The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

A Deep Model with Local Surrogate Loss for General Cost-Sensitive Multi-label Learning

Cheng-Yu Hsieh, Yi-An Lin, Hsuan-Tien Lin
Department of Computer Science and Information Engineering
National Taiwan University
{r05922048, r02922163}@ntu.edu.tw htlin@csie.ntu.edu.tw

Abstract

Multi-label learning is an important machine learning problem with a wide range of applications. The variety of criteria for satisfying different application needs calls for cost-sensitive algorithms, which can adapt to different criteria easily. Nevertheless, because of the sophisticated nature of the criteria for multi-label learning, cost-sensitive algorithms for general criteria are hard to design, and current cost-sensitive algorithms can at most deal with some special types of criteria. In this work, we propose a novel cost-sensitive multi-label learning model for any general criteria. Our key idea within the model is to iteratively estimate a surrogate loss that approximates the sophisticated criterion of interest near some local neighborhood, and use the estimate to decide a descent direction for optimization. The key idea is then coupled with deep learning to form our proposed model. Experimental results validate that our proposed model is superior to existing cost-sensitive algorithms and existing deep learning models across different criteria.

1 Introduction

Multi-label learning (MLL) addresses the problem of associating each data point with a set of relevant labels. It has recently attracted much research attention since the problem setting meets the needs of various real-world applications. For instance, in image classification, an image may contain multiple objects simultaneously (Boutell et al. 2004). Other MLL applications include text categorization (Schapire and Singer 2000), music tag annotation (Lo et al. 2011), and video classification (Qi et al. 2007). Different MLL applications often aim for different goals, and thus a variety of criteria have been proposed to measure the performance of MLL algorithms from different angles. Some popular criteria include Hamming loss, Rank loss, Example-F1, Micro-F1, Macro-F1, and Precision-at-k (Tsoumakas, Katakis, and Vlahavas 2010; Madjarov et al. 2012).

Classical MLL algorithms such as binary relevance (Tsoumakas, Katakis, and Vlahavas 2010), classifier chain (Read et al. 2011), and label powerset (Tsoumakas, Katakis, and Vlahavas 2010) are designed to optimize some specific criterion. Nevertheless, because of the different behaviors of different criteria, an algorithm that optimizes one criterion well may not be a good choice for other criteria, and it is difficult to modify those classical algorithms towards other criteria. The demands from real-world applications call for algorithms that can adapt to optimize different evaluation criteria. Such algorithms allows applications to not only conduct goal-specific optimization but also change their goals more easily if needed. As the evaluation criterion defines the *cost* for misclassifications made by the learning algorithms, MLL algorithms that adapt to optimize different criteria is generally referred to as cost-sensitive multi-label learning (CSMLL) algorithms (Li and Lin 2014).

Many CSMLL algorithms have been proposed in recent years (Dembczynski, Cheng, and Hüllermeier 2010; Lo et al. 2011; Li and Lin 2014; Wu and Lin 2017; Huang and Lin 2017). For example, probabilistic classifier chain (Dembczynski, Cheng, and Hüllermeier 2010) makes cost-sensitive prediction with inference steps towards Bayes-optimal decisions, often with the help of an efficient inference rule that corresponds to the criterion of interest; condensed filter tree (Li and Lin 2014) adapts to optimize different criteria by transforming the criterion into sample weights when training the underlying classifiers; progressive random k-labelsets (Wu and Lin 2017) reduces the original CSMLL problem into multiple cost-sensitive multi-class classification subproblems. Nonetheless, current CSMLL algorithms are generally restricted to a certain class of criteria that can be decomposed to per-instance measures, and cannot deal with other criteria such as Precision-at-k.

In this work, we loosen the restriction and study a more general CSMLL setting that requires the algorithms to adapt to optimize virtually any common MLL criterion. However, given the complicated nature of MLL criteria, designing such *general* cost-sensitive algorithm is challenging. In particular, most of the criteria are highly non-convex and even discontinuous, and it is thus generally impossible to optimize the criterion directly through numerical optimization. A common practice to tackle the difficulty is to develop appropriate surrogate loss function of the criterion of interest to make the optimization procedure tractable. (Petterson and Caetano 2010; 2011; Zhang and Zhou 2006; Gong et al. 2013; Nam et al. 2014; Gao and Zhou 2011; Dembczynski, Kotlowski, and Hüllermeier 2012). The surrogate loss function serves as a smooth proxy of the criterion and carries better optimization properties during training.

Nevertheless, current surrogate loss functions rely on human designs with respect to one or a few criteria, and cannot be systematically applied to solve the *general* CSMLL problem for any criteria of interest.

In this work, we approach the general CSMLL problem by letting the machine *learn* a surrogate loss function for the criterion of interest, therefore escaping from the restrictions of human designs. Nevertheless, given the complicated nature of the MLL criteria, learning a global surrogate loss function turns out to be computationally demanding and conceptually difficult. We thus propose to learn the surrogate loss function locally. That is, we plug the surrogate-learning step into the iterative numerical optimization procedure of training CSMLL classifiers. In each surrogate-learning step, the proposed locally-learned surrogate loss (LLSL) is only required to approximate a given criterion of interest near some *local neighborhood*. The approximation captures the local behavior of the criterion's cost surface, and carries sufficient information to guide the numerical optimization procedure towards a descent direction for optimizing the criterion. We further combine the idea of LLSL with gradient-based optimization in deep learning to propose a novel deep model for multi-label learning that can automatically adapt to optimize general criteria.

The main contributions of this paper are highlighted as follows:

- We present a novel methodology that systematically and automatically learns to optimize any given criterion for multi-label learning. The methodology solves a broader range of CSMLL problems than existing CSMLL algorithms.
- The methodology is used to design the world's first cost-sensitive deep learning model for multi-label learning.
- The proposed deep learning model enjoys superior performances against existing methods across various real-world datasets and evaluation criteria.

