Neural Networks 140 (2021) 39–48

Contents lists available at ScienceDirect

Neural Networks

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

Long-term Cognitive Network-based architecture for multi-label classification

Gonzalo Nápoles [a,*], Marilyn Bello [b,c], Yamisleydi Salgueiro [d]

[a] Department of Cognitive Science & Artificial Intelligence Tilburg University, The Netherlands
[b] Faculty of Business Economics, Hasselt University, Belgium
[c] Department of Computer Science, Central University of Las Villas, Cuba
[d] Department of Computer Science, Faculty of Engineering, Universidad de Talca, Campus Curicó, Chile

ARTICLE INFO

Article history:
Received 1 August 2020
Received in revised form 27 February 2021
Accepted 1 March 2021
Available online 6 March 2021

Keywords:
Long-term cognitive networks
Recurrent neural networks
Backpropagation
Multi-label classification

ABSTRACT

This paper presents a neural system to deal with multi-label classification problems that might involve sparse features. The architecture of this model involves three sequential blocks with well-defined functions. The first block consists of a multilayered feed-forward structure that extracts hidden features, thus reducing the problem dimensionality. This block is useful when dealing with sparse problems. The second block consists of a Long-term Cognitive Network-based model that operates on features extracted by the first block. The activation rule of this recurrent neural network is modified to prevent the vanishing of the input signal during the recurrent inference process. The modified activation rule combines the neurons' state in the previous abstract layer (iteration) with the initial state. Moreover, we add a bias component to shift the transfer functions as needed to obtain good approximations. Finally, the third block consists of an output layer that adapts the second block's outputs to the label space. We propose a backpropagation learning algorithm that uses a squared hinge loss function to maximize the margins between labels to train this network. The results show that our model outperforms the state-of-the-art algorithms in most datasets.

1. Introduction

When it comes to traditional pattern classification (Duda, Hart, & Stork, 2012), each instance (example) is expected to be associated with a single decision class. Thus, one fundamental assumption in classic supervised learning is that each problem instance belongs to only one concept (Bello, Nápoles, Vanhoof, & Bello, 2019). In contrast, in multi-label learning (also referred to as multi-label classification (MLC)), each instance involves multiple semantic meanings simultaneously, i.e. each instance is associated with a set of labels instead of a single label. The learning problem in MLC aims at computing a function able to associate a given instance with a set of labels.

The literature reports a fair variety of algorithms (Gibaja & Ventura, 2014; Herrera, Charte, Rivera, & Del Jesus, 2016b; Zhang & Zhou, 2013) to deal with MLC problems, where the main focus is to obtain the highest prediction rates possible. Herrera et al. (2016b) discern among three types of MLC algorithms: the data transformation methods, adaptation algorithms, and ensemble classifiers. The former group transforms a given multi-label dataset into one or more easier-to-handle single-label problems. After the transformation, a single-label classifier can be applied to solve the individual problems. Two well-known representatives of this family are the binary relevance (Godbole & Sarawagi, 2004; Zhang, Li, Liu, & Geng, 2018) and the label powerset (Boutell, Luo, Shen, & Brown, 2004) transformations. Adaptation methods handle the multi-label dataset directly. Thus they are based on modifications or generalizations of existing single-label classification models. Some solutions reported on the literature include neural networks (Kongsorot & Horata, 2014; Law, Chakraborty, & Ghosh, 2017; Zhang & Zhou, 2006), support vector machines (Elisseeff & Weston, 2002), decision trees (Vens, Struyf, Schietgat, Džeroski, & Blockeel, 2008), and nearest neighbor classifier (Zhang & Zhou, 2007). Finally, the third group consists of ensemble solutions such as ensembles of binary classifiers (Read, Pfahringer, Holmes, & Frank, 2009) and ensembles based on multi-class methods (Read, Pfahringer, & Holmes, 2008).

Since multi-label objects can be associated with several concepts simultaneously, the decision class boundaries often overlap. To handle this issue, researchers have employed neural network solutions (Hinton & Salakhutdinov, 2006) to learn complex multi-label class boundaries. In the MLC literature, modifications to the multilayer perceptron model (Zhang & Zhou, 2006), radial basis functions (Zhang, 2009), extreme learning machines

* Corresponding author.
E-mail address: g.r.napoles@uvt.nl (G. Nápoles).

https://doi.org/10.1016/j.neunet.2021.03.001

(Kongsorot & Horata, 2014) and deep neural networks (Lian, Liu, Lu, & Luo, 2019; Yeh, Wu, Ko, & Wang, 2017) have been reported. Some of these solutions rely on Convolutional Neural Networks (CNNs) (Goodfellow, Bengio, & Courville, 2016; LeCun, Bengio, & Hinton, 2015) to tackle diverse types of problems concerning image (Wei, et al., 2015; Wu, et al., 2015; Zhu, Liao, Lei, & Li, 2017), sound (Choi, Fazekas, & Sandler, 2016), text (Rios & Kavuluru, 2015) and video (Abu-El-Haija, et al., 2016; Janwe & Bhoyar, 2018). For example, in Wang, et al. (2016) the authors proposed a unified framework for multi-label image classification. Such a framework involves CNNs and recurrent neural networks (RNN) to model the label co-occurrence dependency in a joint image/label embedding space. Another interesting solution was reported in Chen, Ye, Xing, Chen, and Cambria (2017). This piece of research presented an ensemble of CNNs and RNNs to capture both the global and the local textual semantics and model high-order label correlations while having a tractable computational complexity. In Chen, Chen, Yeh, and Wang (2018) the authors proposed an RNN-based approach for image multi-label classification. This neural system allows identifying visual objects of interests with varying sizes without the prior knowledge of particular label ordering while also exploiting the underlying information about label co-occurrence.

A serious issue present in many MLC problems concerns data sparsity. Overall, data sparsity negatively impacts the performance of machine learning methods (Adomavicius & Zhang, 2012). This is especially critical in neural-network-based methods where data sparsity often leads to underfitting (Krishnan, Liang, & Hoffman, 2018). However, sparse datasets are ubiquitous in real-world applications, so developing methods capable of handling this condition is necessary. There is no well-accepted approach to handling sparse inputs for neural networks. Most articles addressing this issue pre-process the data either by imputing the values or deleting the cases or attributes, with all the difficulties they imply (García, Luengo, & Herrera, 2015). Other approaches adapt algorithms with capabilities to handle data sparsity without the need for a pre-processing step (Law & Ghosh, 2019).

In this paper, we present a neural system to solve MLC problems described by tabular data that might involve sparse features. The architecture of this model involves three sequentially connected neural blocks. The first block involves a multilayered feed-forward network that extracts high-level features. This block reduces the dimensionality of the space by encoding the relevant information in the high-level features. The second block consists of a Long-term Cognitive Network (LTCN)-based network (Nápoles, Vanhoenshoven, Falcon, & Vanhoof, 2020), which performs the reasoning on the extracted features. The reasoning mechanism of this network is modified to include the neurons' initial state (i.e., the activation vector) when computing the vector state in each abstract layer. This strategy prevents the input signal from vanishing as the recurrent inference process progresses. Finally, the third block adapts the outputs of the recurrent block to the label space. In addition, we propose a backpropagation learning algorithm to compute the network's learnable parameters. This method uses a squared hinge loss function to increase the separability between the labels, potentially reducing the misclassifications.

Overall, the contributions of this paper can be summarized as follows. Firstly, we present a three-block neural system for potentially sparse MLC problems. Secondly, we propose a new updating rule to prevent the input signal of the LTCN model from vanishing when performing the recurrent inference process in the second block. Finally, we introduce a backpropagation algorithm to train the neural system as a whole while taking into account the new LTCN's reasoning mechanism.

The rest of this paper is organized as follows: Section 2 briefly introduces the LTCN model, which is an important building-block of our solution. Section 3 presents the three-block neural system to solve MLC problems, while Section 4 introduces the backpropagation learning formalism. The simulations on benchmark problems and the comparison against state-of-the-art methods are discussed in Section 5. Section 6 provides some concluding remarks and future work directions to be explored.

