Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 58-15 (2024) 241–246

Split-Boost Neural Networks

Raffaele G. Cestari * **Gabriele Maroni** ** **Loris Cannelli** **
Dario Piga ** **Simone Formentin** *

* *Department of Electronics, Information, and Bioengineering, Politecnico di Milano, Milano, Italy.*
** *SUPSI-DTI-IDSIA, Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland.*

Abstract: The calibration and training of a neural network is a complex and time-consuming procedure that requires significant computational resources to achieve satisfactory results. Key obstacles are a large number of hyperparameters to select and the onset of overfitting in the face of a small amount of data. In this framework, we propose an innovative training strategy for feed-forward architectures - called *split-boost* - that improves performance and automatically includes a regularizing behaviour without modeling it explicitly. Such a novel approach ultimately allows us to avoid explicitly modeling the regularization term, decreasing the total number of hyperparameters and speeding up the tuning phase. The proposed strategy is tested on a real-world (anonymized) dataset within a benchmark medical insurance design problem.

Keywords: Artificial intelligence, deep learning, machine learning, neural networks, hyperparameter tuning, regularization

1. INTRODUCTION

Training a neural network is a complex task for several reasons. The high dimensionality of the search space in which the network parameters live (number of layers, number of neurons per layer, learning rate, choice of activation function, batch size, and regularization factor) makes the problem difficult to optimize. The optimization problem for network parameters update is non-convex and must be solved with gradient descent strategies that might suffer from the presence of local minima and saddle points that slow down the convergence to global optima. Very often it happens that the choice of a hyperparameter affects the value that the others must assume and the change of one of them can lead to unexpected changes in the behavior of the network. In addition, since training a neural network is computationally expensive (even if the performance in terms of Graphic Processing Unit (GPU) computing power is increasingly reducing training times, Krizhevsky et al. (2017)) hyperparameter space exploration might be a slow and complex task. Lastly, although there are guidelines for selecting hyperparameters, in general, there is not one specific solution to the tuning problem and the quality of the selection might depend on the specific case, considering the dataset and network architecture. Given these major challenges, we propose an innovative feed-forward neural network training strategy. The goal is twofold:

- removing the calibration step of the regularization term, otherwise indispensable to avoid the onset of overfitting, facilitating the training procedure of the network;
- include regularization *automatically* in the training process exploiting different training data. The goal is to achieve a regularization effect without directly adding a corresponding term to the cost.

The way these goals are pursued gives rise to the name we came up with for the new training strategy proposed in this contribution, namely *split-boost neural network*. The first phase involves dividing the training set into k equal parts. In this, we were inspired by the renowned idea of k-fold cross validation (see Arlot and Celisse (2010)). From this procedure originates the name *split*. In this work, k is chosen equal to 2, but other values can be selected without any loss of generality. We consider a network structure with two fully-connected layers. The idea is to update the parameters of the first layer with gradient descent, explicitly calculating the terms that yield the gradient as a function of optimal values of the weights of the second layer. These weights are identified separately, using each time one portion of the training set. This is what leads the name *boost*. This procedure *boosts* training performances and includes automatic regularization in the network to avoid overfitting. The weights of the second layer, calculated independently on the two portions, are then *averaged* to identify the optimal value to use in prediction. This methodology finds similarities with the panorama of Ex-

⋆ Corresponding author: simone.formentin@polimi.it.

⋆⋆This paper is partially supported by FAIR (Future Artificial Intelligence Research) project, funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, Investment 1.3, Line on Artificial Intelligence), by the Italian Ministry of Enterprises and Made in Italy in the framework of the project 4DDS (4D Drone Swarms) under grant no. F/310097/01-04/X56. The activities of Loris Cannelli, Gabriele Maroni, and Dario Piga were supported by the European Union, Grant Agreement n. 101093126 (project ACES "Autopoietic Cognitive Edge-cloud Services") and by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 22.00490.