2 Background

2.1 Problem Setup

In a multi-label learning (MLL) problem, we denote an instance by a feature vector $\mathbf{x} \in \mathbb{R}^d$, and its relevant labels by a bit vector $\mathbf{y} \in \{0,1\}^K$, where K is the number of labels and $\mathbf{y}[k] = 1$ if and only if the k-th label is relevant. Given a training set $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$, the goal of the multi-label learning problem is to learn a hypothesis $h\colon \mathbb{R}^d \to \mathbb{R}^K$ to make predictions on unseen instances accurately. The flexible definition of h above covers two typical cases: a *classifier* h which is only allowed to output bit vectors that directly decides the relevance of each label, or a *ranker* h whose real-valued outputs can be used to rank the labels by the predicted relevance level.

More specifically, given a test dataset $\mathcal{D}' = \{\mathbf{x}'_m\}_{m=1}^M$, the goal is to make the prediction vectors $\{\hat{\mathbf{y}}'_m = h(\mathbf{x}'_m)\}_{m=1}^M$ close to the hidden ground-truth vectors $\{\mathbf{y}'_m\}_{m=1}^M$. Denote a matrix $\mathbf{Y}'$ that contains $\{\mathbf{y}'_m\}_{m=1}^M$ as its rows, and another matrix $\hat{\mathbf{Y}}'$ that contains $\{\hat{\mathbf{y}}'_m\}_{m=1}^M$ as its rows, the goal can be expressed as minimizing a *criterion* $\Psi(\mathbf{Y}', \hat{\mathbf{Y}}')$ that measures the difference between the two matrices of vectors. A special family of criteria, called example-based criteria, measures the average difference vector by vector (row by row). That is, $\Psi(\mathbf{Y}', \hat{\mathbf{Y}}') = \frac{1}{M}\sum_{m=1}^M \psi(\mathbf{y}'_m, \hat{\mathbf{y}}'_m)$. For instance, when h is a classifier, one simplest choice is $\psi_H(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{K}\sum_{k=1}^K [\![\mathbf{y}[k] \neq \hat{\mathbf{y}}[k]]\!]$, called the Hamming loss. Other popular choices include the Example-F1 loss, where $\psi_F(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{2\mathbf{y}\bullet\hat{\mathbf{y}}}{\|\mathbf{y}\|_1 + \|\hat{\mathbf{y}}\|_1}$; and the Rank loss with $\psi_R(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{R(\mathbf{y})}\sum_{(k,l):\mathbf{y}[k]<\mathbf{y}[l]} [\![\hat{\mathbf{y}}[k] > \hat{\mathbf{y}}[l]]\!] + \frac{1}{2}[\![\hat{\mathbf{y}}[k] = \hat{\mathbf{y}}[l]]\!]$, where $R(\mathbf{y}) = |\{(k,l)|\mathbf{y}[k] < \mathbf{y}[l]\}|$ is a normalizer. When h is a ranker, a common example-based criterion is called (negative) Precision-at-k, where $\psi_P(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{1}{k}\sum_{\ell \in \text{top}_k(\hat{\mathbf{y}})} \mathbf{y}[\ell]$ and $\text{top}_k(\hat{\mathbf{y}})$ returns the indices that correspond to the k largest values in $\hat{\mathbf{y}}$. As minimizing these instance-averaging criteria can be reduced to minimizing ψ on each instance, ψ is used in this paper to replace Ψ for example-based criteria.

In addition, the definition of Ψ covers the more general case of measuring the difference *for the entire test set (prediction matrix)*. For example, when h is a classifier, the popular Macro-F1 loss can be expressed as

$$\Psi_{ma}(\mathbf{Y}, \hat{\mathbf{Y}}) = 1 - \frac{1}{K}\sum_{k=1}^K \frac{2\sum_{m=1}^M \mathbf{Y}_{mk}\hat{\mathbf{Y}}_{mk}}{\sum_{m=1}^M \mathbf{Y}_{mk} + \sum_{m=1}^M \hat{\mathbf{Y}}_{mk}},$$

which is physically the mean F1 loss per label. The Micro-F1 loss can be similarly expressed as

$$\Psi_{mi}(\mathbf{Y}, \hat{\mathbf{Y}}) = 1 - \frac{2\sum_{k=1}^K\sum_{m=1}^M \mathbf{Y}_{mk}\hat{\mathbf{Y}}_{mk}}{\sum_{k=1}^K\sum_{m=1}^M \mathbf{Y}_{mk} + \sum_{k=1}^K\sum_{m=1}^M \hat{\mathbf{Y}}_{mk}},$$

which is the F1 loss on all matrix components. Other criterion such as (negative) Macro-averaged Precision-at-k can also be defined when h is a ranker. Note that we consider all criteria to be the lower the better for the simplicity of comparison.

Notice that for a given criterion Ψ and a ground-truth label matrix $\mathbf{Y}$, a cost function that maps any predicted label matrix $\hat{\mathbf{Y}}$ to a scalar *cost* can be defined as $C_{\Psi|\mathbf{Y}}(\hat{\mathbf{Y}}) = \Psi(\mathbf{Y}, \hat{\mathbf{Y}})$. In the rest of the paper, we refer to $C_{\Psi|\mathbf{Y}}$ as C_Ψ when $\mathbf{Y}$ is clear in the context.

2.2 Related Work

Given such great variety of evaluation metrics for MLL, traditional MLL algorithms are however designed to optimize only a certain or few specific metrics. For example, algorithms such as binary relevance (Tsoumakas, Katakis, and Vlahavas 2010) and classifier chain (Read et al. 2011) that decompose MLL into K binary classification problems can arguably only focus on optimizing Hamming loss. On the other hand, label powerset approach (Tsoumakas, Katakis, and Vlahavas 2010) can merely focus on optimizing 0/1 loss since it transforms the original MLL problem into multi-class classification problem.

