DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL' INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

TOPOLOGY OF A NEURAL NETWORK FOR MULTI-LABEL CLASSIFICATION

Relatore
Prof. Loris Nanni

Laureando
Luca Trambaiollo

Anno Accademico 2021/2022
Padova, 14 marzo 2022

Index

Abstract

The objective of this work is to maximize the performance of a neural network that deals with multi-label classification, therefore it can associate several class labels to a sample. To achieve this, the network topology and therefore the structure has been modified: inserting / removing and exchanging the order of the various levels, modifying the value of the parameters linked to them (e.g. The number of hidden levels or the number of filters). This work proposes a new topology called LSTM_GRU composed by a Long Short-Term Memory and Gated Recurrent Units trained with variants of the Adam optimization approach. The proposed neural network topology is also combined with Incorporating Multiple Clustering Centers (IMCC), which further boosts classification performance.

Multiple experiments on twelve data sets representing a wide variety of multilabel tasks demonstrate the robustness of the ensemble proposed, which is shown to overperform the state-of-the-art.

Introduction

Multilabel learning deals with the common problem of associating a sample with multiple labels and has been successfully applied in various domains [1] such as tag recommendation [2], bioinformatics [3-6], information retrieval [7, 8], speech recognition [9, 10], and user reviews and negative comment classification [11, 12], to list a few. A dependable and robust method for enhancing the performance of multilabel models is to generate ensembles of classifiers [13, 14], with bagging one of the most popular method for producing ensembles that perform well across several multilabel classification benchmarks [15, 16].

In this paper, is proposed a new topology named LSTM_GRU for multilabel classification that combines in a sequential way Long Short-Term Memory (LSTM), dropout layer with a probability of 0.4%, Gated Recurrent Units (GRU) [17], another dropout layer with probability of 0.4%, a fully connected layer and a final sigmoid. A GRU can be conceptualized as a simplified Bidirectional Long Short-Term Memory (LSTM), both of which model hidden temporal states with some gating mechanisms and suffer from intermediate activations that are a function of low-level features.

To ensure diversity in ensembles, LSTM and GRU networks are trained on different variants of Adam [18] optimization. Adam is best known for realizing low minima of the training loss. Ensembles of networks are built by applying different optimization approaches that boost the performance of the original Adam methods. Because Adam variants are unstable, they can augment diversity among classifiers. An additional improvement of performance is obtained by combining the proposed ensemble of LSTM and GRU with IMCC [7].

This thesis analyzes and discusses the results, based on 10 performance indicators, of the various topologies proposed.

The effectiveness of the proposed ensemble is demonstrated by comparing its performance with several baseline approaches across twelve multilabel benchmarks.

The first section introduces artificial neural networks and illustrates the main types of neural networks and their characteristics.

The second explains the various levels used in the creation of the topologies which are instead described in the third chapter and the Adam optimization approach.

The third chapter illustrates the new topology created which has the best performance: LSTM_GRU.

Finally, in the fourth section, are illustrated the twelve datasets used for the tests and the experimental results of the various proposed ensembles are compared.

1. Artificial neural networks

1.1 Neural network

Artificial Neural Networks are inspired by the principle of information processing in biological systems and are made up of mathematical representations of connected processing units called artificial neurons [34]. Like the synapses in a brain, each connection between neurons transmits signals whose strength can be amplified or attenuated by a weight that is continuously adjusted during the learning process.
For the neural network a neuron can be represented in the following way:

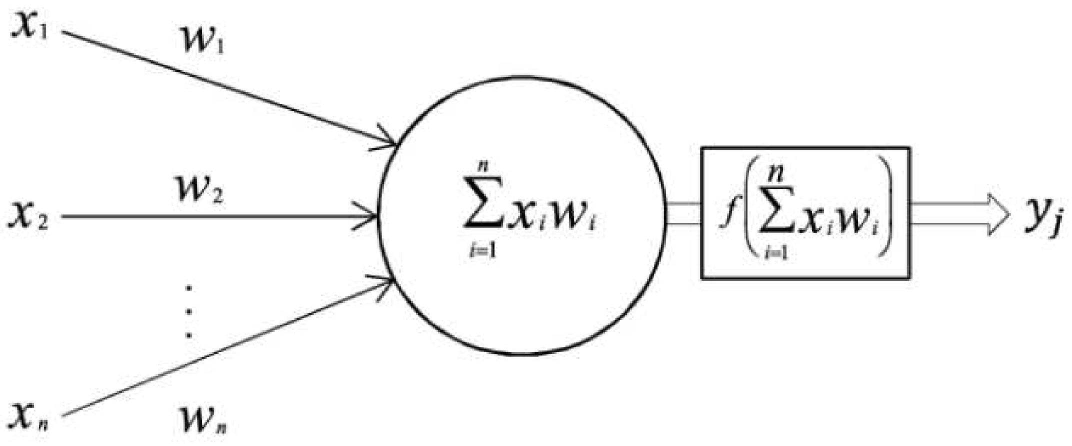


Figure 1.1.1 artificial neuron

Each input x_i has an associated weight w_i. The sum of all weighted inputs, x_i wi, is then passed through a nonlinear activation function f, to transform the preactivation level of the neuron to an output y_j. For simplicity, the bias terms have been omitted. The output y_j then serves as input to a node in the next layer. Several activation functions are available, which differ with respect to how they map a pre-activation level to an output value. The most commonly activation functions used are the rectifier function (where neurons that use it are called rectified linear unit (ReLU)), the hyperbolic tangent function, the sigmoid function and the softmax function. The latter is commonly used in the output layer as it can compute the probability of multiclass labels.

Typically, neurons are organized in networks with several layers. An input layer usually receives the data input and an output layer produces the final result. In between, there are zero or more hidden layers responsible for learning a non-linear mapping between inputs and outputs. The number of layers and neurons, among other property choices, such as the learning rate or the activation function (the function responsible for "signaling" to other neurons), cannot be learned by the learning algorithm. They are the hyperparameters of a model and must be set manually.
The three characteristic elements of each network are:
• the structure of the nodes,
• the network topology,
• the learning algorithm used to find the weights of the network.

1.2 Deep learning and CNN

Deep learning exploits so-called “deep” neural networks (Deep neural networks) composed of at least two hidden levels and inspired by the functioning of the visual system of living organisms. A very common type of "deep" networks are convolutional ones (CNN). They are variants of MLP networks and are mainly used for classification or regression problems. One of the main reasons for their diffusion is that, unlike MLP networks, CNNs exploit two features to reduce their complexity: local processing and shared weights. In the first case the neurons are only locally connected to the neurons of the previous level, while in the second the weights of the connections are shared by groups of neurons. These peculiarities of convolutional neural networks allow to drastically reduce both the number of connections and that of the weights. Convolutional networks are also distinguished by the block construction, these in fact exploit layers of convolution, pooling, activation and completely connected layers. The convolution and pooling layers in particular perform the function of "pre-processing" the data, which is why CNNs use input preprocessing to a minimum. CNNs are very efficient in recognizing images and videos, in recommending systems, in natural language processing and are recently used in the field of bioinformatics.

