} \left(w_j^L\right)^2 - \frac{1+m}{2} \right) + \lambda_2 \sum_{j=1}^{m} w_j^L. \tag{51}$$

We scale the hidden states and true labels in the same way as in Appendix A, Eq. (39) and (40):

$$\begin{aligned} &\mathcal{L}\left(\lambda_1, \lambda_2, b^L, w_j^L, j = 1, 2, \ldots, m\right) \\ &= \mathcal{L}\left(\lambda_1, \lambda_2, w_j^c, j = 1, 2, \ldots, m\right) \\ &= \sum_{i \in S} \left(y_i^c - \sum_{j=1}^{m} w_j^c x_{i,j}^c \right)^2 \\ &+ \lambda_1 \left(\sum_{j=1}^{m} \left(w_j^c\right)^2 - \frac{1+m}{2} \right) + \lambda_2 \sum_{j=1}^{m} w_j^c. \end{aligned} \tag{52}$$

In matrix form, this normalized Lagrange function becomes:

$$\mathcal{L}\left(\lambda_1, \lambda_2, \mathbf{w}^c\right) = \left(\mathbf{y}^c - \mathbf{x}^c \mathbf{w}^c\right)^T \left(\mathbf{y}^c - \mathbf{x}^c \mathbf{w}^c\right) + \lambda_1 \left(\left(\mathbf{w}^c\right)^T \mathbf{w}^c - \frac{1+m}{2} \right) + \lambda_2 \mathbf{1}^T \mathbf{w}^c, \tag{53}$$

with $\mathbf{1}$ a vector with ones of size $m \times 1$.

B.0.1. Solution of the Lagrange function with a mean weight of zero

To solve the Lagrange function, we solve the system of equations:

$$\nabla_{\mathbf{w}^c} \mathcal{L}\left(\lambda_1, \lambda_2, \mathbf{w}^c\right) = \mathbf{0}, \tag{54}$$

$$\nabla_{\lambda_1} \mathcal{L}\left(\lambda_1, \lambda_2, \mathbf{w}^c\right) = 0 \tag{55}$$

$$\nabla_{\lambda_2} \mathcal{L}\left(\lambda_1, \lambda_2, \mathbf{w}^c\right) = 0 \tag{56}$$

Given λ_1 and λ_2, we solve the first gradient $\nabla_{\mathbf{w}^c} \mathcal{L}(\lambda_1, \lambda_2, \mathbf{w}^c) = \mathbf{0}$ with respect to $\mathbf{w}^c$:

$$\begin{aligned} &\nabla_{\mathbf{w}^c} \mathcal{L}\left(\lambda_1, \lambda_2, \mathbf{w}^c\right) = \mathbf{0} \\ &\Rightarrow -2\left(\mathbf{x}^c\right)^T \left(\mathbf{y}^c - \mathbf{x}^c \mathbf{w}^c\right) + 2\lambda_1 \mathbf{w}^c + \lambda_2 \mathbf{1} = 0 \\ &\Rightarrow \mathbf{w}^c = \left(\left(\mathbf{x}^c\right)^T \mathbf{x}^c + \lambda_1 \mathbf{I} \right)^{-1} \left(\left(\mathbf{x}^c\right)^T \mathbf{y}^c - \frac{1}{2}\lambda_2 \mathbf{1} \right). \end{aligned} \tag{57}$$