2. Long-term cognitive networks

Before presenting our proposal, we will briefly describe the LTCN model (Nápoles et al., 2020), a recurrent neural system conceived initially for modeling and simulation purposes. This model does not allow for explicit hidden neurons as modeling scenarios often demand to operate with models that are interpretable to some extent. However, perhaps the most attractive feature of this network is that it allows domain experts to inject prior knowledge into the model through the weight matrix. This is why LTCN-based simulation models are often trained in a non-synaptic fashion (i.e., learning the transfer function parameters).

When designing an LTCN model, both problem features and decision variables are mapped onto neural concepts, while the weights connecting the concepts are expected to be provided by domain experts. These weights do not change from an iteration to another as typically occurs in recurrent systems; only the sigmoid transfer function attached with each neuron does.

Eq. (1) displays the reasoning mechanism used to compute the activation value $a_i^{(r+1)}(k)$ of the ith neural entity in the $(r+1)$-th iteration,

$$a_i^{(r+1)}(k) = f_i^{(r+1)}\left(\sum_{j=1}^{M} w_{ji} a_j^{(r)}(k)\right) \tag{1}$$

where w_{ji} denotes the weight connecting the corresponding neurons, k is the index of the initial activation vector (the need for this index will become clear when presenting the backpropagation method in Section 4), while $f_i^{(r+1)}$ is the transfer function used to keep the neurons' activation values within the activation interval.

Although LTCNs have proven quite effective (in terms of both accuracy and interpretability) to address simulation problems (Nápoles, Salmeron, & Vanhoof, 2019; Nápoles et al., 2020), we need to take into consideration the particularities of MLC problems. For example, if we let an LTCN-based model operate on sparse MLC datasets, the model will likely report good results, but the training time would be unnecessarily costly. Besides, the non-synaptic learning approach (Nápoles et al., 2020; Nápoles, Vanhoenshoven, & Vanhoof, 2019) is no longer the best option since experts are not expected to provide the relationships among variables in complex problems.

3. The proposed ML-LTCN architecture

Before describing the details of our proposal, let us define the multi-label classification problem.

Let us suppose that $\mathcal{U}$ is a N-dimensional instance space called the universe, and $\mathcal{L} = \{l_1, l_2, \ldots, l_L\}$ denotes the label space with L possible class labels. The task of multi-label learning is to learn a function $\Theta : \mathcal{U} \longrightarrow 2^{\mathcal{L}}$ from the multi-label training set $\{(u_k, \mathcal{L}_k) \mid 1 \leq k \leq K\}$. For each multi-label instance $(u_k, \mathcal{L}_k)$, $u_k \in \mathcal{U}$ is a N-dimensional feature vector $(x_1(k), x_2(k), \ldots, x_N(k))$ and $\mathcal{L}_k \subseteq \mathcal{L}$ is the set of labels associated with u_k. For any unseen instance $u \in U$, the multi-label classifier $\Theta(\cdot)$ predicts $\Theta(u) \subseteq \mathcal{L}$ as the set of labels associated with u.

The neural architecture proposed in this section involves three sequentially connected blocks. The first one contains a multilayered feed-forward network aimed at extracting hidden features

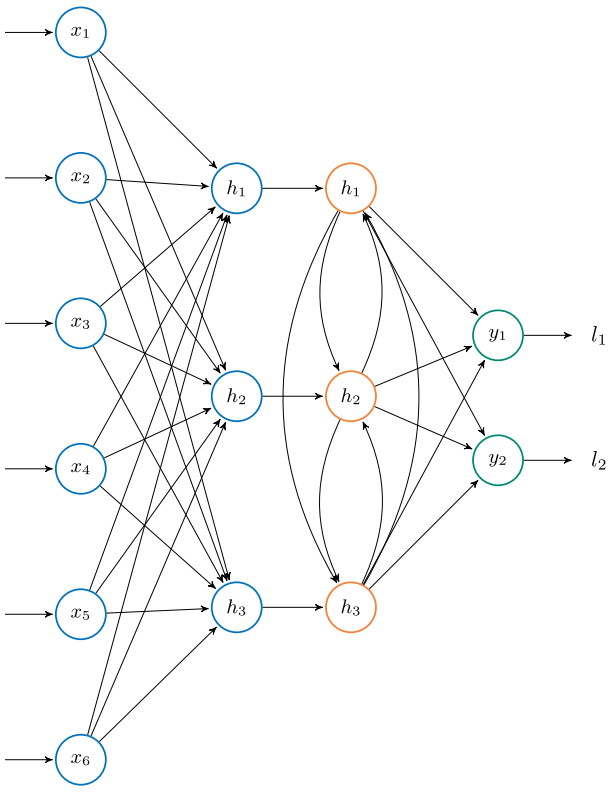

Fig. 1. ML-LTCN classifier with six problem features, three hidden features and two possible labels. In this example, x_i is the ith problem feature, h_j is the jth hidden feature, y_v is the numerical value associated with the vth output, and $l_v \in \{-1, 1\}$ is a binary label as determined by a fixed threshold.

from raw data. The second one involves an LTCN that equips the models with improved reasoning capabilities by operating on the extracted features. Finally, the third block is the output layer connecting the LTCN block with the problem labels.

It would be convenient to define the concept of *abstract layer* in a recurrent neural network such as the LTCN model. We can define an abstract layer as the state vector produced with Eq. (1) in each iteration. Hence, we can say that our architecture will comprise $T = S+R+1$ layers, with S being the number of hidden layers in the first block and R being the number of abstract hidden layers in the second block.

Concerning the number of neurons in the first block, we have adopted the following strategy. The number of hidden neurons in that block decreases with the number of layers, starting with the number of problem features and ending with the desired number of high-level features. The number of abstract hidden neurons in the second block is equal to the number of hidden features. It is worth mentioning that the number of abstract hidden neurons does not change from an iteration to another. Finally, the number of neurons in the third block matches the number of problem labels. Fig. 1 depicts, as an example, the ML-LTCN model for a problem with $N = 6$ features and $L = 2$ labels, whereas $H = 3$ stands for the number of hidden features to be extracted with the aid of the first reasoning block. For the sake of clarity and if no confusion arises, we will assume that the weights feeding the second block are frozen with the identity matrix.

For each instance, the first neural block receives a vector $x \in \mathbb{R}^N$ (assuming that discrete features have already been one-hot encoded if necessary) and produces a vector $h \in \mathbb{R}^H$, which will be used to feed the second layer. Similarly, the third layer receives a vector $h \in \mathbb{R}^H$ that is mapped to an output vector $y \in \mathbb{R}^L$. Notice that the resultant labels are derived from the vector $y \in \mathbb{R}^L$.

It should be highlighted that each block in the pipeline has a well-defined function. For example, the first block reduces the problem's dimensionality while comprising the relevant information of potentially sparse features into high-level ones. The purpose of this block is similar to the encoding sub-network of autoencoders. The second block aims at discovering hidden patterns from the high-level features extracted by the first block. The third block maps the hidden patterns to the label space, thus assigning the corresponding class labels to the instance.

Eq. (2) formalizes how to compute the activation values of neurons in an unfolded ML-LTCN, regardless of the neural block they belong to,

$$a_i^{(t+1)}(k) = f\left(\sum_{j=1}^{M^{(t)}} \underbrace{w_{ji}^{(t)} a_j^{(t)}(k)}_{implicit} + \underbrace{c_i^{(t+1)} a_i^{(0)}(k) - b_i^{(t+1)}}_{explicit}\right) \tag{2}$$

where $M^{(t)}$ stands for the number of neurons in the tth layer, k is the index of the activation vector used as the initial stimulus, b_i denotes the bias weight of each neuron, while w_{ji} is the weight connecting two neurons, the latter belonging to the current layer. Moreover, c_i is a learnable parameter that controls the extent to which the neuron's initial activation value $a_i^{(0)}(k)$ is taken into account when computing its next activation value.