1.3 Recurrent Neural Network

Recurrent neural networks are a type of artificial neural network designed to recognize patterns in sequences of data, such as numerical times series data emanating from sensors, stock markets and government [35]. What differentiates RNNs from other neural networks is that they take time and sequence into account, they have a temporal dimension. While in feed-forward neural network (e.g., traditional ANNs and DNNs) every node (i.e. neuron) is connected only to nodes of the subsequent layer, in RNNs a node can have connections with others of his layer or even previous ones.

This topology is particularly indicated to generate a sort of memory effect, which allows temporal information of the sequential pattern to be preserved through time. To be more precise, the decision a recurrent net reached at time step *t-1* affects the decision it will reach one moment later at time step t. At time step t, nodes with recurrent edges receive input from the current data point $x_{(t)}$ and also from hidden node values $h_{(t-1)}$ in the network's previous state [36]:

$$h(t) = f\left(h_{(t-1)}, x_{(t)}\right) \tag{1.3.1}$$

Unfortunately, most of RNNs suffer from short-term memory caused by the vanishing gradient problem, which is also prevalent in other neural network architectures like CNNs. As the RNN processes more steps, it has troubles retaining information from previous steps. To mitigate this problem, two specialized recurrent neural networks were created: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). LSTM's and GRU's essentially function just like RNN's, but they're capable of learning long-term dependencies using mechanisms called *gates*. These are different tensor operations that can learn what information to add or remove to the hidden state [37].

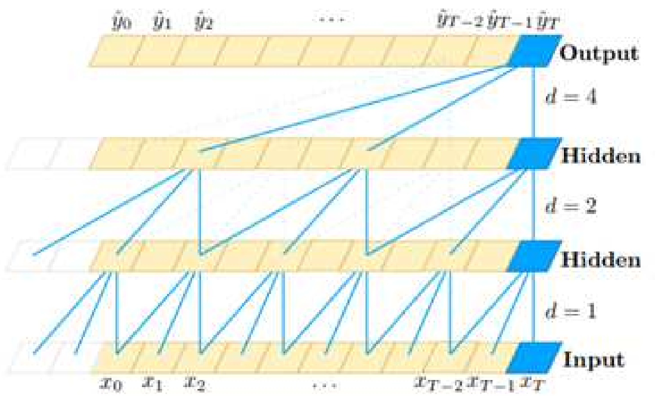
1.4 Temporal Convolutional Neural Network

Temporal Convolutional Neural Networks (TCNs) [26] are a class of neural networks that exploits a hierarchy of convolutions to perform fine-grained detection of events in sequence inputs. Like we've already seen, RNNs represent the most useful networks to deal with sequential samples. However, as has been demonstrated by Bai et al. [38], Convolutional Neural Networks (CNNs) can equal or even outperform recurrent approaches.

The most relevant characteristics of TCNs are the use of 1D convolutions over time that are stacked together to create a deep network. This sequence of convolutions make learning through time possible. Besides, the convolutions of each layer have dilation factor that increases exponentially over the layers, letting the first layers look for short-term connections in the data and deeper layer can spot longer-term dependencies based on the feature extracted by previous layers. This allows TCNs to have a large receptive field, overcoming one of the well-known limitations of most RNN architectures.
Receptive field size can be determined as follows:

$$R = (f-1)(2^K-1)+1 \tag{1.4.1}$$

where *f* is the filter's size and *K* is the number of convolutional layers. The dilation factors of the convolutions are $2^{(k-1)}$ where *k* is the number of the layer.



$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ ... $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 ... x_{T-2} x_{T-1} x_T

Figure 1.4.1. TCN internal structure example, *d* is dilatation factor

1.5 Performance indicators

Multi-label classification is evaluated using several performance indicators so that results can be compared with previous studies. Let X be a data set that includes m samples $\boldsymbol{x}_i \in \Re^d$, each having an actual label $\boldsymbol{y_i} \in \{0,1\}^l$, where l is the number of labels. Letting H and F be the set of predicted labels, where $\boldsymbol{h_i} \in \{0,1\}^l$ is the predicted label vector for sample $\boldsymbol{x}_i$, and $\boldsymbol{f}_i \in \Re^l$ is the confidence relevance of each prediction. The following performance indicators [7] can be defined for H and F:

- Hamming loss is the fraction of misclassified labels,

$$\mathrm{HLoss(H)} = \frac{1}{\mathrm{ml}}\sum_{\mathrm{i=1}}^{\mathrm{m}}\sum_{\mathrm{j=1}}^{\mathrm{l}} \mathrm{I}(\mathbf{y}_{\mathrm{i}}(\mathrm{j}) \neq \mathbf{h}_{\mathrm{i}}(\mathrm{j})) \qquad (1.5.1)$$

where $I()$ is the indicator function. Hamming Loss is a loss function and should be minimized: if the hamming loss is 0, there is no error in the predicted label vector.

- One error is the fraction of instances whose most confident label is wrong. Since it is an error, it should be minimized:

$$\mathrm{OneError(F)} = \frac{1}{m}\sum_{i=1}^{m} I\left(\boldsymbol{h}_i \neq \boldsymbol{y}_i\left(\underset{j}{argmax}\,\boldsymbol{f}_i\right)\right) \qquad (1.5.2)$$

- Ranking Loss is average fraction of reversely ordered label pairs for each instance. It can be obtained from the confidence value considering the number of confidence couples correctly ranked (i.e., a true label is ranked before a wrong label). Ranking loss is a loss function, so it should be minimized.

- Coverage is the average number of steps needed to move down the ranked label list of an instance so as to cover all its relevant labels. Coverage should be minimized.

- Average precision is the average fraction of relevant labels ranked higher than a particular label. It should be maximized.
For the problem of ATC classification, another set of indicators is usually adopted [19]:

- Aiming is the ratio of correctly predicted labels over the practically predicted labels:

$$\mathrm{Aiming(H)} = \frac{1}{m}\sum_{i=1}^{m}\frac{\|\boldsymbol{h}_i \cap \boldsymbol{y}_i\|}{\|\boldsymbol{h}_i\|} \qquad (1.5.3)$$

- Recall, it is the rate of the correctly predicted labels over the actual labels:

$$\mathrm{Recall(H)} = \frac{1}{m}\sum_{i=1}^{m}\frac{\|\boldsymbol{h}_i \cap \mathbf{y}_i\|}{\|\mathbf{y}_i\|} \qquad (1.5.4)$$

- Accuracy, it is the average ratio of correctly predicted labels over the total labels:

$$\text{Accuracy (H)} = \frac{1}{m}\sum_{i=1}^{m}\frac{\|\boldsymbol{h}_i \cap \mathbf{y}_i\|}{\|\boldsymbol{h} \cup \mathbf{y}_i\|} \qquad (1.5.5)$$

- Absolute true, it is the ratio of the perfectly correct prediction events over the total prediction events:

$$\text{AbsTrue(H)} = \frac{1}{m}\sum_{i=1}^{m} I(\boldsymbol{h}_i == \mathbf{y}_i) \qquad (1.5.6)$$

- Absolute false, it is the ratio of the completely wrong predictions over the total prediction events:

$$\text{AbsFalse(H)} = \frac{1}{m}\sum_{i=1}^{m}\frac{\|\boldsymbol{h}_i \cup \mathbf{y}_i\| - \|\boldsymbol{h}_i \cap \mathbf{y}_i\|}{l} \qquad (1.5.7)$$

These last five indicators are inside the range in [0-1] and should be maximized, except for Absolute false.