We again use the singular value decomposition of $\mathbf{x}^c$ (Eq. (46)) to solve the Lagrange function for λ_1 and λ_2. First, we rewrite the optimal value of $\mathbf{w}^c$:

$$\begin{aligned} \mathbf{w}^c &= \left(\mathbf{V}\mathbf{D}^T\mathbf{D}\mathbf{V}^T + \lambda_1 \mathbf{V}\mathbf{V}^T\right)^{-1} \left(\left(\mathbf{U}\mathbf{D}\mathbf{V}^T\right)^T \mathbf{y}^c - \frac{1}{2}\lambda_2 \mathbf{1} \right) \\ &= \left(\mathbf{V} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-1} \mathbf{V}^T \right) \left(\mathbf{V}\mathbf{D}^T\mathbf{U}^T\mathbf{y}^c - \frac{1}{2}\lambda_2 \mathbf{1} \right) \\ &= \mathbf{V} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-1} \mathbf{D}^T\mathbf{U}^T\mathbf{y}^c - \frac{1}{2}\lambda_2 \mathbf{V} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-1} \mathbf{V}^T \mathbf{1}. \end{aligned} \tag{58}$$

We use this result to analyze $\nabla_{\lambda_2} \mathcal{L}\left(\lambda_1, \lambda_2, \mathbf{w}^c\right) = \mathbf{1}^T \mathbf{w}^c$ and to find the optimal value of λ_2. Let v_j^s be the sum of the jth column of $\mathbf{V}$. It then follows that:

$$\begin{aligned} \mathbf{1}^T \mathbf{w}^c &= \mathbf{1}^T \left(\mathbf{V} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-1} \mathbf{D}^T\mathbf{U}^T\mathbf{y}^c - \frac{1}{2}\lambda_2 \mathbf{V} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-1} \mathbf{V}^T \mathbf{1} \right) \\ &= \sum_{j=1}^{m} \frac{s_j v_j^s b_j}{s_j^2 + \lambda_1} - \frac{1}{2}\lambda_2 \sum_{j=1}^{m} \frac{(v_j^s)^2}{s_j^2 + \lambda_1}. \end{aligned}$$

This expression should equal zero (Eq. (55)):

$$\begin{aligned} &\sum_{j=1}^{m} \frac{s_j v_j^s b_j}{s_j^2 + \lambda_1} - \frac{1}{2}\lambda_2 \sum_{j=1}^{m} \frac{(v_j^s)^2}{s_j^2 + \lambda_1} = 0 \\ &\Rightarrow \frac{1}{2}\lambda_2 = \frac{\sum_{j=1}^{m} \frac{s_j v_j^s b_j}{s_j^2 + \lambda_1}}{\sum_{j=1}^{m} \frac{(v_j^s)^2}{s_j^2 + \lambda_1}}. \end{aligned}$$

Using this result, we now analyze $\nabla_{\lambda_1} \mathcal{L}\left(\lambda_1, \lambda_2, \mathbf{w}^c\right) = \left(\mathbf{w}^c\right)^T \mathbf{w}^c - \frac{1+m}{2}$. First, we derive that:

$$\begin{aligned} \left(\mathbf{w}^c\right)^T \cdot \mathbf{w}^c &= \Big(\left(\mathbf{y}^c\right)^T \mathbf{U}\mathbf{D} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-1} \mathbf{V}^T \\ &\quad - \frac{1}{2}\lambda_2 \mathbf{1}^T \mathbf{V} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-1} \mathbf{V}^T \Big) \\ &\quad \Big(\mathbf{V} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-1} \mathbf{D}^T\mathbf{U}^T\mathbf{y}^c \\ &\quad - \frac{1}{2}\lambda_2 \mathbf{V} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-1} \mathbf{V}^T \mathbf{1} \Big) \\ &= \left(\mathbf{y}^c\right)^T \mathbf{U}\mathbf{D} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-2} \mathbf{D}^T\mathbf{U}^T\mathbf{y}^c \\ &\quad - \lambda_2 \left(\mathbf{y}^c\right)^T \mathbf{U}\mathbf{D} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-2} \mathbf{V}^T \mathbf{1} \\ &\quad + \left(\frac{1}{2}\lambda_2\right)^2 \mathbf{1}^T \mathbf{V} \left(\mathbf{D}^T\mathbf{D} + \lambda_1 \mathbf{I}\right)^{-2} \mathbf{V}^T \mathbf{1} \\ &= \sum_{j=1}^{m} \frac{b_j^2 s_j^2}{\left(s_j^2 + \lambda_1\right)^2} - \lambda_2 \sum_{j=1}^{m} \frac{b_j s_j v_j^s}{\left(s_j^2 + \lambda_1\right)^2} \\ &\quad + \left(\frac{1}{2}\lambda_2\right)^2 \sum_{j=1}^{m} \frac{\left(v_j^s\right)^2}{\left(s_j^2 + \lambda_1\right)^2} \\ &= \sum_{j=1}^{m} \left(\frac{\frac{1}{2}\lambda_2 v_j^s - b_j s_j}{s_j^2 + \lambda_1} \right)^2. \end{aligned} \tag{59}$$