Observe that the reasoning mechanism in Eq. (2) makes no distinction between hidden/outputs layers and iterations of the LTCN model. The rationale behind this assumption is that the activation values of LTCN's neurons in each iteration define a sort of abstract hidden neurons that form abstract hidden layers. Another important detail of the ML-LTCN model is that, when it comes to computing the activation values of neurons in the second block, we use the initial activation value of each neuron multiplied by a learnable weight. This strategy prevents the signal that arrives at the second neural block to vanish due to the LTCN's recurrent reasoning process. Hence, we gather the neuron's knowledge into two categories: implicit and explicit. The former refers to the knowledge that each abstract neuron receives from others during the reasoning steps. In contrast, the latter refers to the bias and the memory the neuron has about its initial activation. In the first and third blocks, we can assume that there the initial state of neurons is zero.

Eq. (2) leads to a new type of LTCN that considers the initial state and the previous neurons' states when computing the network state (i.e., the activation values of neurons in a given abstract layer). This is a fundamental difference with the original LTCN model proposed in Nápoles et al. (2020). The inspiration for this modification comes from the Long Short-term Memory (Hochreiter & Schmidhuber, 1997) that uses the input vector in each calculation step. In that way, we significantly reduce the risk for the LTCN to converge to unique fixed-point attractors since the input signal will modify the network state in each abstract layer. As stated in Nápoles et al. (2019), unique fixed points usually hinder the prediction capabilities of recurrent networks without external inputs since different inputs would be associated with a single output.

The second block has two operation modes: the unfolded one (depicted in Eq. (2)) and the folded one, which replaces the weights $w_{ji}^{(t)}$ with $\frac{1}{R}\sum w_{ji}^{(t)}$. The unfolded operation mode is mainly used when performing the backward pass (see Section 4). In contrast, the folded mode is mainly used during the forward pass.

Finally, the proposed ML-LTCN model operates with scaled exponential linear units (Klambauer, Unterthiner, Mayr, & Hochreiter, 2017). These neural units use the transfer function depicted below:

$$f(x) = \begin{cases} \beta x & x > 0 \\ \beta\alpha(e^x - 1) & x \leq 0 \end{cases} \tag{3}$$

where α and β are parameters derived from the inputs. The advantage of this transfer function is that it performs an internal normalization operation such that the current layer preserves the mean and variance of the previous one. The self-normalizing step is convenient in neural models involving many blocks and layers. When classifying an instance, the output of this function in the last layer is transformed with the sign function to determine the labels to be attached to that instance.

4. Backpropagation learning

In this section, we derive a backpropagation algorithm to adjust the parameters attached to the ML-LTCN model. This means that we need to estimate the weights of the multilayered network in the first block, the weights of the folded LTCN model (which include the weights of external inputs) in the second block, the output weights in the last neural block, and the corresponding bias weights in each block. It is worth mentioning that, while we perform the forward pass with the folded LTCN, the backward pass is performed on the unfolded LTCN. In the end, the LTCN model is a recurrent system in which weights do not change from an iteration to another.

Before moving forward, it seems convenient to set the notation straight. Let T denote the total number of layers of the ML-LTCN model, which include both regular layers (the ones in the first and third blocks) and abstract layers (the ones in the unfolded LTCN model). Hereinafter there will be no distinction among the different types of layers. Likewise, we assume that $w_{ji}^{(t)}$, $b_i^{(t)}$ and $c_i^{(t)}$ are the weights that connect two neurons, the neuron's bias, and the input weight in the tth layer, respectively.

Eq. (4) displays the regularized *squared hinge loss* function to be minimized by the algorithm, which is aimed at maximizing the margins between labels,

$$\mathcal{E} = \frac{1}{2}\Big(\sum_{i=1}^{L} \max\{0, l_i(k) a_i^{(T)}(k)\}^2 + \frac{\lambda^{(t)}}{2} \sum_{t=1}^{T} \mathcal{R}^{(t)}\Big) \tag{4}$$

such that

$$\mathcal{R}^{(t)} = \sum_{j=1}^{M^{(t-1)}} \sum_{i=1}^{M^{(t)}} \Big(w_{ji}^{(t)}\Big)^2 \tag{5}$$

where L is the number of labels, $l_i(k)$ takes -1 when the ith label is not associated with the kth instance, otherwise it takes 1. Also, $a_i^{(T)}(k)$ is the activation value of the ith neuron in the output layer whereas $\lambda \geq 0$ is a constant to control the ℓ_2 regularization component.

Case 1. When $t = T$, the partial derivative of the global error $\mathcal{E}$ is computed as follows:

$$\frac{\partial \mathcal{E}}{\partial a_i^{(t)}(k)} = -l_i(k) + a_i^{(t)}(k) l_i(k)^2. \tag{6}$$

Case 2. When $1 < t < T$, the partial derivative of the global error $\mathcal{E}$ is calculated as follows:

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial a_i^{(t)}(k)} &= \sum_{j=1}^{M^{(t+1)}} \frac{\partial \mathcal{E}}{\partial a_j^{(t+1)}(k)} \times \frac{\partial a_j^{(t+1)}(k)}{\partial a_i^{(t)}(k)} \\ &= \sum_{j=1}^{M^{(t+1)}} \frac{\partial \mathcal{E}}{\partial a_j^{(t+1)}(k)} \times \frac{\partial a_j^{(t+1)}(k)}{\partial \bar{a}_j^{(t+1)}(k)} \times \frac{\partial \bar{a}_j^{(t+1)}(k)}{\partial a_i^{(t)}(k)} \\ &= \sum_{j=1}^{M^{(t+1)}} \frac{\partial \mathcal{E}}{\partial a_j^{(t+1)}(k)} \times \frac{\partial f}{\partial \bar{a}_j^{(t+1)}(k)} \times w_{ij}^{(t+1)} \end{aligned} \tag{7}$$

such that $\bar{a}_j^{(t+1)}(k)$ is the raw activation value of the jth neuron, while $f(\cdot)$ is the transfer function.

Once $\partial \mathcal{E} / \partial a_i^{(t)}(k)$ have been calculated for all layers, we can compute the partial derivatives of the global error with respect to the target parameters:

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial w_{ji}^{(t)}(k)} &= \frac{\partial \mathcal{E}}{\partial a_i^{(t)}(k)} \times \frac{\partial a_i^{(t)}(k)}{\partial \bar{a}_i^{(t)}(k)} \times \frac{\partial \bar{a}_i^{(t)}(k)}{\partial w_{ji}^{(t)}(k)} + \lambda^{(t)} w_{ji}^{(t)}(k) \\ &= \frac{\partial \mathcal{E}}{\partial a_i^{(t)}(k)} \times \frac{\partial f}{\partial \bar{a}_i^{(t)}(k)} \times a_j^{(t-1)} + \lambda^{(t)} w_{ji}^{(t)}(k), \end{aligned} \tag{8}$$

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial b_i^{(t)}(k)} &= \frac{\partial \mathcal{E}}{\partial a_i^{(t)}(k)} \times \frac{\partial a_i^{(t)}(k)}{\partial \bar{a}_i^{(t)}(k)} \times \frac{\partial \bar{a}_i^{(t)}(k)}{\partial b_i^{(t)}(k)} \\ &= \frac{\partial \mathcal{E}}{\partial a_i^{(t)}(k)} \times \frac{\partial f}{\partial \bar{a}_i^{(t)}(k)}, \end{aligned} \tag{9}$$

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial c_i^{(t)}(k)} &= \frac{\partial \mathcal{E}}{\partial a_i^{(t)}(k)} \times \frac{\partial a_i^{(t)}(k)}{\partial \bar{a}_i^{(t)}(k)} \times \frac{\partial \bar{a}_i^{(t)}(k)}{\partial c_i^{(t)}(k)} \\ &= \frac{\partial \mathcal{E}}{\partial a_i^{(t)}(k)} \times \frac{\partial f}{\partial \bar{a}_i^{(t)}(k)} \times a_i^{(0)}. \end{aligned} \tag{10}$$

Eq. (11) shows the gradient vector for the model parameters attached to the tth layer,

$$\nabla^{(t)} \mathcal{E} = \left(\frac{\partial \mathcal{E}}{\partial w_{ji}^{(t)}(k)}, \ldots, \frac{\partial \mathcal{E}}{\partial b_i^{(t)}(k)}, \ldots, \frac{\partial \mathcal{E}}{\partial c_i^{(t)}(k)} \right). \tag{11}$$

In addition to the regularization, the MP-LTCN model will likely benefit from adopting a dropout strategy (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) to control overfitting. The dropout seems to be especially useful in the first block, where a large number of problem features might lead to a multilayered structure with many parameters. However, the mathematical formulation of the learning method holds as we only need to determine which weights will be updated in each epoch.