1.6 Related Works

An early application of deep learning to the problem of multilabel classification was the work by Zhang et al., [20], which applied a simple feed forward network to the functional genomics problem in computational biology. Within the last few years, there has been a growing body of work applying deep learning to a host of multilabel problems in which GRUs have formed the core in several systems. In [21], for example, the authors analyzed sentiment in tweets by taking the topics extracted by a C-GRU (Context-aware Gated Recurrent Units). In [22], a new computational method called NCBRPred was proposed to predict the nucleic acid binding residues based on the multilabel sequence labeling model, which used the bidirectional Gated Recurrent Units (BiGRUs) to capture the global interactions among the residues. In [23], the authors proposed a system composed of an Inception model and GRU to identify nine classes of arrhythmias.

In [7] a powerful data augmentation approach was proposed for multilabel training that incorporated multiple cluster centers, an approach the authors call IMCC.

2. Layers and optimization approach

This section explains the different levels used to create the various topologies and the Adam optimization approach.

2.1 Long Short-Term Memory (LSTM)

An LSTM layer learns long-term dependencies between time steps in time series and sequence data. The layer performs additive interactions, which can help improve gradient flow over long sequences during training.
This diagram illustrates the flow of a time series X with C features (channels) of length S through an LSTM layer. In the diagram 2.1.1, h_t and c_t denote the output (also known as the hidden state) and the cell state at time step t, respectively.

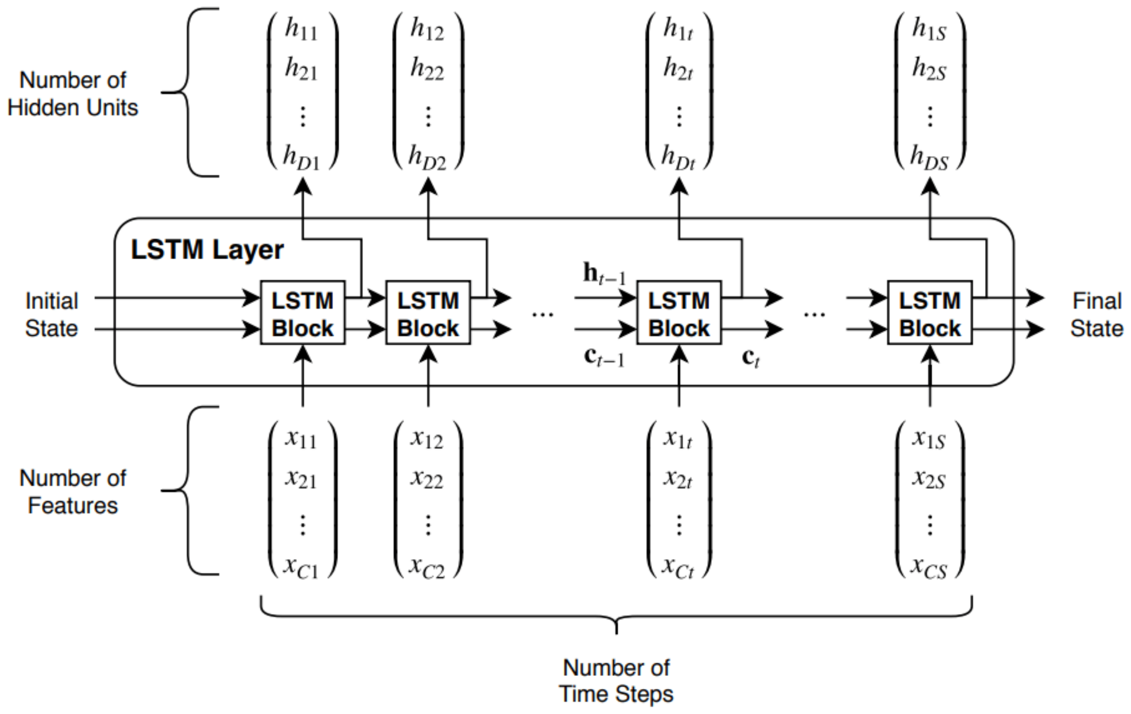

Number of Hidden Units
LSTM Layer
Initial State
LSTM Block
h_{t-1}
c_{t-1}
c_t
Final State
Number of Features
Number of Time Steps
h_{11} h_{21} h_{D1}
h_{12} h_{22} h_{D2}
h_{1t} h_{2t} h_{Dt}
h_{1S} h_{2S} h_{DS}
x_{11} x_{21} x_{C1}
x_{12} x_{22} x_{C2}
x_{1t} x_{2t} x_{Ct}
x_{1S} x_{2S} x_{CS}

Figure 2.1.1 LSTM layer architecture [24]

The first LSTM block uses the initial state of the network and the first time step of the sequence to compute the first output and the updated cell state. At time step t, the block uses the current state of the network (c_{t-1}, h_{t-1}) and the next time step of the sequence to compute the output and the updated cell state ct.

The state of the layer consists of the hidden state (also known as the output state) and the cell state. The hidden state at time step t contains the output of the LSTM layer for this time step. The cell state contains information learned from the previous time steps. At each time step, the layer adds information to or removes information from the cell state. The layer controls these updates using gates.

The basic components of a LSTM are an input gate, forget gate cell candidate and output gate: the first determines level of cell state update, the second determines level of cell state reset (forget), the third adds information to cell state and the fourth controls level of cell state added to hidden state. Let x_t be the input sequence and initialize $h_0 = 0$. Then it's define the input gate i_t , the forget gate f_t , the cell candidate g_t and the output gate o_t as:

$$i_t = \sigma_g(W_i x_t + R_i h_{t-1} + b_i) \tag{2.1.1}$$

$$f_t = \sigma_g\left(W_f x_t + R_f h_{t-1} + b_f\right) \tag{2.1.2}$$

$$g_t = \sigma_c\left(W_g x_t + R_g h_{t-1} + b_g\right) \tag{2.1.3}$$

$$o_t = \sigma_g(W_o x_t + R_o h_{t-1} + b_o) \tag{2.1.4}$$

where $W_i, R_i, b_i, W_f, R_f, b_f, W_g, R_g, b_g, W_o, R_o, b_o$ are matrices and vectors and o_g denotes the gate activation function, σ_c denotes the state activation function. LSTM layer function, by default, uses the sigmoid function given by $\sigma(x) = (1 + e^{-x})^{-1}$ to compute the gate activation function.