This expression should equal $\frac{1+m}{2}$, i.e.,:

$$\sum_{j=1}^{m} \left(\frac{\frac{1}{2}\lambda_2 v_j^s - b_j s_j}{s_j^2 + \lambda_1} \right)^2 = \frac{1+m}{2}. \tag{60}$$

Given the optimal value for $\frac{1}{2}\lambda_2$ in Eq. (59), we can solve this numerically to find the optimal value for λ_1. Using this, we can directly calculate the optimal value for λ_2 from Eq. (59).

Appendix C. Derivation of the constraints on the weights following Glorot and Bengio (2010)

In this Appendix, we derive the same constraints on the weights as in Section 2.2. However, we now follow the same derivation, with the same assumptions, as in Glorot and Bengio (2010).

Requirement 1 : $Var\left(\hat{y}\right) = Var\left(x^L\right)$

We first derive the variance of y^L in terms of the variance of x^L following Glorot and Bengio (2010). In contrast with our

derivation, where we regarded each weight as a constant number, we now regard each weight as a random variable. Let w^L represent the random variable of any element in $\mathbf{w}^L$. Four key assumptions are made in the derivation in Glorot and Bengio (2010): (i) the hidden states x^L are independent and identically distributed, (ii) the weights $w_j^L, j = 1, 2, \ldots, m$ are independent and identically distributed, (iii) the hidden states x^L are independent of the weights w^L, and (iiii) $\mathbb{E}[x^l] = 0$, due to the considered activation function. Note that only the first assumption is made in our derivation as well. Then, it holds that:

$$\begin{aligned}\mathrm{Var}\left(y^L\right) &= \mathrm{Var}\left(b^L + \sum_{j=1}^{m} w_j^L x_j^L\right) \\ &= m \cdot \mathrm{Var}\left(w^L x^L\right) \\ &= m\Big(\mathrm{Var}\left(w^L\right)\mathrm{Var}\left(x^L\right) + \mathbb{E}\left[w^L\right]^2 \mathrm{Var}\left(x^L\right) \\ &\quad + \mathrm{Var}\left(w^L\right)\mathbb{E}\left[x^L\right]^2\Big) \\ &= m\left(\left(\mathrm{Var}\left(w^L\right) + \mathbb{E}\left[w^L\right]^2\right)\mathrm{Var}\left(x^L\right)\right) \\ &= m\mathbb{E}\left[\left(w^L\right)^2\right]\mathrm{Var}\left(x^L\right).\end{aligned} \tag{61}$$

Note that we deviate here from Glorot and Bengio (2010), since we do not assume that the expected value of a weight is zero. This gives:

$$m\mathbb{E}\left[\left(w^L\right)^2\right] = 1. \tag{62}$$

We similarly derive that the sum of the squared weights equals 1 in Section 2.2.