5. Numerical simulations

In this section, we conduct some numerical simulations to evaluate the performance of our model.

5.1. Dataset characterization

Aiming at conducting the numerical simulations, we will use 12 multi-label datasets taken from the RUMDR repository (Charte, Charte, Rivera, del Jesus, & Herrera, 2016). In these problems (see Table 1), the number of objects (K) ranges from 207 to 13,929, the number of features (N) goes from 72 to 1,717, and the number of labels (L) from 4 to 53. Also, we report the sparseness rate (SR), which is defined as the number of zero elements in the feature matrix divided by the size of the feature matrix. Likewise, we show the *TCS* metric (Charte, Rivera, del Jesus, & Herrera, 2016) as a theoretical complexity indicator. The measure is calculated as the product of the number of attributes, the number of labels, and the number of distinct label sets. We transfer the values to a logarithmic scale to avoid reporting large values. The higher the value, the more complex the pre-processing of the dataset. Remark that *TCS* values are logarithmic. Therefore, a difference of only one unit implies one order of magnitude lower or higher.

More details about these datasets are given next. D1 is about classifying music into emotions that it evokes according to the Tellegen–Watson–Clark mood model. D2 and D8 problems try to predict the subcellular locations of proteins according to their sequences. They contain 1,392 sequences for Gram-negative bacterial (Gnegative) species. The Gene Ontology (GO) and PseAAC features (including 20 amino acid, 20 pseudo-amino acid, and

Table 1
Datasets used during the simulations.

ID	Name	Domain	K	N	L	SR	TCS
D1	emotions	Music	593	72	6	0.005	9.360
D2	GnegativePseAAC	Biology	1,392	440	8	0.490	11.11
D3	GpositivePseAAC	Biology	519	440	4	0.476	9.410
D4	scene	Image	2,407	294	6	0.012	10.18
D5	VirusPseAAC	Biology	207	440	6	0.488	10.71
D6	yeast	Biology	2,417	103	14	0.001	12.56
D7	enron	Text	1,702	1,001	53	0.916	17.50
D8	GnegativeGO	Biology	1,392	1,717	8	0.995	12.47
D9	GpositiveGO	Biology	519	912	4	0.991	10.14
D10	ohsumed	Text	13,929	1,002	23	0.959	17.09
D11	slashdot	Text	3,782	1,079	22	0.992	15.12
D12	VirusGO	Biology	207	749	6	0.985	11.24

400 dipeptide components) are provided. The decision classes consist of eight subcellular locations. Similarly, D3 and D9 contain 519 sequences for Gram-positive species and four subcellular locations. D4 contains 2,407 images and six classes: beach, sunset, fall foliage, field, mountain, and urban. Each image is described with 294 visual numeric features corresponding to spatial color moments in the LUV space. D5 and D12 contain 207 sequences for Virus species, described by GO and PseAAC features, respectively. The labels correspond to six subcellular locations. D6 contains micro-array expressions and phylogenetic profiles for 2,417 yeast genes. Each gen is annotated with a subset of 14 functional categories such as metabolism, energy, etc. D7 is a subset of Enron Email Corpus. It is based on a collection of email messages categorized into 53 topic categories, such as company strategy, humor, and legal advice. D10 includes medical abstracts that were associated with 23 cardiovascular disease categories. D11 consists of article blurbs with subject categories representing the label space, which were mined from http://slashdot.org.

5.2. Exploring the ML-LTCN classifier

The first simulations aim to explore the algorithm's performance for different configurations. Particularly, we are interested in studying how the model behaves when altering the learning rate and the number of abstract layers in the second neural block. Likewise, we want to determine a default parameter setting (i.e., parameter values that lead to acceptable predictions overall).

All simulations conducted in this research use an ML-LTCN model with three hidden layers in the first block, such as each layer is one third smaller than the previous one. For example, if we have $N = 1000$ problem features and we want to extract $H = 100$ high-level features with $S = 3$ hidden layers, then the first layer would have 660 neurons, the second one would have 330 neurons, while the last one would have 100 neurons.

Furthermore, we use three hidden layers in the first neural block such as each layer is one third smaller than the previous one, and the number of hidden features is set to 100 for larger datasets ($>$ 440 features) and 50 for smaller ones ($<$ 400 features). The number of abstract layers in the second block varies from one to five. The regularization parameter (see Eq. (4)) was set to $\lambda^{(t)}$=1.0E−2 in all layers while the dropout parameter was set to 0.5. Finally, we use the ADAM optimization method (Kingma & Ba, 2015) such that the number of epochs was set to 200, and the learning rate changes from 0.001 to 0.005. We adopted a batch strategy in the backpropagation learning to keep the simulation times low in these experiments.

Fig. 2 shows the simulation results when varying the number of abstract layers in the LTCN model and the learning rate. Aiming to measure the ML-LTCN classifier's performance, we have adopted the Hamming Loss (HL) measure (Herrera et al., 2016b). This measure quantifies the fraction of labels that are incorrectly predicted. In the subsequent experiments, we will also use the accuracy and the F-measure as performance measures.

Overall, we can conclude that our model performs well for most datasets when increasing the number of abstract layers while keeping the learning rate small. Likewise, the results advocate against using more abstract layers with rather larger learning rates. There are problems such as the D7 dataset in which our proposal performs similarly regardless of the parameter setting.

In the remaining simulations, we will use three abstract layers with a learning rate of 0.001 while retaining the settings of other parameters as described above. This means that we will not optimize the network's parameters for each problem. Instead, we found it more convenient to compare our proposal against other classifiers when using default parameter values. The reason behind this decision is that hyperparameter tuning is a time-consuming process that usually demands extra data.

Next, we will evaluate the effect of both the dropout and the regularization on the algorithm's performance. Fig. 3 shows the HL values attached to each dataset when suppressing these strategies individually. The baseline model refers to the ML-LTCN algorithm using both dropout and regularization, as described above.

Overall, the simulation results suggest that the dropout is more effective when contrasted with the ℓ_2 weight regularization. In the case of datasets D2, D3 and D5, the negative effect of not using dropout becomes more evident. Similarly, the reader can observe that the results do not change significantly when suppressing the regularization component. Notice that no controlling the overfitting at all is not an option as the results often deteriorate perceptibly. Since there seems to be little harm in using both strategies together, we will use the baseline model in the remaining simulations.

5.3. Comparison against state-of-the-art algorithms

In this subsection, we compare the prediction capabilities of our model with the state-of-the-art classifiers, which are implemented in the MULAN (Tsoumakas, Spyromitros-Xioufis, Vilcek, & Vlahavas, 2011) software tool and the Scikit-multilearn library (SzymaĹ, Kajdanowicz, et al., 2019). These state-of-the-art algorithms are briefly described below.