10.1016/j.ifacol.2024.08.535

treme Learning Machines (ELMs), see Huang and Guang-Bin (2015), in which the optimal values of the parameters of the second layer given those of the first are explicitly obtained. However, our work differs from traditional ELMs, where the parameters of the first layer are fixed a priori (and not updated) after initial randomization, see Huang et al. (2021). Additionally, backpropagation in ELMs is removed. Instead, we still perform backpropagation to train the first layer's parameters. Likewise, a one-shot optimization is performed (ridge regression) with the substantial difference in how the data are divided to parallelize the computation on 2 different batches to retrieve the parameters of the second layer explicitly. This strategy is expected to reduce the occurrence of overfitting without resorting to a regularization term. The paper is organized as follows. In Section 2 a review of the state of the art, the main challenges, limitations, and the benchmark network structure are presented. In Section 3 the mathematical formulation of the training strategy of the *split-boost* neural network is presented: in the first part of the section the mathematical notation is introduced, then in Section 3.1 the *split-boost* optimization problem for optimal weights computation is defined, and finally in Section 3.2 the details on training procedure, the policy for best epoch retrieval and the learning rate switching strategy are described. In Section 4 a numerical comparison between the traditional feed-forward neural network training and the split-boost training is presented. The paper is ended by some concluding remarks.

2. PROBLEM STATEMENT

The goal of this work is to propose an alternative training strategy for a classic feed-forward neural network. Without modification of the network structure, we want to show that using the same amount of data in a different way allows us to improve training performances and achieve implicit regularization, namely to obtain a regularization effect without modeling it explicitly. This allows us to simplify the tuning procedure of the network, reducing the number of hyperparameters to be calibrated. In this section, the basic structure of the architecture of a feed-forward neural network is introduced without going into details as this morphology is widely studied in the literature, see Rosenblatt (1958), Goodfellow et al. (2016). A feed-forward neural network is a deep learning model which, based on the interaction of several processing units, called neurons, introducing non-linearities on the inputs, can perform various tasks ranging from classification, regression and prediction of time series. The parameters that build this model are represented by the weight matrices that correspond to the different layers of neurons building the architecture. These weights are updated through epochs (e.g., several passes through the dataset) via gradient descent (or one of its variants) and the help of the well-known backpropagation algorithm Rumelhart et al. (1986), for the calculation of the gradient of the errors with respect to the weights of the network. Neural networks are models with a high descriptive capacity but are characterized by the phenomenon of overfitting, which can cause negative repercussions on the generalization capacity. Overfitting is one of the curses of general statistical learning. It often discourages users of artificial intelligence as the lack of a sufficient amount of data makes the architectures prone to its onset. In the literature, there are several documents relating to the description of the problem (see Tieleman and Hinton (2009), Wan and et al. (2013), Keskar and et al. (2016), Zhang and et al. (2021)). In the case of feed-forward neural networks, there are several strategies that can counter overfitting such as dropout, early stopping, regularization, data augmentation, batch normalization (e.g., Hinton et al. (2015), Loshchilov and Hutter (2017)). The goal of these methodologies is to prevent the neural network from overly relying (e.g. fitting the noise) on the data used in training. To do this, the idea is to reduce the number of network hyperparameters (or limit their norm) or artificially increase the available data. In this study, we assume *regularization* as a benchmark of anti-overfitting methodology. This technique consists in adding into the cost function that must be minimized (or equivalently the reward that must be maximized) a term that depends on the norm of the weights of the different layers of the neural network. During the training step, the neural network is forced to keep the norm of these weights limited to prevent an excessive increase in the cost term. In Figure 1 a sketch of the neural network architecture in matrix notation used in this work is shown. The considered structure consists of 2 layers (hidden and output) and is designed to solve a regression problem. Each input is processed by each layer (and by each neuron per layer) through the non-linear activation function f_1 (rectified linear unit, RELU). There is no restriction on the activation function choice (among the well-known set of activations). $X \in \mathbb{R}^{N\times D}$ is the input data matrix, where N is the number of samples and D is the number of features. $W_1 \in \mathbb{R}^{D\times H}$ is the weight matrix of the first (hidden) layer, where H is the number of neurons. $W_1^b \in \mathbb{R}^{H\times 1}$ is the bias vector of the first (hidden) layer. $Z_1 \in \mathbb{R}^{N\times H}$ are the pre-activations of the first layer. $X_1 \in \mathbb{R}^{N\times H}$ are the activations of the first layer. $W_2 \in \mathbb{R}^{H\times 1}$ is the weight matrix of the second (output) layer. $W_2^b \in \mathbb{R}$ is the bias of the output layer. $Y, \hat{Y} \in \mathbb{R}^{N\times 1}$ are respectively the matrix of the targets and the matrix of the prediction, and finally, J is the loss function. Layer biases W_1^b and W_2^b follow the same training procedure of the corresponding layer weights. For sake of compactness of notation, they are omitted. The optimization problem that must be solved to

