Remember to correct the bias when using deep learning for regression!

Anonymous authors
Paper under double-blind review

Abstract

When training deep learning models for least-squares regression, we cannot expect that the training error residuals of the final model, selected after a fixed training time or based on performance on a hold-out data set, sum to zero. This can introduce a systematic error that accumulates if we are interested in the total aggregated performance over many data points. We suggest adjusting the bias of the machine learning model after training as a default post-processing step, which efficiently solves the problem. The severeness of the error accumulation and the effectiveness of the bias correction are demonstrated in exemplary experiments.

1 Problem statement

We consider regression models $f : \mathbb{X} \to \mathbb{R}^d$ of the form

$$f_{\boldsymbol{\theta}}(x) = \boldsymbol{a}^{\mathrm{T}} h_{\boldsymbol{w}}(x) + b \tag{1}$$

with parameters $\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{a}, b)$ and $x \in \mathbb{X}$. Here $\mathbb{X}$ is some arbitrary input space and w.l.o.g. we assume $d = 1$. The function $h_{\boldsymbol{w}} : \mathbb{X} \to \mathbb{R}^F$ is parameterized by $\boldsymbol{w}$ and maps the input to an F-dimensional real-valued feature representation, $\boldsymbol{a} \in \mathbb{R}^F$, and b is a scalar. If $\mathbb{X}$ is a Euclidean space and h the identity, this reduces to standard linear regression. However, we are more interested in the case where $h_{\boldsymbol{w}}$ is more complex. In particular,

- $f_{\boldsymbol{\theta}}$ can be a deep neural network, where $\boldsymbol{a}$ and b are the parameters of the final output layer and $h_{\boldsymbol{w}}$ represents all other layers (e.g., a convolutional or point cloud architecture);
- $h : \mathcal{X} \mapsto \mathbb{R}$ can be any regression model (e.g., a random forest or deep neural network) and $f_{\boldsymbol{\theta}}$ denotes $h_{\boldsymbol{w}}$ with an additional wrapper, where $a = 1$ and initially $b = 0$.

In the following, we call b the distinct bias parameter of our model (although $\boldsymbol{w}$ may comprise many parameters typically referred to as bias parameters if $h_{\boldsymbol{w}}$ is a neural network). Given some training data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ drawn from a distribution p_{data} over $\mathbb{X} \times \mathbb{R}$, we assume that the model parameters $\boldsymbol{\theta}$ are determined by minimizing the mean-squared-error (MSE)

$$\mathcal{R}_{\mathcal{D}}(f_{\boldsymbol{\theta}}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f_{\boldsymbol{\theta}}(x_i))^2 \ , \tag{2}$$

potentially combined with some form of regularization. Typically, the goal is to achieve a low risk $\mathcal{R}(f_{\boldsymbol{\theta}}) = \mathbb{E}_{(x,y)\sim p_{\text{data}}}[(y - f_{\boldsymbol{\theta}}(x))^2] = \mathbb{E}\mathcal{R}_{\mathcal{D}}(f_{\boldsymbol{\theta}})$, where the second expectation is over all test data sets drawn i.i.d. based on p_{data}. However, here we are mainly concerned with applications where the (expected) *absolute total error* defined as the absolute value of the sum of residuals

$$\Delta_{\mathcal{D}}(f_{\boldsymbol{\theta}}) = \left| \sum_{(x,y)\in\mathcal{D}} (y - f_{\boldsymbol{\theta}}(x)) \right| \tag{3}$$

is of high importance. The *relative total error* is then given by

$$\delta_{\mathcal{D}}(f_{\boldsymbol{\theta}}) = \frac{\Delta_{\mathcal{D}}(f_{\boldsymbol{\theta}})}{\left|\sum_{(x,y)\in\mathcal{D}} y\right|} \ , \tag{4}$$

which is closely related to the bias or *relative systematic error* $\frac{100}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}} \frac{y-f_{\boldsymbol{\theta}}(x)}{y}$ (in %, e.g., Jucker et al., 2017).