Nevertheless, it should be noted that even for a single MLL criterion of interest, optimizing this criterion is in fact difficult owing to the highly non-smooth nature of MLL criteria. As a result, there are currently two main families of methods

that attempt to overcome the challenge. The first common paradigm is to approach the problem by designing surrogate losses that can be optimized by efficient algorithms. For example, (Petterson and Caetano 2010; 2011) derived surrogates for the F-measure which can be optimized efficiently by SVM-style models; (Zhang and Zhou 2006; Gong et al. 2013; Nam et al. 2014) introduced different loss functions for neural networks targeting at different criteria; and (Gao and Zhou 2011; Dembczynski, Kotlowski, and Hüllermeier 2012) proposed consistent surrogates for the Rank loss. While an appropriate surrogate loss can indeed improve model performance on its corresponding criterion, deriving a surrogate for every criterion is nonetheless unsatisfactory for practical use.

Another major family of algorithms, generally termed cost-sensitive multi-label learning algorithms, tackles the problem by considering the *cost* (criterion) information in the model's training or prediction phase (Dembczynski, Cheng, and Hüllermeier 2010; Lo et al. 2011; Li and Lin 2014; Wu and Lin 2017; Huang and Lin 2017). Although these methods can adapt to different criteria more easily, current algorithms can still only deal with example-based criteria due to their restricted problem setting.

3 Proposed Method

Inspired by the rich literatures on cost-sensitive classification (Elkan 2001; Zadrozny, Langford, and Abe 2003; Li and Lin 2014; Wu and Lin 2017), we first propose a sample-weighting CSMLL framework which is able to deal with example-based criteria. We then highlight a preliminary cost-sensitive multi-label deep learning model which can be derived based on the framework. Last, to overcome the drawbacks of such simple model, we present a novel technique which can be used to optimize any given MLL criterion. The idea is coupled with deep learning to form a deep model for general cost-sensitive MLL.

3.1 Sample-weighting CSMLL Framework

In the literatures of both cost-sensitive multi-class classification as well as CSMLL, re-weighting the training samples has been a simple yet effective approach (Zadrozny, Langford, and Abe 2003; Beygelzimer, Langford, and Ravikumar 2009; Li and Lin 2014). Motivated by these work, we propose a sample-weighting framework which can be easily used to develop CSMLL algorithms.

Assume that there are K classifiers $f_k : \mathbb{R}^d \rightarrow \{0, 1\}$ each responsible for predicting a corresponding label $\hat{\mathbf{y}}_n[k]$ of a given instance $\mathbf{x}_n$. The main concept of the framework is to iteratively train these K classifiers on weighted examples, where the sample weights act as the connection to the evaluation criterion ψ. In particular, when training the k-th classifier f_k, each example $\mathbf{x}_n$ is weighted by a corresponding sample weight $w_{n,k}$. The sample weight is decided by how much cost it would incur for misclassifying the k-th label of $\mathbf{x}_n$. To estimate this *misclassification cost* for $\hat{\mathbf{y}}_n[k]$, one can assume that the other $K-1$ classifiers are fixed, and obtain their current predictions via $\{\hat{\mathbf{y}}_n[i] = f_i(\mathbf{x}_n)\}_{i \neq k}$. By having these predictions in hand, the *misclassification cost* can then be calculated as $|c^0_{n,k} - c^1_{n,k}|$, where $c^0_{n,k}$ is the cost for predicting $\hat{\mathbf{y}}_n[k]$ as zero:

$$c^0_{n,k} = \psi(\mathbf{y}_n, (\hat{\mathbf{y}}_n[1, ..., k-1], 0, \hat{\mathbf{y}}_n[k+1, ..., K])), \quad (1)$$

and $c^1_{n,k}$ is the cost for predicting $\hat{\mathbf{y}}_n[k]$ as one:

$$c^1_{n,k} = \psi(\mathbf{y}_n, (\hat{\mathbf{y}}_n[1, ..., k-1], 1, \hat{\mathbf{y}}_n[k+1, ..., K])). \quad (2)$$

By assigning $w_{n,k} = |c^0_{n,k} - c^1_{n,k}|$, the sample weights can guide f_k to focus on the examples that have greater influence on the final cost and optimize the criterion in interest. Based on the proposed framework, various CSMLL algorithms can be designed. In fact, it can also be showed that a previous CSMLL work, condensed filter tree (Li and Lin 2014), is merely a special case that utilizes this sample-weighting technique. We present this general framework for CSMLL in Algorithm 1.

Algorithm 1 Sample-weighting framework for CSMLL

```
Let f_k be a single-label classifier that predicts ŷ[k]
for m = 1 to M iterations do
    for k = 1 to K do
        for each instance (x_n, y_n) do
            Assume the other classifiers {f_i}_{i≠k} fixed
            Calculate c^0_{n,k} by Equation 1
            Calculate c^1_{n,k} by Equation 2
            w_{n,k} ← |c^0_{n,k} − c^1_{n,k}|
            Assign sample weight w_{n,k} to (x_n, y_n)
        end for
        Train f_k with the weighted examples
    end for
end for
```

In short, the key idea behind the framework is that if we are able to refer to all the other label predictions $\{f_i(\mathbf{x}_n)\}_{i \neq k}$ while training a single-label classifier f_k, the cost for wrongly predicting $\hat{\mathbf{y}}_n[k]$ can then be calculated and embedded within the sample weights to achieve cost-sensitiveness.

3.2 A Simple Cost-sensitive Multi-label Deep Learning Model

Having the sample-weighting framework for CSMLL discussed, we now turn our attention to how a simple cost-sensitive multi-label deep learning model can be developed from it. Following previous work on multi-label neural networks (Zhang and Zhou 2006; Gong et al. 2013; Nam et al. 2014), we also consider the architectures with K output nodes, where each output node o_k can be viewed as a single-label classifier that predicts the k-th label. To leverage the sample-weighting technique, one should be able to access the current predictions of the other $K-1$ labels while training the k-th label classifier. In fact, this turns out to be rather intuitive for deep learning models.