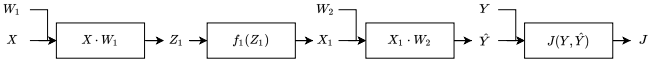

Fig. 1. Neural network architecture

find the optimal weights of a feed-forward neural network with 2 fully connected layers, following the traditional training procedure known in literature has the structure of the following unconstrained minimization problem:

$$\min_{W_1,W_2} \frac{1}{2N_t}||Y_t - f_1(X_t \cdot W_1)\cdot W_2||_2^2 + \frac{\lambda}{2}\sum_{i=1}^{2}||W_i||_2^2 \quad (1)$$

where the subscript t refers to the training set and λ is a hyperparameter that controls the intensity of the regularization. In the next section, we present an alternative, novel, way to write down the same optimization problem for training a 2-layer fully connected neural network in a way that allows us to best exploit the training information introducing an implicit regularizing effect and simultaneously reducing the number of parameters to be tuned. Indeed, within this framework, the regularization factor is not needed anymore.

3. MATHEMATICAL FORMULATION OF SPLIT-BOOST NETWORKS

This section illustrates this alternative training strategy of the proposed Split-Boost Neural Network. The updating of the weight parameters takes place separately for the 2 layers (hidden and output) that characterize the network architecture. The idea is to formulate the training procedure as a bilevel optimization problem, whose outer optimization variables are the weights of the hidden layer W_1, and the inner optimization variables are the weights of the output layer W_2. For a regression problem, the optimal values of the parameters of the output layer W_2 can be obtained in closed form by solving two least square problems. These least square problem solutions represent the constraints of the first optimization problem. The algorithm involves first a *splitting step*, in which the training set is divided into two sub-sets (a reasonable choice is to divide it equally, we do not claim that this choice is the only possible one, however, it guarantees that both optimization problems see the same amount of data, avoiding the unbalancing towards one of the two partitions.). Both subsets are then used to solve, separately, two least squares problems. Once the least squares problems are solved (e.g. the optimal values for W_2 with respect to the two sub-sets are found), the whole training set is used to update the values of W_1. From here, *the boosting idea*. The optimal values obtained with the first sub-set are used to generate the prediction for the data belonging to the second sub-set and vice-versa. Our goal is to show that this methodology can effectively replace the regularization term in a traditional feed-forward neural network, overcoming its performance. This step represents one of the main differences compared to traditional network training: dividing the training set into 2 batches, used to calibrate the parameters of the second layer independently, allows us to improve the information content extraction. In Table 1 the symbols and their description is summarized.

Table 1. Notation.

Symbol	Description
N	Number of samples
t, v, ts	Training, validation and test sets
a	Training set partition "A"
b	Training set partition "B"
D	Number of input features
$X \in \mathbb{R}^{N\times D}$	Input data
$Y, \hat{Y} \in \mathbb{R}^{N\times 1}$	Output data, prediction
$Z \in \mathbb{R}^{N\times H}$	Pre-activations
$X_1 \in \mathbb{R}^{N\times H}$	Activations
H	Number of neurons per layer
$W_1 \in \mathbb{R}^{D\times H}$	Weights of hidden layer
$W_2 \in \mathbb{R}^{H\times 1}$	Weights of output layer
$f_1 : \mathbb{R}^{N\times H} \to \mathbb{R}^{N\times H}$	Activation function
J	Cost function
λ	Regularization parameter
γ	Learning rate

3.1 Optimization problem, forward-backward propagations.

The heart of the training algorithm is represented by the optimization problem shown in Equation (2). Since this cannot be solved in closed form, it is solved iteratively through a descending gradient problem. The value of the W_1 parameters is therefore updated as the epochs pass.