That is, we are interested in how well $\sum_{(x,y)\in\mathcal{D}_{\text{test}}} f_{\boldsymbol{\theta}}(x)$ approximates $\sum_{(x,y)\in\mathcal{D}_{\text{test}}} y$ for a test set $\mathcal{D}_{\text{test}}$. For $|\mathcal{D}_{\text{test}}| \to \infty$ a constant model predicting $\hat{y} = \mathbb{E}_{(x,y)\sim p_{\text{data}}}[y]$ would minimize $\Delta_{\mathcal{D}_{\text{test}}}(f_{\boldsymbol{\theta}})/|\mathcal{D}_{\text{test}}|$. However, in practice $\frac{1}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}} y$ and $\frac{1}{|\mathcal{D}_{\text{test}}|}\sum_{(x,y)\in\mathcal{D}_{\text{test}}} y$ can be considerably different from each other and from $\hat{y}$ because of finite sample effects and violations of the i.i.d. assumption (e.g., due to covariate shift or sample selection bias), so optimization of individual predictions (e.g., minimizing equation 2) is preferred.

Our study is motivated by applications in large-scale ecosystem monitoring such as convolutional neural network-based systems estimating tree canopy area from satellite imagery (Brandt et al., 2020) applied for assessing the total tree canopy cover of a country and learning systems trained on small patches of 3D point clouds to predict the biomass (and thus stored carbon) of large forests (Jucker et al., 2017; Oehmcke et al., 2021). However, there are many other application areas, such as estimating the overall performance of a portfolio based on estimates of the performance of the individual assets or overall demand forecasting based on forecasts for individual consumers.

At a first glance, it seems that optimizing $\mathbb{E}\mathcal{R}_{\mathcal{D}}(f_{\boldsymbol{\theta}})$ guarantees low $\mathbb{E}\Delta_{\mathcal{D}}(f_{\boldsymbol{\theta}})$, where the expectations are agian with respect to data sets drawn i.i.d. based on p_{data}. Obviously, $\mathcal{R}_{\mathcal{D}}(f_{\boldsymbol{\theta}}) = 0$ implies $\Delta_{\mathcal{D}}(f_{\boldsymbol{\theta}}) = 0$. More general, the optimal parameters $\boldsymbol{\theta}^*$ minimizing $\mathcal{R}_{\mathcal{D}}(f_{\boldsymbol{\theta}})$ result in $\Delta_{\mathcal{D}}(f_{\boldsymbol{\theta}^*}) = 0$. Actually, for all parameters $\boldsymbol{\theta}_{b^*} = (\boldsymbol{w}, \boldsymbol{a}, b^*)$, where b^* is the optimal bias parameter minimizing $\mathcal{R}_S(f_{\boldsymbol{\theta}})$ given $\boldsymbol{w}$ and $\boldsymbol{a}$, it is well known that the error residuals sum to zero and thus $\Delta_{\mathcal{D}}(f_{\boldsymbol{\theta}_{b^*}}) = 0$. This can directly be seen from equation 6 below. However, when b is not optimal in the sense above, a low $\mathcal{R}_{\mathcal{D}}(f_{\boldsymbol{\theta}})$ may not imply a low $\Delta_{\mathcal{D}}(f_{\boldsymbol{\theta}})$. In fact, if we are ultimately interested in the total aggregated performance over many data points, a wrongly adjusted parameter b may lead to significant systematic errors. Assume that f^* is the Bayes optimal model for a given task and that f_δ is the model where the optimal bias parameter b^* is replaced by $b^* - \delta_b$. Then for a test set $\mathcal{D}_{\text{test}}$ of cardinality N_{test} we have

$$\sum_{(x,y)\in\mathcal{D}_{\text{test}}} (y - f_{\delta_b}(x)) = N_{\text{test}} \cdot \delta_b + \sum_{(x,y)\in\mathcal{D}_{\text{test}}} (y - f^*(x)) \ . \tag{5}$$

That is, the errors δ_b accumulate. While one typically hopes that errors partly cancel out when applying a model to a lot of data points, the aggregated error due to a badly chose bias parameter increases. This can be a severe problem when using deep learning for regression, because in the canonical training process of a neural network for regression minimizing the (regularized) MSE the parameter b of the final model cannot be expected to be optimal:

- Large deep learning systems are typically not trained until the parameters are close to their optimal values, because this is not necessary to achieve the desired performance in terms of MSE and/or training would take too long.
- The final weight configuration is often picked based on the performance on a validation data set, not depending on how close the parameters are to their optimal values (e.g., Prechelt, 2012).
- Mini-batch learning introduces a random effect in the parameter updates, and therefore in the bias parameter value in the finally chosen network.