Then it's define:

$$\mathrm{c}_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{2.1.5}$$

to be the cell state, where $\odot$ is the Hadamard (component-wise) product.

Then the output vector is:

$$h_t = o_t \odot \sigma_c(c_t\,) \tag{2.1.6}$$

The LSTM layer function, by default, uses the hyperbolic tangent function (tanh) to compute the state activation function.

2.2 Gated Recurrent Unit (GRU)

GRUs are a gating mechanism in recurrent neural networks, created by Cho et al. [17]. They are used to increase the length of term dependencies from the input and handle the gradient vanishing problem. This problem is encountered when training deep neural networks with gradient-based learning methods (backpropagation). The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. Also, this may completely stop the neural network from further training. Non-linear activation functions such as sigmoid function have gradients in the range (0,1], and backpropagation computes gradients by the chain rule. This has the effect of multiplying n of these small numbers to compute gradients of the early layers in an n-layer network, meaning that the gradient decreases exponentially with n while the early layers train very slowly.

GRU can be considered as a simple Long-Short Term Memory (LSTM) variant. Differently from LSTM, GRU has a gate that lets the network decide which part of the old information is relevant to understand the new information [25]. GRU has also fewer parameters and generally has a better performance on smaller data sets.
A GRU layer learns dependencies between time steps in time series and sequence data. The basic components of a GRU layer are a reset gate and an update gate: the first determines how much old information to forget, the second determines what information should forget and what information should pass to the output. Let x_t be the input sequence and initialize $h_0 = 0$. Then it's define the update gate vector z_t and the reset gate vector r_t as:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.2.1}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.2.2}$$

where $W_z, U_z, b_z, W_r, U_r\ b_r$ are matrices and vectors and σ is the sigmoid function. Then it's define:

$$\hat{h}_t = \phi(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{2.2.3}$$

to be the candidate activation vector, where ϕ is the tanh activation and $\odot$ is the Hadamard (component-wise) product. Notice that the term r_t determines the amount of past information that is relevant for the candidate activation vector. Then

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \qquad (2.2.4)$$

is the output vector. The update gate vector z_t measures the amount of old and new information to keep.

2.3 Dropout

A dropout layer randomly sets input elements to zero with a given probability. At each training stage, individual nodes are either dropped out of the net with probability (1-p) or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. It is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

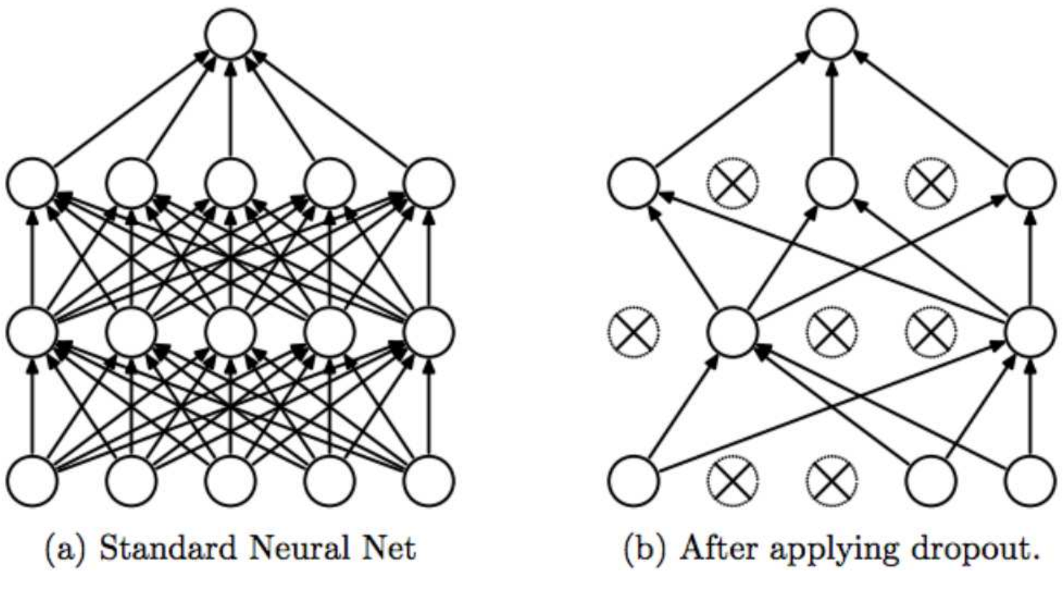


Figure 2.3.1 Effect of dropout layer [27]

Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
It roughly doubles the number of iterations required to converge. However, training time for each epoch is less.

2.4 Convolutional

Convolutional layers are the major building blocks used in convolutional neural networks. A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image. It modifies input features by executing simple mathematical operations with other local features. This may help the model generalize better by consenting features to achieve a higher special independence.

To perform a convolution, it is often necessary to use Stride and Padding operations to optimize the output.

The first is when a filter is slid on the input volume, instead of moving in unit steps (of 1 neuron) a larger step (or Stride) can be used. This operation reduces the size of the feature maps in the output volume and consequently the number of connections.

Padding is a further possibility (to adjust the size of the feature maps) is to add a border (zero values) to the input volume. The Padding parameter denotes the thickness (in pixels) of the border. Padding is useful for filtering the side bits of the image, as without it is not possible to filter these pixels. Without padding, all edge pixels are analyzed by a very small number of filters, as the filters cannot exit the matrix, thus leading to a shrinking (size reduction) of the output and a loss of information.

2.5 Batch-normalization

Batch-normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

2.6 Pooling

Pooling is an important layer that executes the down-sampling on the feature maps coming from the previous layer and produces new feature maps with a condensed resolution. This layer drastically reduces the spatial dimension of input. It serves two main purposes. The first is to reduce the number of parameters or weights, thus lessening the computational cost. The second is to control the overfitting of the network. An ideal pooling method is expected to extract only useful information and discard irrelevant details. There are a lot of methods for the implementation of pooling operation in Neural Networks (e.g. Average Pooling, Max Pooling, GlobalMAx Pooling).

2.7 Fully connected layer

Fully Connected layers in a neural network are those layers where all the inputs from one layer are connected to every activation unit of the next layer. In most popular machine learning models, the last few layers are fully connected layers which compiles the data extracted by previous layers to form the final output. It is the second most time-consuming layer after Convolution Layer.

The fully connected layer is composed by *l* neurons (*l* is the number of output labels of a given problem) fully connected with the previous layer.

2.8 Sigmoid Layer

A sigmoid function is used as activation of this final layer in order to report the activations to the range [0…1] which can be interpreted as the final probabilities (i.e. confidence relevance) of each label. Therefore, the output of the model is a multi-label classification vector: the output of each neuron of the fully connected layer provides a score (ranging from 0 to 1) for a single label.