- Backpropagation for Multi-Label Learning (BP-MLL) (Mańdziuk & Żychowski, 2019; Zhang & Zhou, 2006). This feedforward neural network for MLC problems uses an error function to capture the correlation among the labels. This function penalizes the predictions that include labels that are not truly relevant to the processed instance.
- RAndom k-labELsets (RAkEL) (Padmashani, Nivaashini, & Vidhyapriya, 2019; Tsoumakas & Vlahavas, 2007). This method generates random subsets of labels while training a multiclass classifier for each subset. RAkEL involves two essential parameters, c and k. The former determines the number of classifiers, while the latter denotes the length of the label sets.
- Multi-label kNN (Zhang & Zhou, 2007; Zhu, et al., 2020) is an adaptation of kNN to the MLC scenario. This method uses the *a priori* and conditional probabilities of each label to determine the label set of unseen instances.
- Binary Relevance (BR) (Godbole & Sarawagi, 2004; Zhang et al., 2018). This method generates a binary dataset per each label such that positive patterns are the ones associated with the label. When a new pattern is presented to the model, the output will be the set of positive classes.

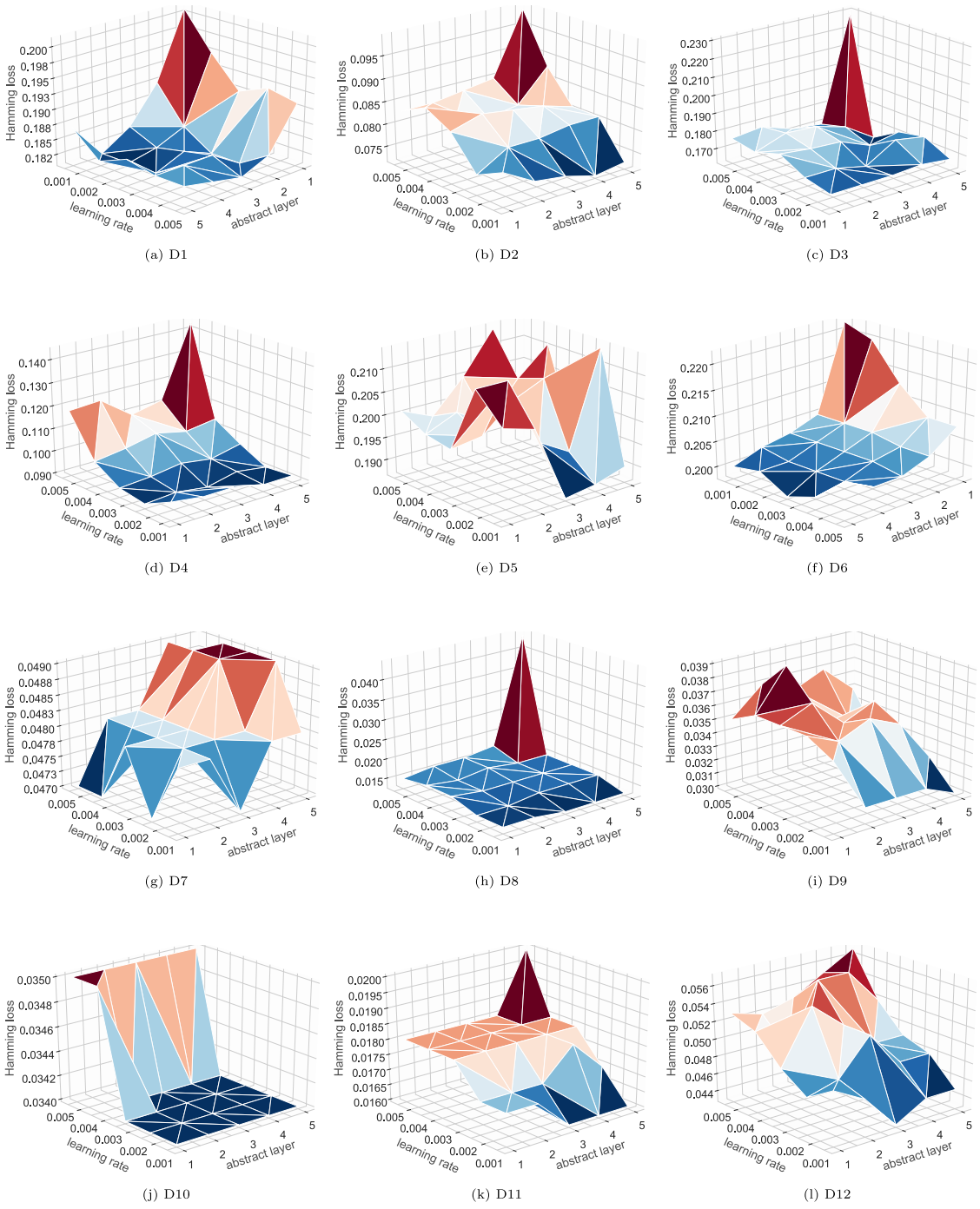

Fig. 2. Performance of the proposed ML-LTCN method in terms of HL values when varying the number of abstract layers in the second block and the learning rate of the ADAM optimizer. Some figures have been rotated for better visualization.

- Adaboost (ABoost) (Bogaert, Lootens, Van den Poel, & Ballings, 2019; Schapire & Singer, 1999) is an adaptation of the Adaboost method for MLC. It breaks down the problem into a binary problem where each test instance is classified according to its label association.
- Multi-label Hierarchical Adaptive Resonance Associative Map Neural Network (ML-HARAM) (Benites & Sapozhnikova, 2015, 2017). This neural system was initially developed for text datasets with high dimensionality. Overall, it aims to increase the classification speed by adding an extra layer of adaptive resonance theory to group the learned prototypes into large clusters.
- Multi-label Twin Support Vector Machine (ML-TSVM) (Chen, Shao, Li, & Deng, 2016). This algorithm uses the one-against-all strategy to construct multiple non-parallel hyperplanes from solving a series of quadratic programming problems. Each hyperplane is closer to its corresponding class but far away from the others.

We use the same default parameter settings provided by the MULAN and Scikit-multilearn library, thus no algorithm performs hyperparameter tuning. It goes without saying that we are especially interested in evaluating how ML-LTCN performs when compared with the BP-MLL algorithm as both are neural network solutions. In addition, we are also interested in the achievements