Thus, despite low MSE, the performance of a (deep) learning system in terms of the total error as defined in equation 3 can get arbitrarily bad. For example, in the tree canopy estimation task described above, you may get a decently accurate biomass estimate for individual trees, but the prediction over a large area (i.e., the quantity you are actually interested in) could be very wrong.

Therefore, we propose to adjust the bias parameter after training a machine learning model for least-squares regression as a default post-processing step. In the next section, we show how to simply compute this correction and discuss the consequences. Section 3 presents experiments demonstrating the problem and the effectiveness of the proposed solution.

2 Solution: Adjusting the bias

Given fixed parameters $\boldsymbol{w}$ and $\boldsymbol{a}$, for an optimally adjusted bias parameter b^* on $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ the first derivative w.r.t. b vanishes:

$$\left.\frac{\partial \mathcal{R}_{\mathcal{D}}(f_{\boldsymbol{\theta}})}{\partial b}\right|_{b=b^*} = \sum_{i=1}^{N} (y_i - \boldsymbol{a}^{\mathrm{T}} h_{\boldsymbol{w}}(x_i) - b^*) = 0 \tag{6}$$

Thus, for fixed $\boldsymbol{w}$ and $\boldsymbol{a}$ we can simply solve for the optimal bias parameter:

$$b^* = \frac{\sum_{i=1}^{N}(y_i - \boldsymbol{a}^{\mathrm{T}} h_{\boldsymbol{w}}(x_i))}{N} = \frac{\sum_{i=1}^{N} y_i - \sum_{i=1}^{N} \boldsymbol{a}^{\mathrm{T}} h_{\boldsymbol{w}}(x_i)}{N} = \underbrace{\frac{\sum_{i=1}^{N} y_i - \sum_{i=1}^{N} f_{\boldsymbol{\theta}}(x_i)}{N}}_{\delta_b} + b \tag{7}$$

In practice, we can either replace b in our trained model by b^* or add δ_b to all model predictions. The costs of computing b^* and δ_b are the same as computing the error on the data set used for adjusting the bias.

The trivial consequences of this adjustment are:

1. The MSE on the training data set is reduced.
2. The residuals on the training set cancel each other.

But what happens on unseen data? Adjusting the single scalar parameter b based on a lot of data is very unlikely to lead to overfitting. On the contrary, in practice we are typically observing a reduced MSE on external test data after adjusting the bias. However, this effect is typically minor. The weights of the neural network and in particular the bias parameter in the final linear layer are learned sufficiently well so that the MSE is not significantly degraded because the single bias parameter is not adjusted optimally – and that is why one typically does not worry about it although the effect on the absolute total error may be drastic.

Which data should be used to adjust the bias? While one could use an additional hold-out set for the final optimization of b, this is not necessary. Data already used in the model design process can be used, because assuming a sufficient amount of data selecting a single parameter is unlikely to lead to overfitting. If there is a validation data set (e.g., for early-stopping), then these data could be used. However, we recommend to simply use all data available for model building (e.g., the union of training and validation set). If data augmentation is used, augmented data sets could be considered.

How to deal with regularization? So far, we just considered empirical risk minimization. However, the bias parameter can adjusted regardless of how the model was obtained. This includes the use of early-stopping (Prechelt, 2012) or regularized risk minimization with an objective of the form $\frac{1}{N}\sum_{i=1}^{N}(y_i - f_{\boldsymbol{\theta}}(x_i))^2 + \Omega(\boldsymbol{\theta})$. Here, Ω denotes some regularization depending on the parameters. This includes weight-decay, however, typically this type of regularization would not consider the bias parameter b of a regression model anyway (e.g., Bishop, 1995, p. 342).

3 Examples

In this section, we present experiments that illustrate the problem of a large total error despite a low MSE and show that adjusting the bias as proposed above is a viable solution. We start with a simple regression task based on a UCI benchmark data set (Dua & Graff, 2017), which is easy to reproduce (see supplementary material). Then we move closer to real-world applications and consider convolutional neural networks for ecosystem monitoring.