$$\min_{W_1} J = \frac{1}{2N_b}\|Y_b - f_1(X_b \cdot W_1)\cdot W^*_{2a}(W_1)\|_2^2 + \frac{1}{2N_a}\|Y_a - f_1(X_a \cdot W_1)\cdot W^*_{2b}(W_1)\|_2^2 \quad (2a)$$

$$W^*_{2a}(W_1) = \underset{W_2}{\operatorname{argmin}} \frac{1}{2N_a}\|Y_a - f_1(X_a \cdot W_1)\cdot W_2\|_2^2 \quad (2b)$$

$$W^*_{2b}(W_1) = \underset{W_2}{\operatorname{argmin}} \frac{1}{2N_b}\|Y_b - f_1(X_b \cdot W_1)\cdot W_2\|_2^2 \quad (2c)$$

In what follows, the subscripts k, j are used to refer both to training subsets A and B. Since the description of the equations referring to the two training sets mirror each other, the notation will follow as $\forall k, j \in [a, b], k \neq j$. The whole is repeated for both admissible values of k and j, training is *split*. Summarizing: $k \to a$ in set A, $k \to b$ in set B, $j \to b$ in set A, $j \to a$ in set B. In the following equations, the forward propagation is described. We do not go into details as this step is well-known in literature (see Rumelhart et al. (1986), LeCun and et al. (1998)).

$$Z_{1k} = X_k \cdot W_1 \quad (3a)$$

$$X_{1k} = f_1(Z_{1k}) \quad (3b)$$

$$\hat{Y}_k = X_{1k} \cdot W_{2j} \quad (3c)$$

$$J_{W_{2k}} = \frac{1}{2N_k}\|Y_k - \hat{Y}_k\|_2^2 \quad (3d)$$

We can compute explicitly the optimal values for the output layer parameters W_2 solving a least square problem:

$$W^*_{2k}(W_1) = (X_{1k}^T \cdot X_{1k})^{-1} \cdot X_{1k}^T Y_k. \quad (4)$$

Solving explicitly Equation (4) we derive the optimal values for the output layer parameters computed with the forward pass in the two separate training sub-sets. Notice that W^*_{2k} is function of W_1 through the dependency on X_{1k}. For sake of brevity in the following derivation, we write W^*_{2k} in place of $W^*_{2k}(W_1)$. For sake of compactness, the Jacobian expression is derived in scalar form. Nonetheless, it must be interpreted as the matrix of first-order partial derivatives of a vector-valued function with respect to its input variables. Since W^*_{2k} is an optimum computed explicitly, it is true that:

$$\left.\frac{\partial J_k(W_1, W_{2k})}{\partial W_{2k}}\right|_{W_{2k}=W^*_{2k}} = \frac{\partial J_k(W_1, W^*_{2k})}{\partial W^*_{2k}} = 0 \quad (5)$$

Differentiating both sides with respect to W_1 and using the chain rule according to the influence diagram in Figure 2:

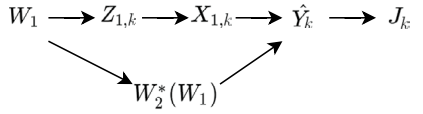

Fig. 2. Influence diagram describing the interaction between network layer parameters.

$$\frac{\partial}{\partial W_1}\left(\frac{\partial J_k(W_1, W^*_{2k})}{\partial W^*_{2k}}\right) = 0$$

$$\frac{\partial}{\partial W_1}\left(\frac{\partial J_k(W_1, W^*_{2k})}{\partial W^*_{2k}}\right) + + \frac{\partial}{\partial W^*_{2k}}\left(\frac{\partial J_k(W_1, W^*_{2k})}{\partial W^*_{2k}}\right)\cdot\frac{\partial W^*_{2k}}{\partial W_1} = 0$$

$$\frac{\partial^2 J_k(W_1, W^*_{2k})}{\partial W_1 \partial W^*_{2k}} + \frac{\partial^2 J_k(W_1, W^*_{2k})}{\partial W^*_{2k}\partial W^{*\,\mathrm{T}}_{2k}}\cdot\frac{\partial W^*_{2k}}{\partial W_1} = 0$$