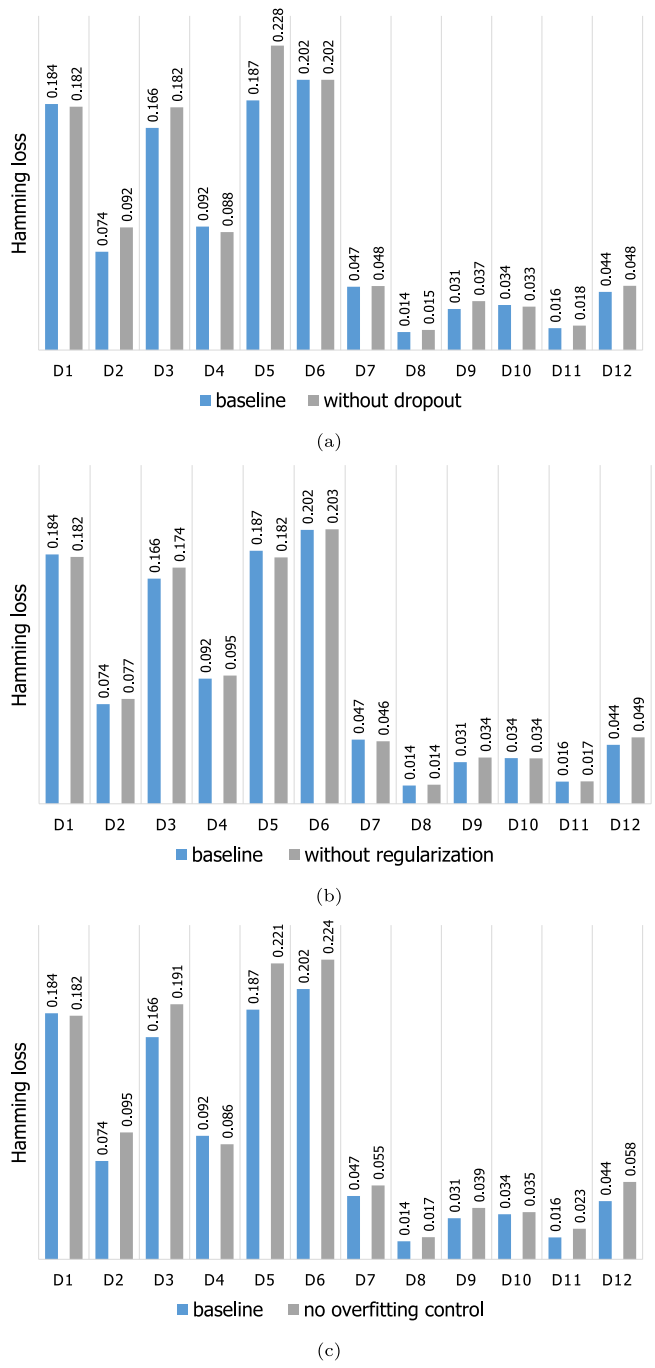

Fig. 3. Performance of the ML-LTCN method (a) without using dropout, (b) without using regularization, and (c) without controlling the overfitting at all. In this experiment, the baseline refers to an ML-LTCN using both dropout and ℓ_2 regularization.

of the ML-LTCN in datasets D7 to D12, as they have the highest sparseness rate (SR over 90%) and include the three of the most complex problems D7, D10 and D11.

Table 2 displays the average HL values attached with each classifier after performing a 10-fold cross-validation process. The best (lowest) values were highlighted in bold. The reader can observe that the proposed ML-LTCN algorithm outperforms (in terms of HL value) the state-of-the-art models in 7 out of 12 benchmark problems while ranking second in the remaining ones.

Table 3 displays the average accuracy reported by each classifier after performing a 10-fold cross-validation process. The best (largest) values were highlighted in bold. The example-based accuracy is defined as the proportion of the predicted correct labels to the total number of labels for that instance. The overall accuracy is computed as the average across all instances in the dataset. According to this measure, our model stands as the best-performing algorithm for all benchmark problems.

Table 4 portrays the average F-measure values reported by each model after performing 10-fold cross-validation. Again, the best (largest) values were highlighted in bold. The F-measure can be understood as the harmonic mean of precision and recall. The ML-LTCN method outperforms the state-of-the-art models in 7 out of 12 problems while performing well in the remaining ones.

Focusing the analysis on the last six datasets (D7 to D12 with SR over 90%), we observe that ML-LTCN outperforms its competitors. In the case of the HL values (Table 2), ML-LTCN was better in four (D7, D8, D10 and D11) out of six problems. Meanwhile, it ranked second in the other two (D9 and D12), only surpassed by the RAkEL method. In the case of accuracy reported by each method (Table 3), our proposal overtakes their competitors, including the last six problems. In terms of F-measure (Table 4), ML-LTCN achieved first place in three out of six problems (D8, D9, and D12), second place in two (D7 and D11), and fourth place

Table 2
Hamming loss reported by each algorithm (the smaller, the better).

ID	BP-MLL	RAkEL	ML-*k*NN	BR	ABoost	ML-HARAM	ML-TSVM	ML-LTCN
D1	0.212	0.217	0.195	0.252	0.304	0.226	0.239	**0.184**
D2	0.563	0.091	0.081	0.145	0.130	0.139	0.124	**0.073**
D3	0.581	0.233	**0.154**	0.182	0.251	0.235	0.225	0.166
D4	0.273	0.101	**0.087**	0.241	0.179	0.124	0.158	0.092
D5	0.590	0.217	0.202	0.228	0.202	0.252	0.212	**0.186**
D6	0.221	0.226	**0.193**	0.303	0.231	0.242	0.311	0.202
D7	0.211	0.048	0.052	0.217	0.062	0.071	0.063	**0.047**
D8	0.651	0.014	0.018	0.027	0.130	0.039	0.088	**0.013**
D9	0.609	**0.028**	0.040	0.050	0.254	0.106	0.191	0.030
D10	0.645	0.058	0.070	0.142	0.072	0.096	0.073	**0.033**
D11	0.629	0.041	0.051	0.068	0.053	0.035	0.038	**0.016**
D12	0.649	**0.038**	0.065	0.069	0.202	0.258	0.174	0.043

Table 3
Accuracy reported by each algorithm (the larger, the better).

ID	BP-MLL	RAkEL	ML-*k*NN	BR	ABoost	ML-HARAM	ML-TSVM	ML-LTCN
D1	0.573	0.512	0.532	0.529	0.048	0.268	0.150	**0.816**
D2	0.100	0.535	0.543	0.574	0	0.445	0.223	**0.927**
D3	0.246	0.498	0.587	0.616	0	0.457	0.170	**0.829**
D4	0.368	0.624	0.667	0.452	0	0.552	0.266	**0.907**
D5	0.170	0.317	0.177	0.341	0	0.230	0.009	**0.791**
D6	0.522	0.484	0.516	0.419	0.335	0.198	0.004	**0.798**
D7	0.221	0.433	0.331	0.195	0.150	0.036	0.031	**0.952**
D8	0.104	0.943	0.920	0.882	0	0.797	0.361	**0.986**
D9	0.220	0.942	0.893	0.884	0	0.745	0.264	**0.967**
D10	0.054	0.403	0.047	0.292	0	0.066	0.202	**0.966**
D11	0.047	0.372	0.072	0.376	0	0.450	0.326	**0.983**
D12	0.184	0.891	0.758	0.821	0	0.142	0.163	**0.957**

Table 4
F-measure reported by each algorithm (the larger, the better).

ID	BP-MLL	RAkEL	ML-*k*NN	BR	ABoost	ML-HARAM	ML-TSVM	ML-LTCN
D1	0.658	0.597	0.613	0.632	0.065	0.659	0.510	**0.661**
D2	0.164	0.552	0.548	0.633	0	0.548	0.393	**0.669**
D3	0.356	0.503	0.590	0.642	0	0.609	0.280	**0.664**
D4	0.491	0.646	0.681	0.566	0	0.702	0.474	**0.721**
D5	0.265	0.354	0.193	0.394	0	**0.453**	0.035	0.448
D6	**0.635**	0.600	0.620	0.538	0.456	0.614	0.112	0.603
D7	0.342	**0.545**	0.428	0.307	0.230	0.392	0.327	0.517
D8	0.180	0.951	0.929	0.905	0	0.858	0.520	**0.959**
D9	0.306	0.944	0.896	0.900	0	0.814	0.418	**0.948**
D10	0.099	**0.467**	0.054	0.399	0	0.213	0.236	0.127
D11	0.088	0.389	0.075	0.435	0	**0.626**	0.213	0.571
D12	0.288	0.909	0.772	0.851	0	0.516	0.279	**0.918**

Table 5
Training time for each algorithm per dataset (in seconds).

ID	BP-MLL	RAkEL	ML-*k*NN [a]	BR	ABoost	ML-ARAM	ML-TSVM	ML-LTCN
D1	542	1,376	N/A	157	541	246	1,368	1,726
D2	14,936	13,168	N/A	358	9,383	2,930	26,265	3,587
D3	4,915	1,023	N/A	59	923	461	3,049	2,285
D4	10,941	16,378	N/A	348	7,382	3,837	23,166	3,505
D5	1,961	1,021	N/A	34	283	45	2,635	1,822
D6	2,889	28,304	N/A	321	4,324	1,233	28,917	1,952
D7	89,848	269,566	N/A	2,894	70,959	462	1,72,253	9,522
D8	192,446	23,077	N/A	1,780	36,184	233	19,910	17,360
D9	24,500	791	N/A	170	2,185	202	2,265	4,030
D10	822,808	6,224,029	N/A	91,088	1,499,204	230,989	2,369,682	57,776
D11	235,175	355,111	N/A	9,985	164,778	3,848	224,379	19,579
D12	6,653	559	N/A	81	848	16	1,490	2,366

[a]In this lazy learner, the training phase technically does not exist since all computations are done during the test phase. Therefore, the training time is O(1).