Table 1: Results for the total CO emissions prediction tasks for the different models, where "linear" refers to linear regression, "not corrected" to a neural network without bias correction, and "corrected" to the same neural network with corrected bias parameter. The results are based on 10 trials. The mean and standard error (SE) are given; values are rounded to two decimals; R^2, Δ and δ denote the coefficient of determinations, the absolute total error, and the relative error; δ is given in percent; $\mathcal{D}$ and $\mathcal{D}_{\text{test}}$ are all data available for model development and testing, respectively.

MODEL	$R^2_{\mathcal{D}}$	$R^2_{\mathcal{D}_{\text{test}}}$	$\Delta_{\mathcal{D}}$	$\Delta_{\mathcal{D}_{\text{test}}}$	$\delta_{\mathcal{D}_{\text{test}}}$
linear	0.56 ±0.0	0.57 ±0.0	0 ± 0	173 ±14	0.49 ±0.04
not corrected	0.78 ±0.0	0.72 ±0.0	1018 ±70	785 ±53	2.21 ±0.15
corrected	0.78 ±0.0	0.72 ±0.0	0 ± 0	122 ± 6	0.34 ±0.02

3.1 Gas turbine emission prediction

First, we look at an artificial example based on real-world data from the UCI benchmark repository (Dua & Graff, 2017), which is easy to reproduce. We consider the *Gas Turbine CO and NOx Emission Data Set* (Kaya et al., 2019), where each data point corresponds to CO and NOx (NO and NO_2) emissions and 11 aggregated sensor measurements from a gas turbine summarized over one hour. The typical tasks are to predict the hourly emissions given the sensor measurements. Here we consider the fictitious task of predicting the total amount of CO emissions for a set of measurements.

Experimental setup. There are 36 733 data points in total. We assumed that we know the emissions for $N_{\text{train}} = 21\,733$ randomly selected data points, which we used to build our models.

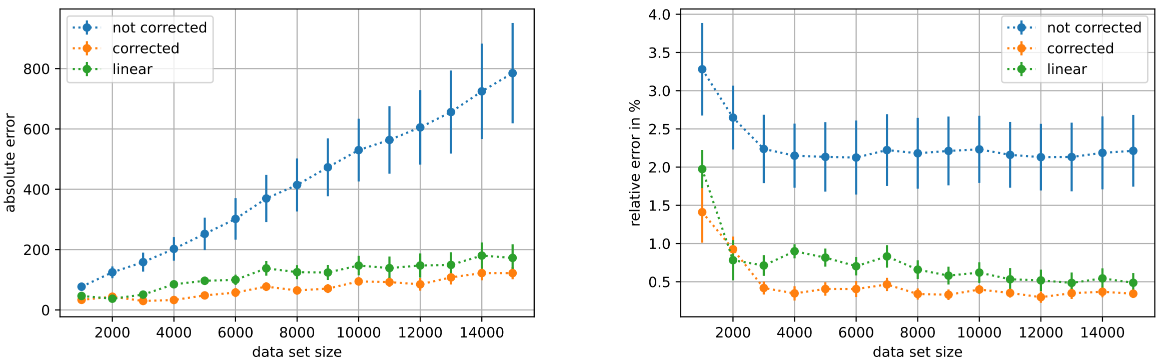
We trained a neural network with two hidden layers with sigmoid activation functions having 16 and 8 neurons, respectively, feeding into a linear output layer. There were shortcut connections from the inputs to the output layer. We randomly split the training data into 16 733 examples for gradient computation and 5000 examples for validation. The network was trained for 1000 epochs using Adam (Kinga & Ba, 2015) with a learning rate of $1 \cdot 10^{-2}$ and mini-batches of size 64. The network with the lowest error on the validation data was selected. For adjusting the bias parameter, we computed δ_b using equation 7 and all N_{train} data points available for model development. As a baseline, we fitted a linear regression model using all N_{train} data points.

We used Scikit-learn (Pedregosa et al., 2011) and PyTorch (Paszke et al., 2019) in our experiments. The input data were converted to 32-bit floating point precision. We repeated the experiments 10 times with 10 random data splits, network initializations, and mini-batch shufflings.