Solving for the Jacobian $\frac{\partial W_{2k}^*}{\partial W_1}$:

$$\frac{\partial W_{2k}^*}{\partial W_1} = -\left(\frac{\partial^2 J_k(W_1, W_{2k}^*)}{\partial W_{2k}^* \partial W_{2k}^{*\,\mathrm{T}}}\right)^{-1} \cdot \frac{\partial^2 J_k(W_1, W_{2k}^*)}{\partial W_1 \partial W_{2k}^*} \quad (7)$$

And then with notation abuse (we use the approximation symbol to treat the Jacobian as in the scalar case even though, as commented earlier, it must be interpreted as the matrix of first-order partial derivatives):

$$\mathbf{J}_{W_1} W_{2k}^* \approx \frac{\partial W_{2k}^*}{\partial W_1} \quad (8)$$

The Jacobian $\mathbf{J}_{W_1} W_{2k}^*$ described in Equations (7) and (8) is a fundamental ingredient in the evaluation of the gradient of the cost function J, to update W_1 values. To complete the training procedure, the backward propagation step must be performed. Similarly to forward propagation, backward propagation also takes place by working simultaneously on both sub-sets of the training set. This allows us to derive the expressions of the gradients necessary for the final calculation of the gradient of J with respect to W_1:

$$\nabla_{\hat{Y}_k} J_k = \frac{1}{N_k}(\hat{Y}_k - Y_k) \quad (9a)$$

$$\nabla_{X_{1k}} J_k = \nabla_{\hat{Y}_k} J_k \cdot W_{2j}^{*\,T} \quad (9b)$$

$$\nabla_{W_{2j}^*} J_k = X_{1k}^T \cdot \nabla_{\hat{Y}_k} J_k \quad (9c)$$

$$\nabla_{Z_{1k}} J_k = \nabla_{X_{1k}} J_k \circ f_1'(Z_{1k}) \quad (9d)$$

For a deeper understanding of how backpropagation works, refer to Hecht-Nielsen (1987), Glorot et al. (2011). Merging together the results obtained by solving Equations (3), (4) and (9) we derive the expression of the gradient of the cost function (2a) with respect to the weights of the first layer, W_1, described in the following equation:

$$\begin{aligned}\nabla_{W_1} J =& X_b^T \nabla_{Z_{1b}} J_b + \mathbf{J}_{W_1} W_{2a}^{*\,T} \nabla_{W_{2a}^*} J_b + \\ &+ X_a^T \nabla_{Z_{1a}} J_a + \mathbf{J}_{W_1} W_{2b}^{*\,T} \nabla_{W_{2b}^*} J_a \end{aligned} \quad (10a)$$

$$W_1 = W_1 - \gamma \nabla_{W_1} J \quad (10b)$$

$$W_2 = \frac{W_{2a}^* + W_{2b}^*}{2} \quad (10c)$$

Thanks to Equations (10) and (4) we can update network parameters between two consecutive epochs. Notice the difference: W_1 is updated through gradient descent in Equation (10b) while W_2 is obtained as the average of the optimal W_2^* computed explicitly by looking at the two training sub-sets separately in Equation (10c). Notice that, in general, this closed-form solution for W_2 might not be practicable (e.g. in the case of classification problems, the concatenation of non-linearities through the different layers (more than 2) of the network would make it unattainable to formulate a least squares problem, as in this case, efficiently solved analytically). In that case, a similar expression to (10a) must be derived and update W_2 with gradient descent. However, for this setting, it improves the computational time. Equations (10) represent the heart of the algorithm, defining the *boosting* step. The mixing of the two sub-sets of the training set embedded in the gradient expression and in optimal values for W_2 can improve training performance achieving the same training cost in a lower number of epochs and at the same time avoiding overfitting without the aid of regularization terms. This is critical for reducing the number of hyperparameters.