2.9 Adam optimizer

Adam is an optimizer introduced in [18] which that computes adaptive learning rates for each parameter combining the ideas of momentum and adaptive gradient. Its update rule is based on the value of the gradient at the current step, and on the exponential moving averages of the gradient and its square. To be more precise, Adam defines the moving averages m_t (the first moment) and u_t (the second moment) as:

$$m_t = \rho_1 m_{t-1} + (1 - \rho_1) g_t \tag{2.9.1}$$

$$u_t = \rho_2 u_{t-1} + (1 - \rho_2) g_t^2 \tag{2.9.2}$$

where g_t is the gradient at time *t*, the square on g_t stands for the component-wise square, ρ_1 and ρ_2 are hyperparameters representing the exponential decay rate for the first moment and the second moment estimates (usually set to 0.9 and 0.999, respectively) and the moments are initialized to 0: $m_o = u_0 = 0$. In order to take into account the fact that the value of moving averages will be very small due to their initialization to zero (especially in the first steps), the authors of Adam define a bias-corrected version of the moving averages:

$$\widehat{m}_t = \frac{m_t}{(1 - \rho_1^t)} \tag{2.9.3}$$

$$\hat{u}_t = \frac{u_t}{(1 - \rho_2^t)} \tag{2.9.4}$$

The final update for each θ_t parameter of the network is:

$$\theta_t = \theta_{t-1} - \lambda \frac{\widehat{m}_t}{\sqrt{\hat{u}_t} + \epsilon} \tag{2.9.5}$$

where λ is the learning rate, ϵ is a very small positive number to prevent any division by zero (usually set to 10^{-8}) and all the operations are meant to be component-wise. Notice that, while g_t might have positive or negative components, g_t^2 has only positive components. Hence, if the gradient changes sign often, the value of $\widehat{m}_t$ might be much lower than $\sqrt{\hat{u}_t}$. This means that in this case the step size is very small.

3. Model architecture of LSTM_GRU

The focus of this study is on the topology of neural networks to improve the performances. A lot of different structures created with the combination of the layers presented in the section above are tested in “Image” and “mAn” dataset. Then given the computational times only for LSTM_GRU, exhaustive tests have been launched.

Training is performed using the different variants of Adam optimizer. It is used a high learning rate of 0.01 and specify gradient decay and squared gradient decay factors of 0.5 and 0.999, respectively. Moreover, the gradients is clipped with a threshold of 1 using L2 norm gradient clipping. The minibatch size has been fixed to 30, while the number of epochs is set to 150.

The base schema of the model is provided in Figure 3.1. The network is composed by a first LSTM layer with N hidden units (set to 125), followed by a dropout layer that randomly sets input elements to zero with probability of 0.4. Then there is a GRU layer with N hidden units (set to 100) and then another dropout layer with probability of 0.4. At the end there is a fully connected layer that is followed by a sigmoid Layer.

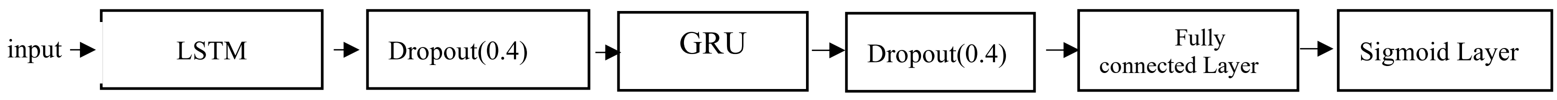


Figure 3.1. Schema of LSTM_GRU

The loss function is the binary cross entropy loss between the outputs (predicted labels) and the target or actual labels. The binary cross entropy loss calculates the loss of a set of m observations by computing the following average:

$$\text{CELoss} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{l}\mathbf{y}_i(j)\ \cdot\log\big(\mathbf{h}_i(j)\big) + (1-\mathbf{y}_i(j))\ \cdot\log\big(1-\mathbf{h}_i(j)\big)\,,$$

where $\boldsymbol{y_i} \in \{0,1\}^l$ and $\boldsymbol{h_i} \in \{0,1\}^l$ are the actual and predicted label vectors of each sample ($i \in 1 \dots m$), respectively.

In the figures 3.2 and 3.3 we can see the trend of the loss function as the number of iterations increases over time, the first one is for “Image” dataset the second one for “mAn”.

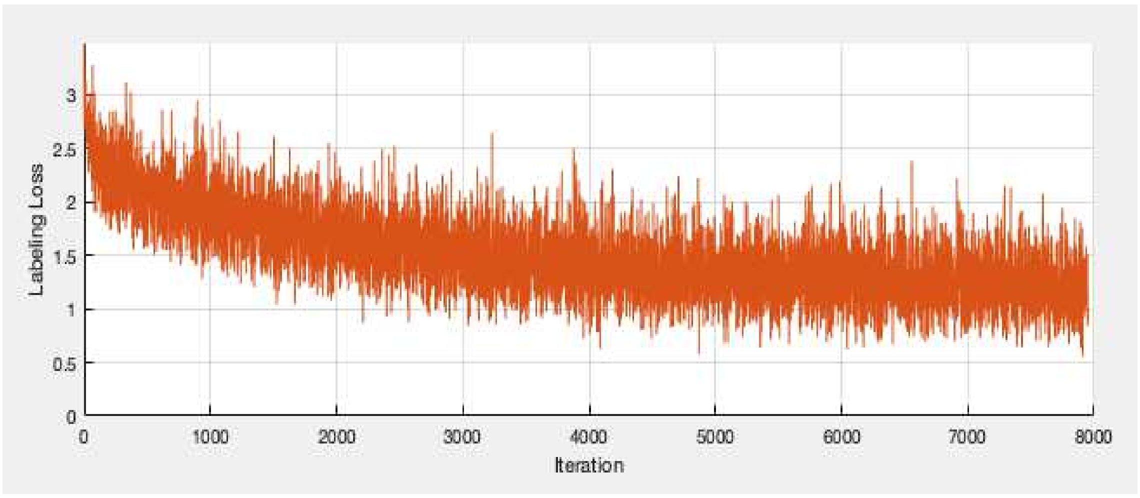


Figure 3.2 Loss function for “mAn” dataset

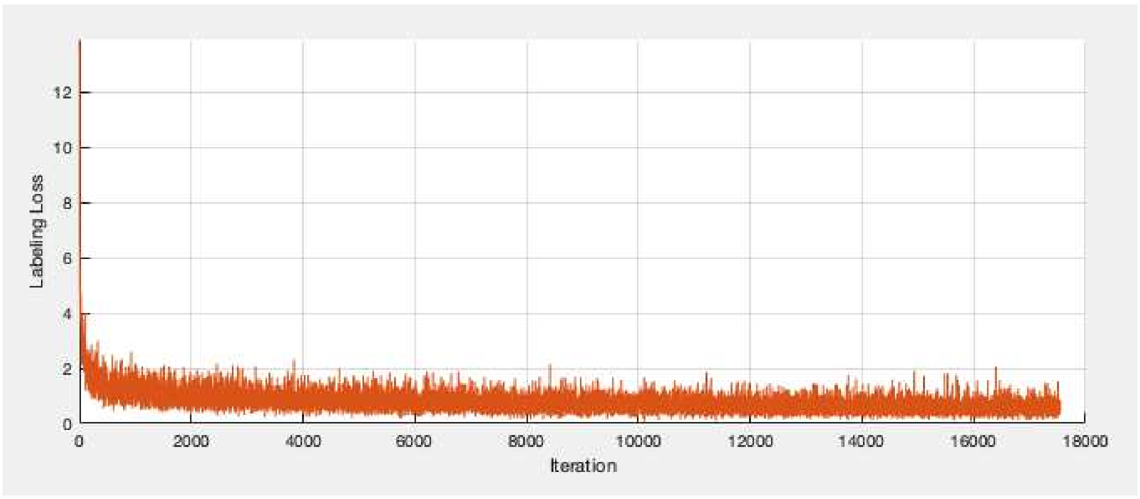


Figure 3.3 Loss function for “Image” dataset

4. Experimental results

In this section created ensembles and experimental result are presented.