Let $h_{\theta_t} : \mathbb{R}^d \rightarrow [0, 1]^K$ denotes a multi-label neural network, where θ_t is the network weights at timestep t. At any t, the complete label prediction $\hat{\mathbf{y}}_n$ for an instance $\mathbf{x}_n$ can be simply obtained by feeding the example as the network input, i.e., $\hat{\mathbf{y}}_n = h_{\theta_t}(\mathbf{x}_n)$. With $\hat{\mathbf{y}}_n$ available, for each output node o_k (or the k-th single-label classifier), one can then follow the framework and associate each example $\mathbf{x}_n$ with the calculated sample weight $w_{n,k}$. As each o_k is considered a binary

classifier for $\hat{\mathbf{y}}[k]$, an intuitive choice of the loss function for these output nodes is the commonly known *logloss*:

$$L_{\log}(\mathbf{y}[k], \hat{\mathbf{y}}[k]) = -(\mathbf{y}[k] \log(\hat{\mathbf{y}}[k]) + (1 - \mathbf{y}[k]) \log(1 - \hat{\mathbf{y}}[k]))$$

By coupling the *logloss* with the sample weights and considering training all K label classifiers jointly, the final loss function to be optimized by the neural network at time t then becomes:

$$L_{\text{WBCE}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} w_{n,k} L_{\log}(\mathbf{y}_n[k], \hat{\mathbf{y}}_n[k]) \qquad (3)$$

We note that if the sample weights are not used, i.e., $w_{n,k} = 1$ for all n and k. Eq.3 degenerates to the conventional *binary cross entropy* (BCE), as proposed in (Nam et al. 2014). Thus, we term the loss in Eq.3 as *weighted binary cross entropy* (WBCE). It is also worthwhile to note that the sample weights $w_{n,k}$ change as the network updates. Hence, the loss function L_{WBCE} is in fact changing according to the sample weights at different timestep t. We present this simple method to train a cost-sensitive multi-label deep learning model in Algorithm 2.

Weighted BCE versus BCE By decomposing Eq.3, the weighted BCE loss for an instance $(\mathbf{x}_n, \mathbf{y}_n)$ is:

$$\sum_{k=1}^{K} w_{n,k} L_{\log}(\mathbf{y}_n[k], \hat{\mathbf{y}}_n[k]) \qquad (4)$$

From the perspective of a single instance, the original sample weights $\{w_{n,k}\}_{k=1}^{K}$ can also be viewed as the weights for each label. These weights encode the information about the relative importance of each label. That is, if $w_{n,k} > w_{n,l}$, the network should probably focus more on making the prediction on $\hat{\mathbf{y}}_n[k]$ correct even at the cost of wrongly predicting $\hat{\mathbf{y}}_n[l]$.

To compare the proposed WBCE with the ordinary BCE, we visualize the contours and gradients computed w.r.t. both losses in an illustrative two-dimensional scenario in Figure 1. It can be seen that the first label (dimension) has more influence on the cost than the second label (dimension) as the cost differences on the first axis is much greater than those on the second axis. Nonetheless, as the ordinary BCE does not take any cost information into account, the gradient it provides is unaware of the relative importance of labels. In contrast, gradient computed w.r.t. the weighted BCE loss is inclined much toward the first dimension, suggesting a direction to a relatively low-cost region.

While the weighted loss is able to take the cost information into account and provides a trajectory along the low-cost regions, the gradient direction it suggests is however relatively naive. The weights $\{w_{n,k}\}_{k=1}^{K}$ for an instance $(\mathbf{x}_n, \mathbf{y}_n)$ are in fact calculated as merely the cost differences between the current prediction $\hat{\mathbf{y}}$ and its one-bit neighbors, i.e., label vectors $\mathbf{y} \in \{\mathbf{y} | \|\mathbf{y} - \hat{\mathbf{y}}\|_1 = 1\}$. In other words, the weights are only the first-order approximation to the cost surface, and thereby the gradient suggested by the weighted loss leverages only limited cost information. Most importantly, this preliminary model can still only handle example-based criteria, as with current cost-sensitive methods.

Algorithm 2 Weighted binary cross entropy for deep learning models

Input: Training set $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ and an example-based MLL criterion ψ

Randomly initialize the neural network h_{θ_0}
repeat
  Split $\mathcal{D}$ into M random mini-batches $\{\mathcal{D}_m\}_{m=1}^{M}$
  for $m = 1$ **to** M **do**
    for each instance $(\mathbf{x}_n, \mathbf{y}_n) \in \mathcal{D}_M$ **do**
      $\hat{\mathbf{y}}_n \leftarrow h_\theta(\mathbf{x}_n)$
      for $k = 1$ **to** K **do**
        Calculate the sample weight $w_{n,k}$
      end for
    end for
    Update the network weights with gradients computed w.r.t L_{WBCE}
  end for
until converge

3.3 Locally-learned Surrogate Loss for General Cost-sensitive Multi-label Deep Learning

From Figure 1, it can be seen that the key to designing a cost-sensitive model is that the loss surface for which the model is optimized should sufficiently reflect the curvature of the criterion in interest. This in fact matches the main concept behind the previous literatures that work on developing surrogates for MLL criteria, as their main goal is also to come up with smooth approximates that preserve the characteristics of their corresponding criteria.

However, when designing a model for general criteria, it is inefficient, or even impossible, to manually derive surrogate losses for every criterion. Therefore, we call for a surrogate that can by itself *learns* to adapt to different criteria. In essence, we ask the question: can a surrogate loss be automatically learned to approximate a target criterion rather than explicitly designed by human? Nevertheless, learning to approximate the complicated MLL criteria is undoubtedly difficult, and it is certainly not preferable ending up with another complex surrogate which actually does not decrease the problem complexity. Therefore, what we aim for is an *optimization-friendly* surrogate that can however provide decent approximation to the target criterion.