With the updated weights obtained in Equations (10), the prediction is computed as follows:

$$\hat{Y} = f_1(X \cdot W_1) W_2 \quad (11)$$

3.2 Details on training procedure

Early stopping procedure is used. The stop condition for training is obtained by monitoring the status of the validation cost, as follows:

$$|J_v(k) - J_v(k-1)| < \epsilon \quad (12)$$

where k is the epoch index and ϵ) is the stop threshold. If the variation of the validation cost between two consecutive epochs is less than ϵ, training stops and the optimal number of epochs is retrieved. After the computation of the optimal number of epochs, the network is re-trained using as new training set the sum of the original training set, and the validation set. Performances are evaluated considering the test set. The split-boost strategy has a learning rate varying with the number of epochs. This choice avoids oscillations in training cost (an excessively large learning rate leads to fluctuations and consequent degradation of performance). In the example reported in the next section, the following switch condition is adopted:

$$\begin{cases} \gamma = \gamma^* \;\; if \;\; J_t(k) - J_t(k-1) > 0 \\ \gamma = \dfrac{\gamma^*}{10} \;\; otherwise \end{cases} \quad (13)$$

where γ^* is the best learning rate, chosen after a sensitivity analysis (see Section 4.2) carried on the validation cost.

4. AN EXPERIMENTAL CASE STUDY

In this section, we show the comparison between a traditional feed-forward neural network and the split-boost neural network applied to a real-world regression problem. The code is written in Python 3.7.13. The Python library used to develop the neural networks is PyTorch Paszke and et al. (2019). Simulations run on an Intel Core i7-8750H with 6 cores, at 2.20 GHz (maximum single core frequency: 4.10 GHz), with 16 GB RAM.

4.1 Dataset description

The case-study is the medical insurance forecast of patients living in the U.S., given a set of clinical features. Data are open-source and offered in Lantz (2013). In Table 4.1 data features and targets are summarized. The goal is to predict the medical insurance charge ($) given a set of $D = 6$ features: age, sex, BMI, number of children, smoking condition, and region of residence. The number

Features	**Description**
Age	Age of primary beneficiary
Sex	Insurance contractor gender
BMI	Body mass index
Children	Number of children
Smoker	Smoker condition
Region	Residential area in the U.S.
Target	**Description**
Charge	Medical insurance bill [$]

Table 2. Medical Insurance Dataset: features and target.

of people in the study is $N = 1338$. Dataset is split into training, validation and test sets according to the proportions shown in Figure 3. Test set is of 20% of the total. The validation set is 16%. The training set is 64%. In the case of the split-boost neural network, the training set is further divided into two halves, 32% each.

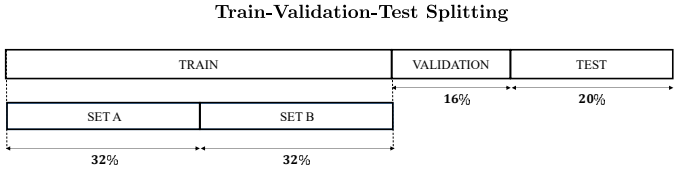

Fig. 3. Train-Validation-Test splitting. The training set is divided into two sub-sets A and B.

4.2 Networks Hyperparameters

In this section, we show the hyperparameter tuning procedure. Sensitivity analysis of the learning rate γ and the regularization factor λ is performed. In the case of the split-boost neural network, there is the advantage of not having a regularization term which, instead, is replaced by the boosting procedure. Sensitivity analysis with respect to γ is described in the upper plot of Figure 4. The validation cost $J_v = \frac{1}{2N_v}(Y_v - \hat{Y}_v)^2$ associated to different choices of γ is shown. The best choice for both networks is $\gamma = 0.1$. In the lower plot of Figure 4 the sensitivity analysis with respect to the regularization hyper-parameter λ is shown (only for the feed-forward neural network), evaluating the validation cost. The goal is to compare the best possible feed-forward regularized neural network with the split-boost network. The best choice is $\lambda = 0.01$. In Table 3 the

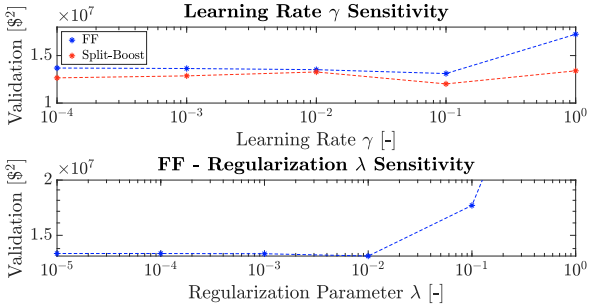

Fig. 4. Hyperparameter tuning: sensitivity analysis with respect to the learning rate γ (upper panel); sensitivity analysis with respect to the regularization parameter λ (lower plot).

network parameters are summarized. Network architecture is the same. Same number of layers (2), neurons per layer ($H = 10$) and activation function (RELU). Split-boost strategy does not need any regularization parameter.