Requirement 2 : $Var\left(\frac{\partial Loss}{\partial y^L}\right) = Var\left(\frac{\partial Loss}{\partial y^{L-1}}\right)$

In Section 2.2 we derived that:

$$\frac{\partial \mathrm{Loss}}{\partial \mathbf{y}_j^{L-1}} \approx \frac{\partial \mathrm{Loss}}{\partial \mathbf{y}^L} w_j^L.$$

We now additionally assume, following Glorot and Bengio (2010), that the weights of the last layer and the gradient $\frac{\partial \mathrm{Loss}}{\partial y^L}$ are independent of each other, and that the variance of $\frac{\partial \mathrm{Loss}}{\partial y_j^{L-1}}$ is the same for each node $j \in \{1, 2, \ldots, m\}$. This gives:

$$\begin{aligned}\mathrm{Var}\left(\frac{\partial \mathrm{Loss}}{\partial y^{L-1}}\right) &= \mathrm{Var}\left(\frac{\partial \mathrm{Loss}}{\partial y^L} w^L\right) \\ &= \mathrm{Var}\left(w^L\right)\mathrm{Var}\left(\frac{\partial \mathrm{Loss}}{\partial y^L}\right) + \mathbb{E}\left[w^L\right]^2 \mathrm{Var}\left(\frac{\partial \mathrm{Loss}}{\partial y^L}\right) \\ &\quad + \mathrm{Var}\left(w^L\right)\mathbb{E}\left[\frac{\partial \mathrm{Loss}}{\partial y^L}\right]^2 \\ &= \left(\mathrm{Var}\left(w^L\right) + \mathbb{E}\left[w^L\right]^2\right)\mathrm{Var}\left(\frac{\partial \mathrm{Loss}}{\partial y^L}\right) \\ &= \mathbb{E}\left[\left(w^L\right)^2\right]\mathrm{Var}\left(\frac{\partial \mathrm{Loss}}{\partial y^L}\right).\end{aligned} \tag{63}$$

It thus follows that:

$$\mathbb{E}\left[\left(w^L\right)^2\right] = 1. \tag{64}$$

We similarly derive that the sum of the squared weights is m in Section 2.2.

Appendix D. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.neunet.2023.07.035.

References

Adam, S. P., Karras, D. A., Magoulas, G. D., & Vrahatis, M. N. (2014). Solving the linear interval tolerance problem for weight initialization of neural networks. *Neural Networks*, *54*, 17–37.

Aguirre, D., & Fuentes, O. (2019). Improving weight initialization of ReLU and output layers. In *International conference on artificial neural networks* (pp. 170–184). Springer.

Cao, W., Wang, X., Ming, Z., & Gao, J. (2018). A review on neural networks with random weights. *Neurocomputing*, *275*, 278–287.

Chumachenko, K., Iosifidis, A., & Gabbouj, M. (2022). Feedforward neural networks initialization based on discriminant learning. *Neural Networks*, *146*, 220–229.

Cubuk, E. D., Zoph, B., Shlens, J., & Le, Q. V. (2020). Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops* (pp. 702–703).

de Pater, I., & Mitici, M. (2022). Novel metrics to evaluate probabilistic remaining useful life prognostics with applications to turbofan engines. In *PHM society European conference, vol. 7* (pp. 96–109).

de Pater, I., Reijns, A., & Mitici, M. (2022). Alarm-based predictive maintenance scheduling for aircraft engines with imperfect remaining useful life prognostics. *Reliability Engineering & System Safety*, *221*, Article 108341.

Fernández-Navarro, F., Riccardi, A., & Carloni, S. (2014). Ordinal neural networks without iterative tuning. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(11), 2075–2085.

Frühwirth-Schnatter, S., & Frèuhwirth-Schnatter, S. (2006). *Finite mixture and markov switching models, vol. 425*. Springer.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256). JMLR Workshop and Conference Proceedings.

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction, vol. 2*. Springer.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026–1034).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Heij, C., Heij, C., de Boer, P., Franses, P. H., Kloek, T., van Dijk, H. K., et al. (2004). *Econometric methods with applications in business and economics*. Oxford University Press.

Kim, M. (2021). The generalized extreme learning machines: Tuning hyperparameters and limiting approach for the Moore–Penrose generalized inverse. *Neural Networks*, *144*, 591–602.