Results. The results are shown in Table 1 and Figure 1. The neural networks without bias correction achieved a higher R^2 (coefficient of determination) than the linear regression on the training and test data, see Table 1. On the test data, the R^2 averaged over the ten trials increased from 0.56 to 0.78 when using the neural network. However, the $\Delta_{\mathcal{D}}$ and $\Delta_{\mathcal{D}_{\text{test}}}$ were much lower for linear regression. This shows that a better MSE does not directly translate to a better total error (sum of residuals).

Correcting the bias of the neural network did not change the networks' R^2, however, the total errors went down to the same level as for linear regression and even below. Thus, correcting the bias gave the best of both world, a low MSE for individual data points and a low accumulated error.

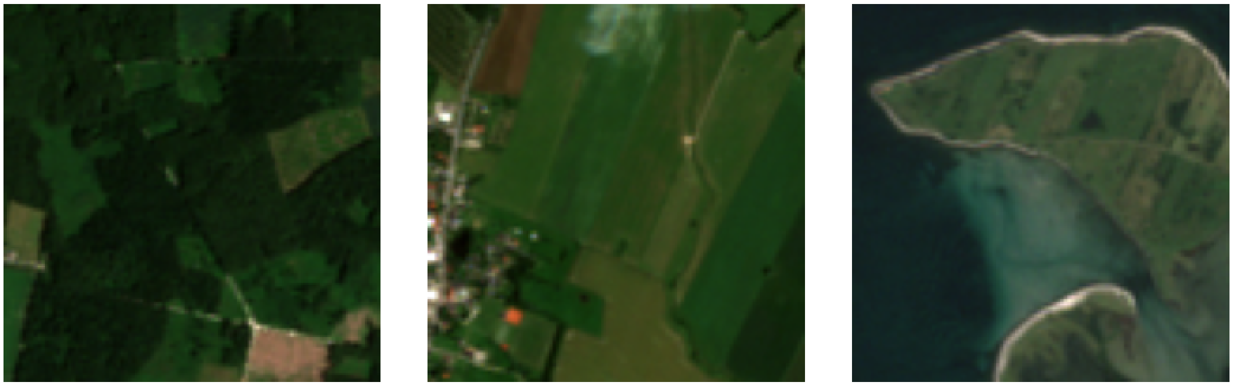
Figure 1 demonstrates how the total error developed as a function of test data set size. As predicted, with a badly adjusted bias parameter the total error increased with the number of test data points, while for the linear models and the neural network with adjusted bias this negative effect was less pronounced. The linear models performed worse than the neural networks with adjusted bias parameters, which can be explained by the worse accuracy of the individual predictions.



(a) Absolute total error CO

(b) Relative total error CO

Figure 1: Absolute errors (absolute value of the summed residuals) are shown on the left and the relative errors on the right for the CO emission prediction task. Both are are presented in relation to the test set size. The error bars show the standard error (SE). Here "linear" refers to linear regression, "not corrected" to a neural network without bias correction, and "corrected" to the same neural network with corrected bias parameter. The results are averaged over 10 trials, the error bars show the standard error (SE).



(a) $y = 7567$

(b) $y = 247$

(c) $y = 11$

Figure 2: Exemplary inputs and targets (y) for the forest coverage dataset. (a) shows a scene with 75.7%, (b) with 2.5%, and (c) with 0.1% forest.

Table 2: Results of forest coverage prediction, R^2 and Δ denote the coefficient of determinations and absolute total error; $\mathcal{D}$ and $\mathcal{D}_{\text{test}}$ are all data available for model development and testing, respectively. The relative total error δ is given in percent. Average and standard error (SE) of these metrics are given over 10 trials for the different models, where "mean" refers to predicting the constant mean of the training set, "not corrected" to EfficientNet-B0 without bias correction, and "corrected" to the same neural network with corrected bias parameter. Values are rounded to three decimals.