Among various optimization strategies, gradient descent based algorithms are simple but powerful, and are nowadays the most prevalent practices for training modern models. One key characteristic of gradient descent based algorithms is they leverage only local information of the error surface to decide to descent direction for optimization. Therefore, if we consider using gradient descent based algorithms for the optimization of the surrogate loss, instead of a global approximation of the MLL criterion, it is arguably sufficient for the approximation to be *locally faithful*. In addition, a direct advantage gained from considering local approximation is that simpler (smoother) approximator is perhaps enough to learn an accurate local estimation.

To this end, we answer the previously posed question by a novel surrogate called locally-learned surrogate loss (LLSL), which is a (a) smooth surrogate (b) learned automatically (c)

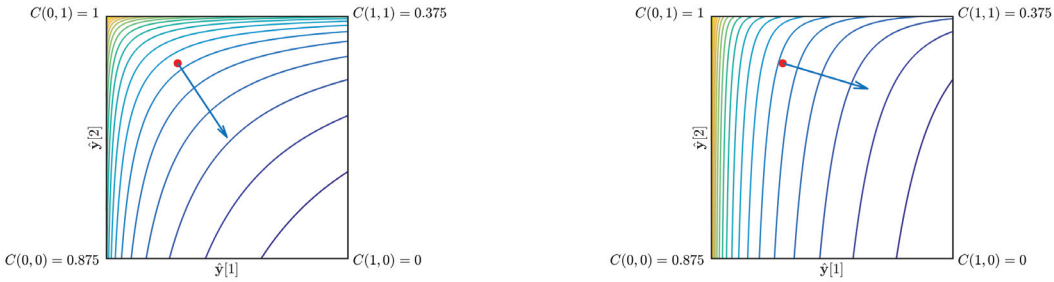

Figure 1: The contour and gradient direction of ordinary BCE (left) and WBCE (right). Note that each vertex on the square corresponds to a predicted label vector, and $C(\hat{\mathbf{y}})$ is the cost function defined by the evaluation criterion.

to provide locally faithful approximation to the criterion of interest (d) that guides the descent direction for optimization. In particular, for a given criterion of interest, we consider an iterative procedure for optimization. In each iteration, LLSL is first updated to approximate the local behavior of the criterion and is then used to determine the descent direction for model optimization in that specific iteration. Such routine is carried out repeatedly until the underlying model converges. Since LLSL is automatically learned, the key idea is in fact applicable to any MLL criterion. To the best of our knowledge, this is the first surrogate proposed that can adapt to general MLL criteria by itself. We note that as LLSL can essentially be viewed as a loss function, it can be coupled with any descent based optimization model such as deep learning to form a CSMLL model, as we shall show shortly.

Formally, let $h_{\theta_t} : \mathbb{R}^d \to \mathbb{R}^K$ be a MLL model parameterized by the weights θ_t at time t, $\mathbf{Y}$ be the ground truth label matrix where its n-th row is $\mathbf{y}_n$, and $\hat{\mathbf{Y}}$ be the predicted label matrix where its n-th row is $\hat{\mathbf{y}}_n = h_{\theta_t}(\mathbf{x}_n)$. To optimize a MLL criterion Ψ, we wish to provide a smooth surrogate to the cost surface C_Ψ of the MLL criterion near the local neighborhood of $\hat{\mathbf{Y}}$. Specifically, for an instance $(\mathbf{x}_n, \mathbf{y}_n)$, we like to estimate how the perturbation in $\hat{\mathbf{y}}_n$ affects the behavior of C_Ψ, and use this estimate to decide the descent direction for the instance. Let G be a class of models considered for the local approximation, such as linear models, and $\{\mathbf{z}_l\}_{l=1}^L$ be the local neighbors around $\hat{\mathbf{y}}_n$. We first form a dataset $\mathcal{Z}^{(t)} = \{\mathbf{z}_l, C_\Psi(\mathbf{Z}_l)\}_{l=1}^L$, where $\mathbf{Z}_l$ is the label matrix obtained by replacing the n-th row of $\hat{\mathbf{Y}}$ with $\mathbf{z}_l$. As the dataset consists of a set of local neighbors around $\hat{\mathbf{Y}}$ and their corresponding cost, LLSL can then be learned as:

$$L^{(t)}_{\text{LLSL}}(.) = \arg\min_{g \in G} \mathcal{L}(g, \mathcal{Z}^{(t)}) \tag{5}$$

where $\mathcal{L}$ is a measurement for the closeness of the learned surrogate g and the cost surface C_Ψ. We note that the learned surrogate is a regressor $g : \mathbb{R}^K \to \mathbb{R}$ who takes as input a label vector and predicts its corresponding cost. With the above formulation, the surrogate loss can be learned with different sets of local neighbors $\mathcal{Z}^{(t)}$, closeness measurement $\mathcal{L}$ and approximation models G. For example, when $\mathcal{L}$ is the square loss, and G is assumed to be linear. The surrogate loss is actually learned as a linear regression:

$$L^{(t)}_{\text{LLSL}}(\mathbf{o}) = (\arg\min_{\mathbf{w}} \sum_{(\mathbf{z},c) \in \mathcal{Z}^{(t)}} (c - \mathbf{w}^T\mathbf{z})^2)^T\mathbf{o} \tag{6}$$

where $\mathbf{o} \in \mathbb{R}^K$ is the predicted label vector space. Other types of regressors (approximators) such as polynomial regression can also be considered by proper choices on G and $\mathcal{L}$. After the surrogate loss is learned, its gradient can now be easily computed to lead the update of any gradient descent based optimization model. For instance, if the underlying MLL model is a neural network with K output nodes, continuing from Eq.6, the partial derivative of the surrogate loss computed w.r.t. each output node o_k is obtained by:

$$\frac{\partial L^{(t)}_{\text{LLSL}}(\mathbf{o})}{\partial o_k} = \mathbf{w}[k] \tag{7}$$

As the gradient for the output layer is computed, the whole network weights can then be updated by backpropagation to optimize the surrogate loss, as well as the target criterion. We present this novel deep learning model for general MLL criteria in Algorithm 3.