Hyperparameter	FF	Split-Boost
L (Layers)	2	2
H	10	10
f_1	RELU	RELU
γ	0.1	0.1
λ	0.01	–

Table 3. Networks Hyperparameters.

4.3 Numerical Simulations

In this section we show numerical insights about the training procedure of the split-boost network. Figure 5 shows the trend of the training cost along the training epochs for the two networks. The split-boost network is represented by the solid red line, the feed-forward is represented by the blue lines for varying values of λ. The solid blue line corresponds to the best λ identified (see Table 3). As λ increases, the training cost of the feed-forward network increases. For smaller values of λ, it decreases. Split-boost training cost converges to values close to those of the feed-forward regime for a substantially lower number of epochs, $E^*_{TB} = 50$ epochs against $E^*_{FF} = 200$ epochs. This allows us to conclude that the split-boost procedure, in this case, can converge to the maximum information content extractable from the training set, in a smaller number of epochs, and with an implicit regularization effect. Notice that the selection of the best number of epochs (implementing early stopping strategy) is performed looking at validation cost as discussed in Section 3.2 for both the networks. After the best epoch retrieval, both the networks are re-trained considering as training set the union of the previous training and validation sets. In Figure 6 the computational training

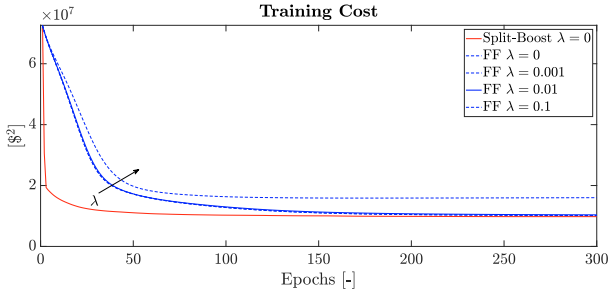

Fig. 5. Training cost: comparison between *split-boost* and *feed-forward* network for different values of the regularization hyper-parameter λ. Solid line is the best configuration of λ.

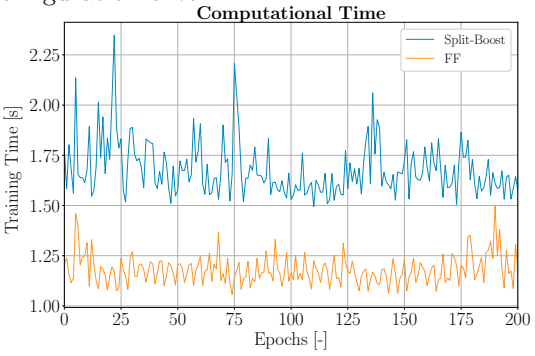

Fig. 6. Training time [s] per epoch: comparison between *split-boost* and *feed-forward* network over 200 epochs.

time required by the 2 strategies is shown over 200 epochs. Split-boost strategy shows an average training time of $T_{SB} = 1.679\,s$ while the feed-forward of $T_{FF} = 1.179\,s$. On average, each epoch of the split-boost takes 42% more time than the feed-forward. However, considering the training convergence shown in Figure 5, which highlights that split-boost training cost converges at the regime in $E^*_{SB} = 50$ epochs against the $E^*_{FF} = 200$ of the feed-forward, leads to the average computational requirements of:

$$E^*_{SB} \cdot T_{SB} = 83.95\,s \leq E^*_{FF} \cdot T_{FF} = 235.8\,s. \qquad (14)$$

In Figure 7 the re-training procedure of the split-boost neural network is shown. On the left the training and test costs are shown. If compared with Figure 5, the number of epochs to which the training cost regime is reached is higher. This depends on the fact that the training set is larger (it includes also the previous validation set). In the middle, the plot of the regression prediction versus target