4.1 Datasets

Evaluation of the proposed approach requires data sets with binary multilabel categorization. The following twelve data sets were selected because they represent a wide range of applications (such as image, music, biology, and drug classification) and because they are often used when comparing multilabel classification systems (note: the names of the data sets are those typically used in the literature and not necessarily those in the original papers):

1. Cal500 [28]: this is a collection of human-generated annotations that describe popular Western music tracks produced by 500 unique artists. Cal500 includes 502 instances represented by 68 numeric features and 174 distinct labels.
2. Scene [29]: this is a collection of 2407 color images (divided into training and testing images) of different scenes grouped into six base categories *beach* (369), *sunset* (364), *fall foliage* (360), *field* (327), *mountain* (223), and *urban* (210), with sixty-three images belonging to two categories and only one to three categories. Taking into account the joint categories, the total number of labels is fifteen. Each image in this data set went through a four-step preprocessing procedure. In step 1, an image was converted to the CIE Luv space since this space is perceptually uniform (i.e., close to Euclidean distances). In step 2, the image was divided into 49 blocks with a 7×7 grid. In step 3, the mean and variance of each band were computed. The mean is equivalent to a low-resolution image, whereas the variance corresponds to computationally inexpensive texture features. Finally, in step 4, the image was transformed into a 294-dimensional feature vector (49×3×2).
3. Image [30]: this is a collection of 2,000 natural scene images grouped into five base categories *desert* (340), *mountains* (268), *sea* (341), *sunset* (216), and *trees* (378) that intended to produce a much larger set of images than scene that belong to two categories (442) and three categories (15). Taking into account the joint

categories, the total number of labels is 20. The images in this data set went through the same preprocessing procedure as in [29].

4. Yeast [2]: this is a biologic data set for classifying 2417 micro-array expression data and phylogenetic profiles (represented by 103 features) into 14 functional classes. A gene can belong to more than one class.
5. Arts [7]: this data set was built using 5000 art images, each described by 462 numeric features where each image can belong to some 26 classes.
6. Liu [29]: this is a data set of drugs collected to predict drug side effects. It includes 832 compounds represented by 2892 features and 1385 labels.
7. ATC [31]: this is a collection of 3883 ATC coded pharmaceuticals with each sample represented by 42 features and 14 classes.
8. ATC_f: this is a variant of the ATC collection where the same patterns are represented by an 806-dimensional descriptor (i.e., all the three descriptors are tested as in [3]).
9. mAn [6]: this is a data set of proteins represented by 20 features and 20 labels.
10. Bibtex, Enron, Health: these are three highly sparse datasets used in [7]

A summary of all the data sets, including the number of patterns, features, labels, and the average number of class labels per pattern (LCard), is reported in Table 4.1.1. A 5-fold cross-validation testing protocol with results averaged is used for data set 6, and a 10-fold protocol for data sets 7-9. Data sets 1-5 are in the MATLAB IMCC toolkit [7] available at https://github.com/keauneuh/Incorporating-Multiple-Cluster-Centers-for-Multi-Label-Learning/tree/master/IMCCdata (accessed 9/9/21). All other data sets can be obtained from the authors in the references provided above.

Name	#patterns	#features	#labels	LCard
CAL500	502	68	174	26.044
Image	2000	294	5	1.236
Scene	2407	294	5	1.074
Yeast	2417	103	14	4.24
Arts	5000	462	26	1.636
ATC	3883	42	14	1.265
ATC_f	3883	700	14	1.265
Liu	832	2892	1385	71.160
mAn	3916	20	20	1.650
bibtex	7395	1836	159	2.402
enron	1702	1001	53	3.378
health	5000	612	32	1.662

Table *4.1.1.* Summary of the twelve data sets tested in this work.

4.2 Ensemble description

Ensembles combine the output of multiple models to improve the system performance and to counter overfitting. A feasible method to improve predictions and generalization of the ensemble is to increase the diversity of the classifiers. Our ensemble architecture is based on the fusion with the average rule of several models trained on the same problem.

Optimizers play a fundamental rule in finding a minimum of the loss function: different optimization strategies can converge to different local minima and thus they can achieve different optima. Several optimizers suitable for ensemble creation are evaluated: Adam optimizer [18], diffGrad [33] and eleven novel variants of this approach (DGrad, Cos#1, Exp, Sto, CLRW, EpochCos#1, EpochDecay, Sqrt, Hyp, CyclicExp and LRClipping).

Experiments' role is to test the performance of new topologies.

Incorporating Multiple Clustering Center (IMCC) represents one of the most advanced strategies for Multi-label classification and can be used to compare our results to the state-of-the-art.

IMMC [7] is a two-step process: 1) the generation of virtual examples for augmenting the training set and 2) multilabel training. As step 1, augmentation, is what gives IMCC its main performance advantage. Augmentation is performed via clustering using *k*-means [32] and the calculation of clustering centers.

Ensembles architecture is based on the fusion with the average rule of several models trained on the same problem. Here's a description of ensembles created and trained:

- StoGRU is an ensemble composed of 40 GRU_A (GRU with *N* hidden units (the number *N* of hidden units is set to 50 in our experiments), followed by a max pooling layer, a fully connected layer and sigmoid), combined by average rule;
- StoGRU_B as StoGRU but based on GRU_B (as GRU_A but with a convolutional level applied immediately before the network itself and followed by batch-normalization layer);
- StoTCN is an ensemble of 40 TCN_A (TCN with *N* hidden units (the number *N* of hidden units is set to 50 in our experiments), followed by a fully connected layer, a max pooling layer and sigmoid), combined by average rule;
- StoTCN_B as StoTCN but based on TCN_B (as TCN_A but with a convolutional level applied immediately before the network itself);
- StoGRU_TCN is an ensemble of 40 GRU_TCN (sequential combination of GRU_A, without the pooling layer, and TCN_A) each coupled with the stocastic approach;
- LSTM_GRU is an ensemble composed of 40 LSTM_GRU (the new topology presented in section 3)
- ENNbase is the fusion by average rule of StoGRU and StoTCN.
- ENN is the fusion by average rule of StoGRU, StoTCN, StoGRU_B, StoTCN_B and StoGRU_TCN;
- ENNnew: is the fusion by average rule of ENN and LSTM_GRU;