For the selection of the local neighbors $\{\mathbf{z}_l\}_{l=1}^L$ from which the surrogate loss is learned, a natural choice for a classifier h is $\{\mathbf{z} | \|\mathbf{z} - \hat{\mathbf{y}}_n\|_1 \leq n\}$, i.e., the label vectors whose Hamming distance to the current prediction is less than n, or, the n-bit neighbors. As for a ranker h, a natural choice would be $\{\mathbf{z} | \hat{\mathbf{y}}_n + \mathbf{p}\}$ where $\mathbf{p}$ is a random perturbation. We also note that more advanced methods for defining the neighborhood can be considered. For example, one can select the local neighbors using the cost function as the distance measurement.

While the general definition of Eq.5 and the non-smooth nature of the criteria make it hard to derive rigorous theoretical results for the proposed method, analyses on simple cases are still available. For instance, when the local learner is linear, it is rather straightforward to show that the time complexity of fitting a LLSL is polynomial in K, and the resulting learned surrogate indeed points to a descending direction under mild conditions. We conjecture that the descending direction, when coupled with careful line-search step, can then prove optimization convergence to a local minimum. We leave the work as our future direction, and first demonstrate the superiority of our method by the empirical results as we shall show in the next section.

Algorithm 3 Locally-learned surrogate loss for deep learning models

Input: Training set $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$, criterion in interest Ψ, a class of approximators G and $\mathcal{L}$

```
Randomly initialize the neural network h_{θ_0}
repeat
    Split D into M random mini-batches {D_m}_{m=1}^M
    for m = 1 to M do
        for each instance (x_n, y_n) ∈ D_M do
            ŷ_n ← h_{θ_t}(x_n)
            Collect a set local neighbors and their
            corresponding cost Z^(t) = {z_l, C_Ψ(Z_l)}_{l=1}^L
            Learn the local surrogate loss L_LLSL on Z
        end for
        Update the network with gradients computed
        w.r.t L_LLSL
    end for
until converge
```

Connection to Weighted BCE It is worthwhile to note that the proposed LLSL can be viewed as a generalization of the weighted BCE. For WBCE, the weights for an instance $(\mathbf{x}_n, \mathbf{y}_n)$ are calculated simply as $w_{n,k} = |\psi(\mathbf{y}_n, \mathbf{z}_{n,k}) - \psi(\mathbf{y}_n, \hat{\mathbf{y}}_n)|$, where $\mathbf{z}_{n,k}$ is the label vector obtained from flipping the k-th bit of $\hat{\mathbf{y}}_n$. This in fact corresponds to the LLSL framework with $\mathcal{Z} = \{\mathbf{z} | \|\mathbf{z} - \hat{\mathbf{y}}_n\|_1 = 1\}$, $\mathcal{L}$ and G being linear least squares, which utilizes the cost information along each axis separately. Nonetheless, given the complicated behavior of MLL criteria, a more sophisticated approximation that leverages the cost information around the current prediction jointly is perhaps necessary to capture the curvature of the cost surface.

4 Experiments

4.1 Experiment setup

To evaluate the effectiveness of the proposed models, we conduct experiments on a total of eleven datasets across different evaluation criteria. First, on seven benchmark datasets[1] (Tsoumakas et al. 2011), we compare our methods with: the state-of-the-art cost-sensitive MLL algorithm, condensed filter tree (CFT) (Li and Lin 2014), and existing deep learning models including BP-MLL (Zhang and Zhou 2006), WARP (Gong et al. 2013), and BCE (Nam et al. 2014). In the experiments, we consider two main classes of evaluation criteria: (a) example-based criteria: Hamming loss, Rank loss, and Example-F1 loss; (b) set-based criteria: Micro-F1 and Macro-F1. Since both CFT and the deep model coupled with WBCE can only optimize example-based criteria, their results on Micro-F1 and Macro-F1 are not available. In addition, as WBCE essentially degenerates to BCE on Hamming loss[2], the results for it are also omitted.

For fair comparison, all deep learning models are deployed with a fixed architecture. The architecture is composed of two fully-connected layers, where the number of hidden units for each layer is set to $min(d, 1024)$ with d being the input dimension. Each fully-connected layer is followed by a dropout layer with dropout ratio of 0.5. For the hidden units, Leaky ReLU is considered as the activation function.

For the proposed LLSL, we utilize several different settings to approximate the criterion of interest. Specifically, we consider three types of the underlying learners: (a) an ordinary least square regressor that learns from the one-bit neighbors; (b) an ordinary least square regressor that learns from the two-bit neighbors; (c) a second-degree polynomial regressor that learns from the two-bit neighbors. The three different settings will be referred to as $\text{LLSL}_{\text{linear-1}}$, $\text{LLSL}_{\text{linear-2}}$, and $\text{LLSL}_{\text{poly-2}}$ respectively.

In each run of the experiment, we randomly split 50%, 25%, and 25% of the dataset for training, validation, and testing. Finally, the results are averaged over 10 different random runs. Due to space limit, the relative ranking for all algorithms are shown in Figure 2, and the detailed numerical results are provided in Appendix.

To demonstrate the scalability of our model, we further compare our method to the state-of-the-art algorithm, sparse local embeddings for extreme classification (SLEEC) (Bhatia et al. 2015), which is designed specially for handling datasets with many labels. This set of experiments are conducted on four benchmark datasets[3] with many labels. We follow (Bhatia et al. 2015) to use Precision-at-k as the evaluation criterion.

4.2 Comparisons with Cost-sensitive Algorithm

From Figure 2, we see cases where even cost-insensitive deep learning models can outperform the traditional non-deep cost-sensitive algorithm CFT. This somewhat stresses the importance of studying deep learning models for MLL. In addition, when comparing to CFT, our model almost constantly reaches better performances against CFT. Most importantly, while CFT can only deal with example-based criteria, our proposed model can adapt easily toward optimizing any general criteria.