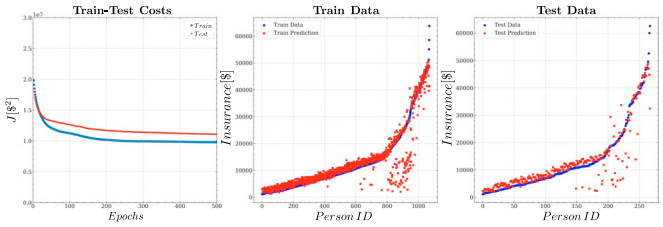

Fig. 7. Training and test cost for the split-boost (left panel); predicted vs estimated output in the training dataset (middle panel); predicted vs estimated output in the test dataset (right panel);

values for each person in the training set is shown. On the right plot, regression prediction versus target values for test data is shown. In the middle and right panels it is shown that the neural network can map successfully the non-linear relationship between the features collected in Table 4.1 and the regression target. There are some patients whose characteristics escape mapping: with similar features compared to the remaining patients, a higher medical insurance cost is attributed to them by the experts. To evaluate the performance of the split-boost

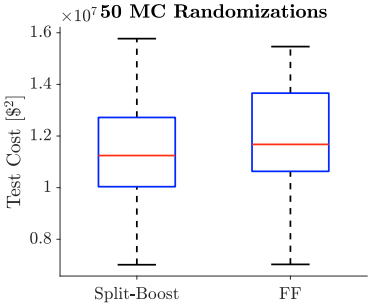

Fig. 8. Test cost (after re-training procedure) for 50 randomizations of the extraction of training, validation and test sets.

strategy with respect to the best regularized feed-forward neural network with $\lambda = 0.01$, derived from the sensitivity analysis carried on the regularizing term for the traditional FFNN in Fig. 4, 50 Monte Carlo randomizations of the dataset were performed, extracting 50 different combinations of training, validation and test sets. The results obtained on the test set were collected in Figure 8. The test cost $J_{ts} = \frac{1}{2N_{ts}}(Y_v - \hat{Y}_{ts})^2$ for the split-boost strategy is statistically lower. Split-boost test cost is lower in 72% of the cases. This proves that, with statistical evidence, the split-boost network overcomes the feed-forward one also in terms of prediction accuracy.

5. CONCLUSIONS

In this article, we show an alternative training approach for feed-forward neural networks. We call this strategy "split-boost" to recall the idea that dividing (*split*) the dataset and combining the subsets *might* lead to an improvement (*boost*) in performance. In a real-world case study, the "split-boost" approach turns out to: lead to higher predictive performance than traditional training; be computationally advantageous since in training as it converges within a smaller number of epochs, although having a larger computational time per epoch. The proposed strategy implicitly counteracts overfitting. Future activities will focus on an extensive validation of the proposed training strategy, as well as on its generalization and extension to multi-layer networks. We will apply the architecture on other ML problems (e.g., time-series classification or prediction).

REFERENCES

Arlot, S. and Celisse, A. (2010). A survey of cross-validation procedures for model selection.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. *14th international conference on artificial intelligence and statistics*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Hecht-Nielsen, R. (1987). Kolmogorov's mapping neural network existence theorem.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv:1503.02531*.

Huang and Guang-Bin (2015). What are extreme learning machines? filling the gap between frank rosenblatt's dream and john von neumann's puzzle. *Cognitive Computation*, (7), 263–278.

Huang, Xuyang, and et al. (2021). A backpropagation extreme learning machine approach to fast training neural network-based side-channel attack. *AsianHOST,IEEE*.

Keskar, N. and et al. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv:1609.04836*.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 84–90.

Lantz, B. (2013). Machine learning with r: Learn how to use r to apply powerful machine learning methods and gain an insight into real world applications. *Livery Place*.

LeCun, Y. and et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86.11, 2278–2324.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv:1711.05101*.

Paszke, A. and et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65.6, 386.

Rumelhart, D.E., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323.6088, 533–536.

Tieleman, T. and Hinton, G. (2009). Using fast weights to improve persistent contrastive divergence. *26th annual international conference on machine learning*.

Wan, L. and et al. (2013). Regularization of neural networks using dropconnect. *International conference on machine learning, PMLR*.

Zhang, C. and et al. (2021). Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3), 107–115.