- LeaveOO_Sparse: extract one dataset from the total of n and use the others (n-1) to select methods that classify the given dataset. In this case I consider only the scattered datasets;
- LeaveOO_NonSparse: extract one dataset from the total of n and use the others (n-1) to select methods that classify the given dataset, considering only non-scattered datasets;
- LeaveOO: extract one dataset from the total of n and use the others (n-1) to select methods that classify the given dataset, considering all datasets;

I also report different fusions between ENNnew/ENN and IMCC:

- ENN+IMCC is the sum rule between ENN and IMCC; before fusion the scores of ENN were normalized since it has a different range of values compared to IMCC. Normalization was performed as ENN=(ENN-0.5)×2 with the classification threshold set to zero;
- ENN+3×IMCC is the same as the previous fusion but the scores of IMCC are weighted by a factor of three.
- ENNnew+IMCC is the sum rule between ENNnew and IMCC; before fusion the scores of ENN were normalized since it has a different range of values compared to IMCC.
- ENNnew+3×IMCC is the same as the previous fusion but the scores of IMCC are weighted by a factor of three.

4.3 Results

Experimental results obtained by previously described ensembles are reported in the following tables:

Average Precision	Cal500	image	scene	yeast	arts	ATC	ATC_f	Liu	mAn	bibtex	enron	health	Average
IMCC	**0.502**	0.836	0.904	0.773	0.619	0.866	0.922	0.523	0.978	0.623	0.714	0.781	0.753
StoGRU	0.498	0.851	0.911	0.740	0.561	0.872	0.872	0.485	**0.979**	0.403	0.680	0.739	0.715
StoGRU_B	0.490	**0.861**	0.908	0.741	0.555	0.877	0.848	0.485	0.978	0.400	0.688	0.724	0.712
StoTCN	0.498	0.847	0.913	0.764	0.506	0.882	0.900	0.498	0.977	0.406	0.669	0.710	0.714
StoTCN_B	0.497	0.855	0.917	0.765	0.541	0.883	0.903	0.505	0.976	0.404	0.666	0.732	0.720
StoGRU_TCN	0.491	0.852	0.916	0.752	0.592	0.890	0.913	0.510	0.977	0.354	0.674	0.764	0.724
LSTM_GRU	0.493	0.837	0.888	0.768	**0.637**	0.873	0.742	**0.551**	0.976	0.596	0.714	0.784	0.738
ENNbase	**0.502**	0.855	0.922	0.756	0.552	0.888	0.916	0.497	**0.979**	0.417	0.687	0.735	0.726
ENN	0.499	0.859	**0.924**	0.762	0.582	**0.893**	0.916	0.505	**0.979**	0.424	0.689	0.749	0.732
ENNnew	0.498	0.866	0.922	0.778	0.628	0.892	0.917	0.521	0.978	0.521	0.707	0.782	0.751
LeaveOO_Sparse	---	---	---	---	0.628	---	---	0.530	---	**0.628**	**0.724**	**0.792**	----
LeaveOO_NonSparse	0.501	0.854	0.918	0.781	---	0.882	0.925	---	**0.979**	---	---	---	---
LeaveOO	0.500	0.847	0.918	0.781	0.635	0.882	0.923	0.530	**0.979**	**0.628**	0.720	0.789	0.761
ENN+IMCC	0.502	0.856	0.922	**0.783**	0.631	0.883	**0.927**	0.521	**0.979**	0.622	0.714	0.783	0.760
ENN+3×IMCC	**0.503**	0.845	0.917	0.777	0.627	0.876	0.926	0.524	**0.979**	0.624	0.716	0.784	0.758
ENNnew+IMCC	0.502	0.855	0.922	0.782	0.634	0.883	**0.927**	0.528	**0.979**	0.626	0.718	0.789	**0.762**
ENNnew+3×IMCC	**0.503**	0.847	0.913	0.778	0.629	0.875	0.925	0.527	**0.979**	0.626	0.718	0.787	0.759

Table 4.3.1 Comparisons between the ensembles using average precision

Each number in these tables represents the average precision for the ensemble tested (reported in row) on a certain dataset (reported in column). I made bold the methods which reach the best performances for each dataset.

As we can see, tests are executed for twelve different datasets. For the highly sparse data sets (Liu, Arts, bibtex, enron and health), I perform a dimensionality reduction by PCA by retaining 99% of variance. LSMT_GRU does not converge if a normalization step is performed for the ATC_f dataset; moreover, it works very poorly also if normalization step is not performed. Instead the performance increase if the PCA projection is applied to that dataset, anyway, its performance is still lower than that obtained by other methods.

It used to avoid overfitting when the number of features is much larger than the number of observations. Overfitting is a particular situation in which, because of patterns' high dimensionality, the network needs to tune lots of parameters and gets too many degrees of freedom. This can bring the network to be extremely accurate in training (by memorizing all the samples) but very inaccurate in testing (because of its inability to generalize well dealing with real world samples).

The performance reported in the previous table 4.3.1 permits to draw the following conclusions:

- LSTM_GRU works very well in sparse datasets, in non-sparse datasets the obtained performance is similar to that obtained by the other methods based on GRU/TCN;
- LeaveOO_Sparse permits to boost performance of ENN+3×IMCC, instead LeaveOO_NonSparse and LeaveOO seem to not reach interesting performance.
- ENNnew+IMCC is our suggested ensemble, it always outperforms IMCC and it is more robust than ENN+IMCC to sparse data (*Arts*, *Liu, bibtex, enron and health*).

In general, the new topology LSTM_GRU tends to equalize and, in some case, to improve the performances of previous ensembles. But by fusing it with ENN and then with the sum rule with IMCC, results overperform the state-of-the-art.

Conclusions

Multi-label learning addresses the problem that each instance is associated with multiple labels at the same time.

In this paper it has been proposed to combine ensemble of the new topology LSTM_GRU with other models, trained with variants of Adam optimization approach. This approach is also combined with Incorporating Multiple Clustering Centers (IMCC) for superior multilabel classification.

To validate this approach a set of twelve data sets have been used, the reported results show that the proposed ensemble overperforms state of the art performance.

The merger with other deep learning topologies for feature extraction and new optimizations could be an interesting starting point for future experiments.

References

[1] E. G. Galindo and S. Ventura, "Multi label learning: a review of the state of the art and ongoing research," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 4, no. 6, pp. 411-444, 2014, doi: doi.org/10.1002/widm.1139.

[2] A. Elisseeff and J. Weston, *A kernel method for multi-labelled classification* (NIPS). MIT Press Direct, 2001.

[3] L. Nanni, A. Lumini, and S. Brahnam, "Neural networks for Anatomical Therapeutic Chemical (ATC)," *ArXiv,* vol. abs/2101.11713, 2021.

[4] X. Cheng, W.-Z. Lin, X. Xiao, and K.-C. Chou, "pLoc_bal-mAnimal: predict subcellular localization of animal proteins by balancing training dataset and PseAAC," *Bioinformatics,* vol. 35, no. 3, pp. 398-406, 2018, doi: 10.1093/bioinformatics/bty628.