4.3 Comparisons between Deep Learning Models

Cost-insensitive versus Cost-sensitive To validate our proposed methods, we begin with the comparison between WBCE and BCE. Given that BCE is designed as the soft counterpart for Hamming loss, it shows its competence on Hamming loss in Figure 2. Nonetheless, since BCE cannot adapt to different criteria, WBCE outperforms BCE on Example-F1 and Rank loss, as shown in Figure 2. The results demonstrate that the proposed WBCE is indeed a (simple) way to make deep learning models cost-sensitive.

Although WBCE can reach better performances than BCE on Example-F1 and Rank loss, when comparing it to the proposed LLSL, LLSL wins by a large margin in Figure 2. The results justify the need for studying a more sophisticated method to make deep learning models cost-sensitive. Furthermore, we shall note that LLSL is able to optimize any given criterion while WBCE is restricted to cope with example-based criteria.

In Figure 2, when comparing the proposed LLSL to other deep learning models, our model steadily shows superior

[1] birds, emotions, enron, medical, scene, tmc2007, and yeast.

[2] All sample weights $w_{n,k}$ are equal to $\frac{1}{K}$ under *Hamming loss*.

[3] Bibtex, Delicious, EURLex-4K, and Wiki10-31K.

Table 1: Approximation error in RMSE

Datasets	Criterion	Approximators: linear-1	linear-2	poly-2	Consistent
birds	Rank	**0.073 ± 0.079**	2.150 ± 3.148	1.274 ± 2.849	✓
	Example-F1	0.034 ± 0.048	0.053 ± 0.044	**0.023 ± 0.032**	✓
	Micro-F1	0.146 ± 0.046	**0.116 ± 0.021**	0.132 ± 0.033	✓
	Macro-F1	0.485 ± 0.025	0.551 ± 0.061	**0.450 ± 0.064**	✓
emotions	Ranking	**0.170 ± 0.072**	0.613 ± 0.329	0.990 ± 0.560	✓
	Example-F1	0.159 ± 0.070	0.173 ± 0.033	**0.093 ± 0.029**	✓
	Micro-F1	0.227 ± 0.054	0.242 ± 0.088	**0.152 ± 0.026**	✓
	Macro-F1	0.209 ± 0.047	0.187 ± 0.114	**0.172 ± 0.016**	✓
scene	Ranking	**0.110 ± 0.058**	0.603 ± 0.291	0.478 ± 0.495	✓
	Example-F1	0.154 ± 0.088	0.133 ± 0.072	**0.123 ± 0.067**	✗
	Micro-F1	0.390 ± 0.137	0.355 ± 0.140	**0.225 ± 0.175**	✓
	Macro-F1	0.200 ± 0.153	0.184 ± 0.090	**0.172 ± 0.089**	✓

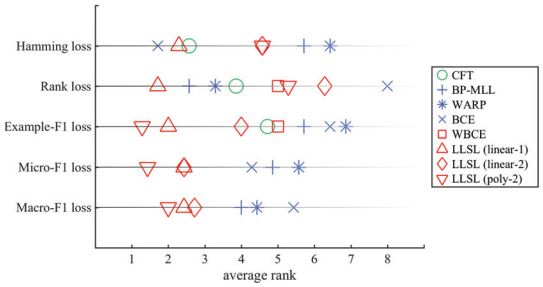

Figure 2: The average rank of different models on different criteria. The lower (left) the rank, the better the performance.

Table 2: Results on datasets with many labels

Datasets	(n) P@k	Algorithms: SLEEC	BCE loss	Locally-learned loss
Bibtex	(n) P@1	**0.3492**	0.4482	0.3647
	(n) P@3	**0.6036**	0.6698	0.6157
	(n) P@5	**0.7113**	0.7612	0.7270
Delicious	(n) P@1	0.3241	0.3194	**0.2980**
	(n) P@3	0.3862	0.3778	**0.3581**
	(n) P@5	0.4344	0.4282	**0.4081**
EURLex-4K	(n) P@1	**0.2074**	0.2345	0.2287
	(n) P@3	**0.3570**	0.3654	0.3579
	(n) P@5	0.4767	0.4731	**0.4677**
Wiki10-31K	(n) P@1	0.1412	0.1442	**0.1396**
	(n) P@3	0.2702	0.2665	**0.2563**
	(n) P@5	0.3730	0.3631	**0.3586**

performances across different criteria, while the other models can only sometimes reach the best result on the criteria for which they are designed to optimize. The results again demonstrate the ability of LLSL to adapt to general criteria.

The Approximators for LLSL In order to gain more insights on the proposed LLSL, we further compare the performances of $\text{LLSL}_{\text{linear-1}}$, $\text{LLSL}_{\text{linear-2}}$, and $\text{LLSL}_{\text{poly-2}}$ to see how different underlying approximators behave on different criteria. In Figure 2, we see that $\text{LLSL}_{\text{linear-1}}$ performs the best against the others on Hamming loss and Rank loss. On the other hand, $\text{LLSL}_{\text{poly-2}}$ outperforms the other two on Example-F1, Micro-F1 and Macro-F1. $\text{LLSL}_{\text{linear-2}}$ can only reach the best result on few cases.

To explain the results, we take a step further to investigate the reason behind so. In particular, we investigate the goodness of the estimations learned by different approximators. The goodness is measured by the RMSE between the true cost and the estimated cost on a set of points sampled from the local neighborhood where the estimation is learned. The results are reported in Table 1. From the table, it can be seen that the performance of a model is strongly correlated to how well the underlying local approximation to the criterion is. That is, the lower the approximation error, the better the model performs. We mark every row in the table as *consistent* if the above holds. The results also suggest a general guideline to choose suitable approximator for LLSL when it comes to different criteria. While the bias of the estimation varies with the choice of local learner, we observe that the estimation error decreases as the optimization proceeds.