MODEL	$R^2_{\mathcal{D}}$	$R^2_{\mathcal{D}_{\text{test}}}$	$\Delta_{\mathcal{D}}$	$\Delta_{\mathcal{D}_{\text{test}}}$	$\delta_{\mathcal{D}_{\text{test}}}$
mean	0.000 ±0.000	$-3 \cdot 10^{-5}$ ±0.0	6.4 ± 2	169 665.8 ±48 944	0.955 ±0.272
not corrected	0.992 ±0.027	0.977 ±0.0	389 747.2 ±77 987	152 666.2 ±22 164	0.864 ±0.124
corrected	0.992 ±0.027	0.977 ±0.0	3.2 ± 1	59 818.8 ±10 501	0.338 ±0.059

3.2 Forest Coverage

Deep learning holds great promise for large-scale ecosystem monitoring (Persello et al., 2022; Yuan et al., 2020), for example for estimating tree canopy cover and forest biomass from remote sensing data (Brandt et al., 2020; Oehmcke et al., 2021). Here we consider a simplified task where the goal is to predict the amount of pixels in an image that belong to forests given a satellite image. We generated the input data from Sentinel 2 measurements (RGB values) and the accumulated pixels from a landcover map[1] as targets, see Figure 2 for examples. Both, input and target are at the same 10 m spatial resolution, collected/estimated in 2017, and cover the country of Denmark. Each sample is a 100×100 large image with no overlap between images.

Experimental setup. From the 127 643 data points in total, 70% (89 350) were used for training, 10% (12 764) for validation and 20% (25 529) for testing. For each of the 10 trials a different random split of the data was considered.

We employed the EfficientNet-B0 (Tan & Le, 2019), a deep convolutional network that uses mobile inverted bottleneck MBConv (Tan et al., 2019) and squeeze-and-excitation (Hu et al., 2018) blocks. It was trained for 300 epochs with Adam and a batch size of 256. For 100 epochs the learning rate was set to $3 \cdot 10^{-4}$ and thereafter reduced to $1 \cdot 10^{-5}$. The validation set was used to select the best model w.r.t. R^2. When correcting the bias, the training and validation set were combined. We considered the constant model predicting the mean of the training targets as a baseline.

Results. The results are summarized in Figure 3 and Table 2. The bias correction did not yield a better R^2 result, with 0.992 on the training set and 0.977 on the test set. However, $\Delta_{\mathcal{D}_{\text{test}}}$ on the test set decreased by a factor of 2.6 from 152 666 to 59 819. The R^2 for the mean prediction is by definition 0 on the training set and was close to 0 on the test set, yet $\Delta_{\mathcal{D}_{\text{test}}}$ is 169 666, meaning that a shift in the distribution center occurred, rendering the mean prediction unreliable.

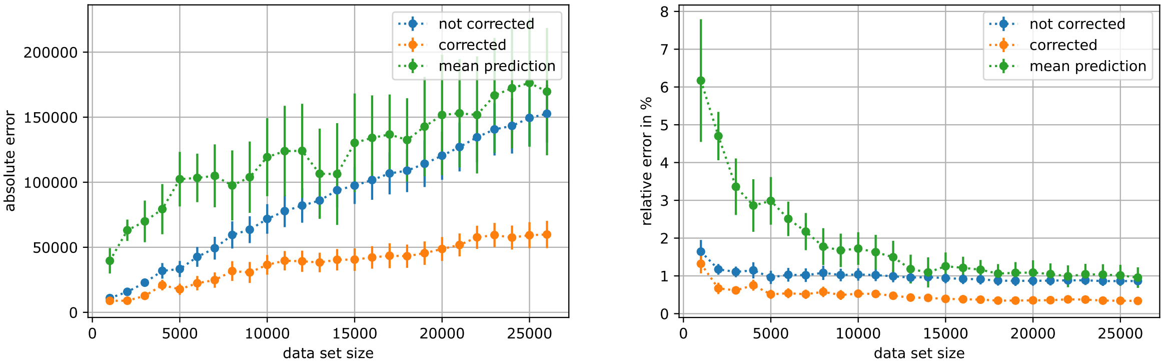
In Figure 3, we show $\Delta_{\mathcal{D}_{\text{test}}}$ and $\delta_{\mathcal{D}_{\text{test}}}$ while increasing the test set size. As expected, the total absolute error of the uncorrected neural networks increases with increasing number of test data points. Simply predicting the mean gave similar results in terms of the accumulated errors compared to the uncorrected model, which shows how misleading the R^2 can be as an indicator how well regression models perform in terms of the accumulated total error. When the bias was corrected, this effect drastically decreased and the performance was better compared to mean prediction.