(out of seven methods) in D10. Overall, the results suggest that our method is a competitive solution for potentially sparse MLC problems.

5.4. Further discussion

In this subsection, we will elaborate on the performance of the state-of-the-art methods on datasets associated with larger SR values (D7 to D12).

In our experiments, RAkEL ranked second on the last six datasets, only surpassed by our proposal. RAkEL trains one multi-class classifier for each subset of labels. This ensemble method, RAkEL balances its bias and variance, being less sensitive to sparse and high dimensional data, thus improving its performance (Ren, Zhang, & Suganthan, 2016).

BR and ML-kNN have similar behavior in the datasets of interest. The first one relies on a collection of binary classifiers and a strategy to combine their outputs. The second one internally works as a BR classifier since a separate set of a priori and conditional probabilities are independently computed for each label. Both methods disregard any potential correlations among labels (Herrera, Charte, Rivera, & del Jesús, 2016a), which is often considered a drawback. However, in Yu, Pedrycz, and Miao (2014) the authors argue that exploiting label correlations might introduce unnecessary or misleading limitations on instances that do not contain such correlations.

ML-HARAM and BP-MLL are neural network models. ML-HARAM implements a winner-take-all approach where a limited number of neurons is activated. This is equivalent to penalizing the model (Aggarwal, 2018), which translates into learning sparse representations. In contrast, the performance of BP-MLL was rather poor in datasets D7 to D12. This method has an input layer with one neuron per each input attribute, while the number of output neurons corresponds with the number of labels. These nodes use a sigmoid activation function, where even small weights produce half of their saturation regime, decreasing the network performance in sparse data sets (Glorot, Bordes, & Bengio, 2011).

In MLTSVM, each resulting hyper-plane is closer to its corresponding instances but far away from the others (Chen et al., 2016). However, the lack of pre-processing might have affected the hyperplane generation, thus leading to sub-optimal predictions. A similar behavior revealed ABoost, even though it is an ensemble-based method, reported the worst Hamming loss, accuracy, and F-measure.

5.5. Computational complexity

Before concluding the paper, we will study the temporal complexity of our proposal.

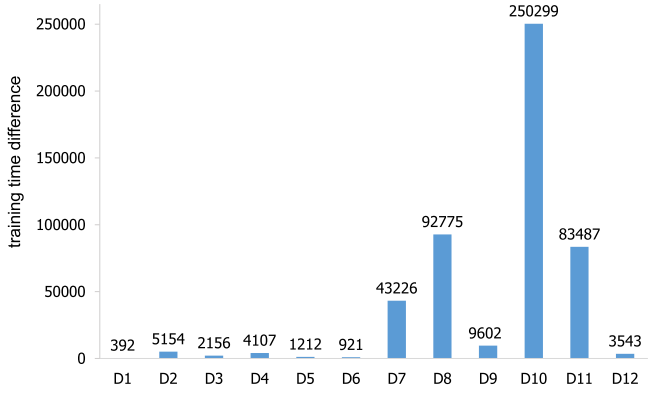

Fig. 4. Training time difference when suppressing the first neural block in the baseline architecture.

Table 5 displays the training time (in seconds) of all algorithms across the datasets. It seems opportune to mention that all experiments in this paper were performed in an HP EliteBook laptop (with Intel Core i5 8-th generation processor). The results show that BR is the fastest algorithm in this experiment; however, it does not produce impressive prediction results. Therefore, it seems reasonable to compare the training time of the best-perming algorithms, that is to say, RAkEL and ML-LTCN. The results show that our proposal reports shorter training times in 7 out of 12 datasets (D2, D4, D6, D7, D8, D10, D11) while being competitive for D1 and D5.

Finally, we measure the training time when removing the first neural block from the ML-LTCN architecture. More explicitly, the values in Fig. 4 concern the training time differences when suppressing that block in the baseline architecture discussed in Section 3.

The results of this experiment illustrate the advantages of extracting high-level features in problems with large SR values (e.g., D7, D8, D9, D10, and D11). These features contain relevant information while being encoded in a lower-dimensional space. This neural block not only brings a reduction in the training time but also leads to better predictions. For example, we noticed that suppressing the first block causes the average HL to increase from 0.09 to 0.12. This behavior holds even if we increase the number of abstract layers in the second block.

6. Concluding remarks

In this paper, we have presented an algorithm termed ML-LTCN to deal with potentially sparse MLC problems. The proposed architecture involves three neural blocks. While the first block is

devoted to extracting the high-level features from sparse data, the remaining ones are oriented to computing the class labels to be assigned to unseen instances. When it comes to the learning algorithm, we have adopted a squared hinge loss function (used by support vector machines) since it allows increasing the margins between positive and negative labels. The theoretical advantage of having well-separated labels is that the model becomes less sensitive to the threshold determining the labels attached to each instance. This eventually translates into having fewer misclassifications.

The simulations suggest that the proposed model behaves well when adding more abstract layers to the second block while using rather small learning rate values. Moreover, the comparison against the state-of-the-art methods shows that our model is a fair competitor in terms of Hamming Loss, example-based accuracy and F-measure. It has not escaped our notice that we could have obtained even better results by using a smaller batch size. Despite this fact, we preferred to keep the simulation time as low as possible. The further research steps will focus on exploiting the cognitive component of LTCNs, which allows injecting prior knowledge into the model. At the same time, we will try to extract features from the problem labels in a bidirectional fashion, thus notably reducing the burden of training the recurrent block.

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.

Acknowledgments

The authors would like to sincerely thank Isel Grau from the Vrije Universiteit Brussel, Belgium, who pointed out the advantages of using the squared hinge function instead of the mean squared error. This paper was partially supported by the Program CONICYT FONDECYT de Postdoctorado, Chile through the project 3200284.

References

Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B., et al. (2016). Youtube-8m: A large-scale video classification benchmark. arXiv preprint arXiv:1609.08675.

Adomavicius, G., & Zhang, J. (2012). Impact of data characteristics on recommender systems performance. *ACM Transactions on Management Information Systems*, *3*(1).

Aggarwal, C. C. (2018). Machine learning with shallow neural networks. In *Neural networks and deep learning* (pp. 53–103). Springer, chapter 2.

Bello, M., Nápoles, G., Vanhoof, K., & Bello, R. (2019). Methods to edit multi-label training sets using rough sets theory. In T. Mihálydeák, F. Min, G. Wang, M. Banerjee, I. Düntsch, Z. Suraj, & D. Ciucci (Eds.), *Rough sets* (pp. 369–380). Springer International Publishing.

Benites, F., & Sapozhnikova, E. (2015). HARAM: A hierarchical ARAM neural network for large-scale text classification. In *2015 IEEE international conference on data mining workshop* (pp. 847–854). IEEE.

Benites, F., & Sapozhnikova, E. (2017). Improving scalability of ART neural networks. *Neurocomputing*, *230*(January), 219–229.

Bogaert, M., Lootens, J., Van den Poel, D., & Ballings, M. (2019). Evaluating multi-label classifiers and recommender systems in the financial service sector. *European Journal of Operational Research*, *279*(2), 620–634.

Boutell, M. R., Luo, J., Shen, X., & Brown, C. M. (2004). Learning multi-label scene classification. *Pattern Recognition*, *37*(9), 1757–1771.

Charte, F., Charte, D., Rivera, A., del Jesus, M. J., & Herrera, F. (2016). R ultimate multilabel dataset repository. In *International conference on hybrid artificial intelligence systems* (pp. 487–499). Springer.

Charte, F., Rivera, A., del Jesus, M. J., & Herrera, F. (2016). On the impact of dataset complexity and sampling strategy in multilabel classifiers performance. In *International conference on hybrid artificial intelligence systems* (pp. 500–511). Springer.