Kim, S., Lu, P. Y., Mukherjee, S., Gilbert, M., Jing, L., Čeperić, V., et al. (2020). Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(9), 4166–4177.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Krizhevsky, A., Hinton, G., et al. (2009). *Learning multiple layers of features from tiny images*. Toronto, ON, Canada.

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade* (pp. 9–48). Springer.

Lee, J., & Mitici, M. (2023). Deep reinforcement learning for predictive aircraft maintenance using probabilistic Remaining-Useful-Life prognostics. *Reliability Engineering & System Safety*, *230*, Article 108908.

Li, X., Ding, Q., & Sun, J.-Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering & System Safety*, *172*, 1–11.

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2021). A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*.

Martínez, F., Charte, F., Frías, M. P., & Martínez-Rodríguez, A. M. (2022). Strategies for time series forecasting with generalized regression neural networks. *Neurocomputing*, *491*, 509–521.

Mishkin, D., & Matas, J. (2015). All you need is a good init. arXiv preprint arXiv:1511.06422.

Narkhede, M. V., Bartakke, P. P., & Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. *Artificial Intelligence Review*, *55*(1), 291–322.

Pan, X., Xu, J., Pan, Y., Wen, L., Lin, W., Bai, K., et al. (2022). Afinet: Attentive feature integration networks for image classification. *Neural Networks*, *155*, 360–368.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Poole, D. (2011). *Linear algebra: A modern introduction* (3rd ed.). Cengage Learning.

Ramasso, E., & Saxena, A. (2014). Review and analysis of algorithmic approaches developed for prognostics on CMAPSS dataset. In *Annual conference of the prognostics and health management society 2014*.

Roh, J., Park, S., Kim, B.-K., Oh, S.-H., & Lee, S.-Y. (2021). Unsupervised multi-sense language models for natural language processing tasks. *Neural Networks*, *142*, 397–409.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252. http://dx.doi.org/10.1007/s11263-015-0816-y.

Saxe, A. M., McClelland, J. L., & Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.6120.

Saxena, A., & Goebel, K. (2008). Turbofan engine degradation simulation data set. (pp. 878–887). NASA Ames Prognostics Data Repository (https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/).

Vasilev, I. (2019). *Advanced deep learning with Python: Design and implement advanced next-generation AI solutions using tensorflow and PyTorch*. Packt Publishing Ltd.

Vollert, S., & Theissler, A. (2021). Challenges of machine learning-based RUL prognosis: A review on NASA's C-MAPSS data set. In *2021 26th IEEE international conference on emerging technologies and factory automation* (pp. 1–8). IEEE.

Vural, N. M., Ilhan, F., Yilmaz, S. F., Ergüt, S., & Kozat, S. S. (2021). Achieving online regression performance of LSTMs with simple RNNs. *IEEE Transactions on Neural Networks and Learning Systems*.

Xie, X., Pu, Y.-F., & Wang, J. (2023). A fractional gradient descent algorithm robust to the initial weights of multilayer perceptron. *Neural Networks*, *158*, 154–170.

Yam, Y.-F., & Chow, T. W. (1995). Determining initial weights of feedforward neural networks based on least squares method. *Neural Processing Letters*, *2*(2), 13–17.

Yam, Y., Chow, T. W., & Leung, C.-T. (1997). A new method in determining initial weights of feedforward neural networks for training enhancement. *Neurocomputing*, *16*(1), 23–32.

Yan, W. (2012). Toward automatic time-series forecasting using neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *23*(7), 1028–1039.

Yilmaz, A., & Poli, R. (2022). Successfully and efficiently training deep multi-layer perceptrons with logistic activation function simply requires initializing the weights with an appropriate negative mean. *Neural Networks*.

Zhao, Z.-Q., Zheng, P., Xu, S.-t., & Wu, X. (2019). Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(11), 3212–3232.