4 Conclusions

Adjusting the bias such that the residuals sum to zero should be the default post-processing step when doing least-squares regression using deep learning. It comes at the cost of at most a single forward propagation

---

[1] `https://www.esa.int/Applications/Observing_the_Earth/Copernicus/Sentinel-2/Land-cover_maps_of_Europe_from_the_Cloud`



(a) Absolute total error

(b) Relative total error

Figure 3: The absolute errors (absolute value of the summed residuals) are shown on the left and the relative errors on the right for the forest coverage prediction task. Both are are presented in relation to the test set size. The error bars show the standard error (SE). Results were averaged over 10 trials and show the different models, where "mean" refers to predicting the constant mean of the training set, "not corrected" to EfficientNet-B0 without bias correction, and "corrected" to the same neural network with corrected bias parameter.

of the training and/or validation data set, but removes a systematic error that accumulates if individual predictions are summed.

References

Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

Martin Brandt, Compton J. Tucker, Ankit Kariryaa, Kjeld Rasmussen, Christin Abel, Jennifer Small, Jerome Chave, Laura Vang Rasmussen, Pierre Hiernaux, Abdoul Aziz Diouf, Laurent Kergoat, Ole Mertz, Christian Igel, Fabian Gieseke, Johannes Schöning, Sizhuo Li, Katherine Melocik, Jesse Meyer, Scott Sinno, Eric Romero, Erin Glennie, Amandine Montagu, Morgane Dendoncker, and Rasmus Fensholt. An unexpectedly large count of trees in the western Sahara and Sahel. *Nature*, 587:78–82, 2020.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7132–7141, 2018.

Tommaso Jucker, John Caspersen, Jérôme Chave, Cécile Antin, Nicolas Barbier, Frans Bongers, Michele Dalponte, Karin Y van Ewijk, David I Forrester, Matthias Haeni, Steven I. Higgins, Robert J. Holdaway, Zoshiko Iida, Craig Lorime, Peter L. Marshall, Stéphane Momo, Glenn R. Moncrieff, Pierre Ploton, Lourens Poorter, Kassim Abd Rahman, Michael Schlund, Bonaventure Sonké, Frank J. Sterck, Anna T. Trugman, Vladimir A. Usoltsev, Mark C. Vanderwel, Peter Waldner, Beatrice M. M. Wedeux, Christian Wirth, Hannsjörg Wöll, Murray Woods, Wenhua Xiang, Niklaus E. Zimmermann, and David A. Coomes. Allometric equations for integrating remote sensing imagery into forest monitoring programmes. *Global Change Biology*, 23(1):177–190, 2017.

Heysem Kaya, Pinar Tüfekci, and Erdinç Uzun. Predicting CO and NOx emissions from gas turbines: novel data and a benchmark PEMS. *Turkish Journal of Electrical Engineering & Computer Sciences*, 27(6): 4783–4796, 2019.

Diederik P. Kinga and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Stefan Oehmcke, Lei Li, Jaime Revenga, Thomas Nord-Larsen, Katerina Trepekli, Fabian Gieseke, and Christian Igel. Deep learning based 3D point cloud regression for estimating forest biomass. *CoRR*, abs/2112.11335, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In Hanna Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8024–8035. 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

Claudio Persello, Jan Dirk Wegner, Ronny Hansch, Devis Tuia, Pedram Ghamisi, Mila Koeva, and Gustau Camps-Valls. Deep learning and earth observation to support the sustainable development goals: Current approaches, open challenges, and future opportunities. *IEEE Geoscience and Remote Sensing Magazine*, pp. 2–30, 2022.

Lutz Prechelt. Early stopping — but when? In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller (eds.), *Neural Networks: Tricks of the Trade: Second Edition*, pp. 53–67. Springer, 2012.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*, pp. 6105–6114. PMLR, 2019.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2820–2828, 2019.

Qiangqiang Yuan, Huanfeng Shen, Tongwen Li, Zhiwei Li, Shuwen Li, Yun Jiang, Hongzhang Xu, Weiwei Tan, Qianqian Yang, Jiwen Wang, Jianhao Gao, and Liangpei Zhang. Deep learning in environmental remote sensing: Achievements and challenges. *Remote Sensing of Environment*, 241:111716, 2020.