Chen, S.-F., Chen, Y.-C., Yeh, C.-K., & Wang, Y.-C. F. (2018). Order-free RNN with visual attention for multi-label classification. In *Thirty-second AAAI conference on artificial intelligence*.

Chen, W.-J., Shao, Y.-H., Li, C.-N., & Deng, N.-Y. (2016). MLTSVM: a novel twin support vector machine to multi-label learning. *Pattern Recognition*, *52*, 61–74.

Chen, G., Ye, D., Xing, Z., Chen, J., & Cambria, E. (2017). Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. In *2017 International joint conference on neural networks* (pp. 2377–2383). IEEE.

Choi, K., Fazekas, G., & Sandler, M. B. (2016). Automatic tagging using deep convolutional neural networks. In *ISMIR*.

Duda, R. O., Hart, P. E., & Stork, D. G. (2012). *Pattern classification* (2nd ed.). John Wiley & Sons.

Elisseeff, A., & Weston, J. (2002). A kernel method for multi-labelled classification. In *Advances in neural information processing systems* (pp. 681–687).

García, S., Luengo, J., & Herrera, F. (2015). Data preprocessing in data mining. In *Data preprocessing in data mining (vol. 72)* (pp. 1–17). Springer International Publishing.

Gibaja, E., & Ventura, S. (2014). Multi-label learning: a review of the state of the art and ongoing research. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *4*(6), 411–444.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 315–323).

Godbole, S., & Sarawagi, S. (2004). Discriminative methods for multi-labeled classification. In *Pacific-Asia conference on knowledge discovery and data mining* (pp. 22–30). Springer.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Herrera, F., Charte, F., Rivera, A. J., & del Jesús, M. J. (2016a). Adaptation-based classifiers abstract. In *Multi-label classification. problem analysis, metrics and techniques* (pp. 81–99). Springer.

Herrera, F., Charte, F., Rivera, A. J., & Del Jesus, M. J. (2016b). Multilabel classification. In *Multilabel classification* (pp. 17–31). Springer.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Janwe, N. J., & Bhoyar, K. K. (2018). Multi-label semantic concept detection in videos using fusion of asymmetrically trained deep convolutional neural networks and foreground driven concept co-occurrence matrix. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, *48*(8), 2047–2066.

Kingma, D. P., & Ba, J. (2015). Adam: a method for stochastic optimization. In *Proceedings of the 3rd international conference on learning representations*.

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 972–981). Curran Associates Inc.

Kongsorot, Y., & Horata, P. (2014). Multi-label classification with extreme learning machine. In *2014 6th international conference on knowledge and smart technology* (pp. 81–86). IEEE.

Krishnan, R. G., Liang, D., & Hoffman, M. D. (2018). On the challenges of learning with inference networks on sparse, high-dimensional data. In *International conference on artificial intelligence and statistics (vol. 84)* (pp. 143–151).

Law, A., Chakraborty, K., & Ghosh, A. (2017). Functional link artificial neural network for multi-label classification. In *International conference on mining intelligence and knowledge exploration* (pp. 1–10). Springer.

Law, A., & Ghosh, A. (2019). Multi-label classification using a cascade of stacked autoencoder and extreme learning machines. *Neurocomputing*, *358*, 222–234.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Lian, S.-m., Liu, J.-w., Lu, R.-k., & Luo, X.-l. (2019). Captured multi-label relations via joint deep supervised autoencoder. *Applied Soft Computing*, *74*, 709–728.

Mańdziuk, J., & Żychowski, A. (2019). Dimensionality reduction in multilabel classification with neural networks. In *2019 International joint conference on neural networks* (pp. 1–8). IEEE.

Nápoles, G., Salmeron, J. L., & Vanhoof, K. (2019). Construction and supervised learning of long-term grey cognitive networks. *IEEE Transactions on Cybernetics*, 1–10.

Nápoles, G., Vanhoenshoven, F., Falcon, R., & Vanhoof, K. (2020). Nonsynaptic error backpropagation in long-term cognitive networks. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(3), 865–875.

Nápoles, G., Vanhoenshoven, F., & Vanhoof, K. (2019). Short-term cognitive networks, flexible reasoning and nonsynaptic learning. *Neural Networks*, *115*, 72–81.

Padmashani, R., Nivaashini, M., & Vidhyapriya, R. (2019). Rakel algorithm and mahalanobis distance-based intrusion detection system against network intrusions. In *International conference on artificial intelligence, smart grid and smart city applications* (pp. 689–696). Springer.

Read, J., Pfahringer, B., & Holmes, G. (2008). Multi-label classification using ensembles of pruned sets. In *8th IEEE international conference on data mining* (pp. 995–1000). IEEE.

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2009). Classifier chains for multi-label classification. In *Joint European conference on machine learning and knowledge discovery in databases* (pp. 254–269). Springer.

Ren, Y., Zhang, L., & Suganthan, P. N. (2016). Ensemble classification and regression-recent developments, applications and future directions. *IEEE Computational Intelligence Magazine*, *11*(1), 41–53.

Rios, A., & Kavuluru, R. (2015). Convolutional neural networks for biomedical text classification: application in indexing biomedical articles. In *Proceedings of the 6th ACM conference on bioinformatics, computational biology and health informatics* (pp. 258–267). ACM.

Schapire, R. E., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, *37*(3), 297–336.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

SzymaĹ, P., Kajdanowicz, T., et al. (2019). Scikit-multilearn: A python library for multi-label classification. *Journal of Machine Learning Research*, *20*(6), 1–22.

Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., & Vlahavas, I. (2011). Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, *12*(Jul), 2411–2414.

Tsoumakas, G., & Vlahavas, I. (2007). Random k-labelsets: An ensemble method for multilabel classification. In *European conference on machine learning* (pp. 406–417). Springer.

Vens, C., Struyf, J., Schietgat, L., Džeroski, S., & Blockeel, H. (2008). Decision trees for hierarchical multi-label classification. *Machine Learning*, *73*(2), 185.

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., & Xu, W. (2016). Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2285–2294).

Wei, Y., Xia, W., Lin, M., Huang, J., Ni, B., Dong, J., et al. (2015). HCP: A flexible CNN framework for multi-label image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*(9), 1901–1907.

Wu, F., Wang, Z., Zhang, Z., Yang, Y., Luo, J., Zhu, W., et al. (2015). Weakly semi-supervised deep learning for multi-label image annotation. *IEEE Transactions on Big Data*, *1*(3), 109–122.

Yeh, C.-K., Wu, W.-C., Ko, W.-J., & Wang, Y.-C. F. (2017). Learning deep latent space for multi-label classification. In *Thirty-first AAAI conference on artificial intelligence* (pp. 2838—2844).

Yu, Y., Pedrycz, W., & Miao, D. (2014). Multi-label classification by exploiting label correlations. *Expert Systems with Applications*, *41*(6), 2989–3004.

Zhang, M.-L. (2009). Ml-rbf: Rbf neural networks for multi-label learning. *Neural Processing Letters*, *29*(2), 61–74.

Zhang, M.-L., Li, Y.-K., Liu, X.-Y., & Geng, X. (2018). Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, *12*(2), 191–202.

Zhang, M.-L., & Zhou, Z.-H. (2006). Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, *18*(10), 1338–1351.

Zhang, M.-L., & Zhou, Z.-H. (2007). ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, *40*(7), 2038–2048.

Zhang, M.-L., & Zhou, Z.-H. (2013). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *26*(8), 1819–1837.

Zhu, J., Liao, S., Lei, Z., & Li, S. Z. (2017). Multi-label convolutional neural network based pedestrian attribute classification. *Image and Vision Computing*, *58*, 224–229.

Zhu, D., Zhu, H., Liu, X., Li, H., Wang, F., Li, H., et al. (2020). CREDO: Efficient and privacy-preserving multi-level medical pre-diagnosis based on ML-kNN. *Information Sciences*, *514*, 244–262.