[5] L. Chen *et al.*, "Predicting gene phenotype by multi-label multi-class model based on essential functional features," *Molecular genetics and genomics : MGG,* vol. 296, no. 4, pp. 905-918, 2021, doi: 10.1007/s00438-021-01789-8.

[6] Y. Shao and K. Chou, "pLoc_Deep-mAnimal: A Novel Deep CNN-BLSTM Network to Predict Subcellular Localization of Animal Proteins," *Natural Science,* vol. 12, 5, pp. 281-291, 2020, doi: 10.4236/ns.2020.125024.

[7] S. Shu *et al.*, "Incorporating Multiple Cluster Centers for Multi-Label Learning," *ArXiv,* vol. abs/2004.08113, 2020.

[8] M. Ibrahim, M. U. G. Khan, F. Mehmood, M. Asim, and W. Mahmood, "GHS-NET a generic hybridized shallow neural network for multi-label biomedical text classification," *Journal of biomedical informatics,* vol. 116, p. 103699, 2021, doi: 10.1016/j.jbi.2021.103699.

[9] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light Gated Recurrent Units for Speech Recognition," *IEEE Transactions on Emerging Topics in Computational Intelligence,* vol. 2, no. 2, pp. 92-102, 2018, doi: 10.1109/TETCI.2017.2762739.

[10] Y. Kim and J. Kim, "Human-Like Emotion Recognition: Multi-Label Learning from Noisy Labeled Audio-Visual Expressive Speech," presented at the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018.

[11] M. B. Messaoud, I. Jenhani, N. B. Jemaa, and M. W. Mkaouer, "A Multi-label Active Learning Approach for Mobile App User Review Classification," in *KSEM*, 2019.

[12] J. P. Singh and K. Nongmeikapam, "Negative Comments Multi-Label Classification," *2020 International Conference on Computational Performance Evaluation (ComPE),* pp. 379-385, 2020, doi: 10.1109/ComPE49325.2020.9200131.

[13] W. Zhang, F. Liu, L. Luo, and J. Zhang, "Predicting drug side effects by multi-label learning and ensemble learning," *BMC Bioinformatics,* vol. 16, no. 365, 2015.

[14] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Random k-Labelsets for Multilabel Classification," *IEEE Transactions on Knowledge and Data Engineering,* vol. 23, no. 7, pp. 1079-1089, 2011.

[15] Y. Yang and J. Jiang, "Adaptive Bi-Weighting Toward Automatic Initialization and Model Selection for HMM-Based Hybrid Meta-Clustering Ensembles," *IEEE Transactions on Cybernetics,* vol. 49, no. 5, pp. 1657-1668, 2019.

[16] J. M. Moyano, E. G. Galindo, K. Cios, and S. Ventura, "Review of ensembles of multi-label classifiers: Models, experimental study and prospects," *Inf. Fusion,* vol. 44, no. November, pp. 33-45, 2018.

[17] K. Cho *et al.*, "Learning Phrase Representations using RNN Encoder Decoder for Statistical Machine Translation," in *EMNLP*, Varna, Bulgaria, 2014, pp. 25-32.

[18] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR,* vol. abs/1412.6980, 2015.

[19] K. C. Chou, "Some remarks on predicting multi-label attributes in molecular biosystems," *Molecular Biosystems,* vol. 9, pp. 10922-1100, 2013.

[20] M.-L. Zhang and Z.-H. Zhou, "Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization," *IEEE Transactions on Knowledge and Data Engineering,* vol. 18, no. 10, pp. 1338-1351, 2006, doi: 10.1109/TKDE.2006.162.

[21] A. E. Samy, S. R. El-Beltagy, and E. Hassanien, "A Context Integrated Model for Multi-label Emotion Detection," *Procedia Computer Science,* vol. 142, pp. 61-71, 2018/01/01/ 2018, doi: https://doi.org/10.1016/j.procs.2018.10.461.

[22] J. Zhang, Q. Chen, and B. Liu, "NCBRPred: predicting nucleic acid binding residues in proteins based on multilabel learning," *Briefings in bioinformatics,* vol. 22, no. 2, 2021.

[23] D. Li, H. Wu, J. Zhao, Y. Tao, and J. Fu, "Automatic Classification System of Arrhythmias Using 12-Lead ECGs with a Deep Neural Network Based on an Attention Mechanism," *Symmetry,* vol. 12, no. 11, p. 1827, 2020. [Online]. Available: https://www.mdpi.com/2073-8994/12/11/1827.

[24] Hochreiter, S., and J. Schmidhuber. "Long short-term memory." Neural computation. Vol. 9, Number 8, 1997, pp.1735–1780.

[25] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation,* vol. 12, no. 10, pp. 2451-2471, 2000, doi: 10.1162/089976600300015015.

[26] K. Zhang, Z. Liu, and L. Zheng, "Short-Term Prediction of Passenger Demand in Multi-Zone Level: Temporal Convolutional Neural Network With Multi-Task Learning," *IEEE Transactions on Intelligent Transportation Systems,* vol. 21, no. 4, pp. 1480-1490, 2020, doi: 10.1109/TITS.2019.2909571.

[27] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

[28] D. Turnbull, L. Barrington, D. A. Torres, and G. Lanckriet, "Semantic Annotation and Retrieval of Music and Sound Effects," *IEEE Transactions on Audio, Speech, and Language Processing,* vol. 16, no. 22, pp. 467-476, 2008, doi: 10.1109/TASL.2007.913750.

[29] M. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern Recognit.,* vol. 37, no. 9, pp. 1757-1771, 2004.

[30] M.-L. Zhang and Z. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," *Pattern Recognit.,* vol. 40, no. 7, pp. 2038-2048, 2007, doi: 10.1016/j.patcog.2006.12.019.

[31] L. Chen, "Predicting anatomical therapeutic chemical (ATC) classification of drugs by integrating chemical-chemical interactions and similarities," *PLoS ONE,* vol. 7, no. e35254, 2012.

[32] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comp Surv,* vol. 31, no. 3, pp. 264-323, 1999.

[33] S. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh, and B. Chaudhuri, "diffGrad: An Optimization Method for Convolutional Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 31, no. 11, pp. 4500-4511, 2020.

[34] Janiesch, C., Zschech, P. & Heinrich, K., «Machine learning and deep learning. Electron Markets,» 2021. [Online]. Available: https://doi.org/10.1007/s12525-021-00475-2.

[35] C. Nicholson, «A.I. Wiki- A Beginner's Guide to Important Topics in AI, Machine Learning, and Deep Learning.,» [Online]. Available: https://wiki.pathmind.com/lstm.

[36] Z. C. Lipton, J. Berkowitz, C. Elkan, «A Critical Review of Recurrent Neural Networks,» arXiv:1506.00019, 2015.

[37] M. Phi, «Illustrated Guide to Recurrent Neural Networks,» Towards Data Science, [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9.

[38] J. Z. K. V. K. Shaojie Bai, «An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling,» arXiv:1803.01271, 2018.