Interestingly, while we generally believe that more sophisticated approximator may provide more faithful estimation to the criterion of interest, it is shown that the relatively simple *linear-1* approximator gives the best estimation to Hamming loss and Rank loss. The interesting finding can be explained by the inherent nature of the two criteria. In other words, as the minimization of Hamming loss and Rank loss can actually be decomposed by labels (Dembczynski, Kotlowski, and Hüllermeier 2012), a linear approximator that treats each label independently might be enough good for estimating these criteria. Nevertheless, for the other more complicated criteria such as Macro-F1, a more sophisticated approximation is required for better performance.

4.4 Scaling Up to Datasets with Many Labels

On large scale datasets, we demonstrate the scalability and flexibility of the proposed LLSL by using it to fine-tune deep models that are originally pre-trained on BCE loss. In table 2, BCE stands for models that are trained with the conventional BCE loss, and LLSL stands for models that are fine-tuned with the proposed locally-learned surrogate loss. In the experiments, we find the gradients of the surrogate loss learned for Precision-at-k appear to be very sparse, resulting in slower convergence. An useful practical finding to tackle such issue is to optimize the mixture loss between Hamming loss, repre-

sented by BCE, and the learned surrogate loss. In a high-level sense, as Hamming loss treats each label equally, it in fact encodes the global information about the target metric. Thus, it can be viewed as a regularizer to the locally-learned surrogate loss which exploits the local information. As long as we could find the sweet spot between the two losses, optimizing the mixture loss between them works well in practice. Furthermore, following the idea, LLSL can actually be mixed with other objectives such as the proposed WBCE. A joint optimization between LLSL and other loss functions might also lead to an interesting future direction.

In Table 2, it is shown that our proposed LLSL successfully improves the performances of cost-insensitive models on large scale datasets. In addition, our model reaches competitive performance to the state-of-the-art. This not only demonstrates the scalability of our proposed method, but also shows the capability for LLSL to cope with criterion like Precision-at-k when the deep model is a ranker.

5 Conclusion

We propose a novel locally-learned surrogate loss (LLSL) that can adapt toward optimizing general MLL criteria by learning local approximation to the criterion of interest. The learned surrogate loss is then coupled with deep learning model to optimize the target criterion. The proposed LLSL can successfully capture the local behavior of the target MLL criterion and in turn provides cost-aware gradients guiding the network updates. Extensive experimental results show that our proposed deep model achieves outstanding performances against the state-of-the-art methods.

Acknowledgements

We thank the anonymous reviewers and the members of NTU CLLab for valuable suggestions. This material is based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-15-1-4012, and by the Ministry of Science and Technology of Taiwan under number MOST 103-2221-E- 002-149-MY3

References

Beygelzimer, A.; Langford, J.; and Ravikumar, P. 2009. Error-correcting tournaments. *CoRR* abs/0902.3176.

Bhatia, K.; Jain, H.; Kar, P.; Varma, M.; and Jain, P. 2015. Sparse local embeddings for extreme multi-label classification. In *NIPS 2015*.

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

Dembczynski, K.; Cheng, W.; and Hüllermeier, E. 2010. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML 2010*.

Dembczynski, K.; Kotlowski, W.; and Hüllermeier, E. 2012. Consistent multilabel ranking through univariate losses. In *ICML 2012*.

Elkan, C. 2001. The foundations of cost-sensitive learning. In *IJCAI 2001*.

Gao, W., and Zhou, Z. 2011. On the consistency of multi-label learning. In *COLT 2011*.

Gong, Y.; Jia, Y.; Leung, T.; Toshev, A.; and Ioffe, S. 2013. Deep convolutional ranking for multilabel image annotation. *CoRR* abs/1312.4894.

Huang, K.-H., and Lin, H.-T. 2017. Cost-sensitive label embedding for multi-label classification. *Machine Learning* 106(9–10):1725–1746.

Li, C.-L., and Lin, H.-T. 2014. Condensed filter tree for cost-sensitive multi-label classification. In *ICML 2014*.

Lo, H.; Wang, J.; Wang, H.; and Lin, S. 2011. Cost-sensitive multi-label learning for audio tag annotation and retrieval. *IEEE Trans. Multimedia* 13(3):518–529.

Madjarov, G.; Kocev, D.; Gjorgjevikj, D.; and Dzeroski, S. 2012. An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition* 45(9):3084–3104.

Nam, J.; Kim, J.; Loza Mencía, E.; Gurevych, I.; and Fürnkranz, J. 2014. Large-scale multi-label text classification - revisiting neural networks. In *ECML PKDD 2014*.

Petterson, J., and Caetano, T. S. 2010. Reverse multi-label learning. In *NIPS 2010*.

Petterson, J., and Caetano, T. S. 2011. Submodular multi-label learning. In *NIPS 2011*.

Qi, G.; Hua, X.; Rui, Y.; Tang, J.; Mei, T.; and Zhang, H. 2007. Correlative multi-label video annotation. In *Proceedings of the 15th International Conference on Multimedia 2007*.

Read, J.; Pfahringer, B.; Holmes, G.; and Frank, E. 2011. Classifier chains for multi-label classification. *Machine Learning* 85(3):333–359.

Schapire, R. E., and Singer, Y. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning* 39(2/3):135–168.

Tsoumakas, G.; Xioufis, E. S.; Vilcek, J.; and Vlahavas, I. P. 2011. MULAN: A java library for multi-label learning. *Journal of Machine Learning Research* 12:2411–2414.

Tsoumakas, G.; Katakis, I.; and Vlahavas, I. P. 2010. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook, 2nd ed.* 667–685.

Wu, Y.-P., and Lin, H.-T. 2017. Progressive k-labelsets for cost-sensitive multi-label classification. *Machine Learning* 106(5):671–694.

Zadrozny, B.; Langford, J.; and Abe, N. 2003. Cost-sensitive learning by cost-proportionate example weighting. In *ICDM 2003*, 435.

Zhang, M., and Zhou, Z. 2006. Multi-label neural networks with applications to functional genomics and text categorization. *IEEE Trans. Knowl. Data Eng.* 18(10):1338–1351.