Deep Learning with a Rethinking Structure
for Multi-label Classification

Yao-Yuan Yang   B01902066@NTU.EDU.TW
Yi-An Lin   ANDYLIN514@GMAIL.COM
Hong-Min Chu   R04922031@CSIE.NTU.EDU.TW
Hsuan-Tien Lin   HTLIN@CSIE.NTU.EDU.TW
Department of Computer Science and Information Engineering, National Taiwan University

Abstract

Multi-label classification (MLC) is an important class of machine learning problems that come with a wide spectrum of applications, each demanding a possibly different evaluation criterion. When solving the MLC problems, we generally expect the learning algorithm to take the hidden correlation of the labels into account to improve the prediction performance. Extracting the hidden correlation is generally a challenging task. In this work, we propose a novel deep learning framework to better extract the hidden correlation with the help of the memory structure within recurrent neural networks. The memory stores the temporary guesses on the labels and effectively allows the framework to rethink about the goodness and correlation of the guesses before making the final prediction. Furthermore, the rethinking process makes it easy to adapt to different evaluation criteria to match real-world application needs. In particular, the framework can be trained in an end-to-end style with respect to any given MLC evaluation criteria. The end-to-end design can be seamlessly combined with other deep learning techniques to conquer challenging MLC problems like image tagging. Experimental results across many real-world data sets justify that the rethinking framework indeed improves MLC performance across different evaluation criteria and leads to superior performance over state-of-the-art MLC algorithms.

Keywords: multi-label, deep learning, cost-sensitive

1. Introduction

Human beings master our skills for a given problem by working on and thinking through the same problem over and over again. When a difficult problem is given to us, multiple attempts would have gone through our mind to simulate different possibilities. During this period, our understanding to the problem gets deeper, which in term allows us to propose a better solution in the end. The deeper understanding comes from a piece of consolidated knowledge within our memory, which records how we build up the problem context with processing and predicting during the "rethinking" attempts. The human-rethinking model above inspires us to design a novel deep learning model for machine-rethinking, which is equipped with a memory structure to better solve the multi-label classification (MLC) problem.

The MLC problem aims to attach multiple relevant labels to an input instance simultaneously, and matches various application scenarios, such as tagging songs with a subset of emotions (Trohidis et al., 2008) or labeling images with objects (Wang et al., 2016). Those

MLC applications typically come with an important property called label correlation (Cheng et al., 2010; Huang and Zhou, 2012). For instance, when tagging songs with emotions, "angry" is negatively correlated with "happy"; when labeling images, the existence of a desktop computer probably indicates the co-existence of a keyboard and a mouse. Many existing MLC works implicitly or explicitly take label correlation into account to better solve MLC problems (Cheng et al., 2010).

Label correlation is also known to be important for human when solving MLC problems (Bar, 2004). For instance, when solving an image labeling task upon entering a new room, we might notice some more obvious objects like sofa, dining table and wooden floor at the first glance. Such a combination of objects hints us of a living room, which helps us better recognize the "geese" on the sofa to be stuffed animals instead of real ones. The recognition route from the sofa to the living room to stuffed animals require rethinking about the correlation of the predictions step by step. Our proposed machine-rethinking model mimics this human-rethinking process to digest label correlation and solve MLC problems more accurately.

Next, we introduce some representative MLC algorithms before connecting them to our proposed machine-rethinking model. Binary relevance (BR) (Tsoumakas et al., 2009) is a baseline MLC algorithm that does not consider label correlation. For each label, BR learns a binary classifier to predict the label's relevance independently. Classifier chain (CC) (Read et al., 2009) extends BR by taking some label correlation into account. CC links the binary classifiers as a chain and feeds the predictions of the earlier classifiers as features to the latter classifiers. The latter classifiers can thus utilize (the correlation to) the earlier predictions to form better predictions.

The design of CC can be viewed as a memory mechanism that stores the label predictions of the earlier classifiers. Convolutional neural network recurrent neural network (CNN-RNN) (Wang et al., 2016) and order-free RNN with Visual Attention (Att-RNN) (Chen et al., 2017) algorithms extend CC by replacing the mechanism with a more sophisticated memory-based model—recurrent neural network (RNN). By adopting different variations of RNN (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), the memory can store more sophisticated concepts beyond earlier predictions. In addition, adopting RNN allows the algorithms to solve tasks like image labeling more effectively via end-to-end training with other deep learning architectures (e.g., convolutional neural network in CNN-RNN).

The CC-family algorithms above for utilizing label correlation are reported to achieve better performance than BR (Read et al., 2009; Wang et al., 2016). Nevertheless, given that the predictions happen sequentially within a chain, those algorithms generally suffer from the issue of label ordering. In particular, classifiers in different positions of the chain receive different levels of information. The last classifier predicts with all information from other classifiers while the first classifier label predicts with no other information. Att-RNN addresses this issue with beam search to approximate the optimal ordering of the labels, and dynamic programming based classifier chain (CC-DP) (Liu and Tsang, 2015) searches for the optimal ordering with dynamic programming. Both Att-RNN and CC-DP can be time-consuming when searching for the optimal ordering, and even after identifying a good ordering, the label correlation information is still not shared equally during the prediction process.

Our proposed deep learning model, called RethinkNet, tackles the label ordering issue by viewing CC differently. By considering CC-family algorithms as a rethinking model based on the partial predictions from earlier classifiers, we propose to *fully* memorize the temporary predictions from *all* classifiers during the rethinking process. That is, instead of forming a chain of binary classifiers, we form a chain of *multi-label* classifiers as a sequence of rethinking. RethinkNet learns to form preliminary guesses in earlier classifiers of the chain, store those guesses in the memory and then correct those guesses in latter classifiers with label correlation. Similar to CNN-RNN and Att-RNN, RethinkNet adopts RNN for making memory-based sequential prediction. We design a global memory for RethinkNet to store the information about label correlation, and the global memory allows all classifiers to share the same information without suffering from the label ordering issue.

Another advantage of RethinkNet is to tackle an important real-world need of Cost-Sensitive Multi-Label Classification (CSMLC) (Li and Lin, 2014). In particular, different MLC applications often require different evaluation criteria. To be widely useful for a broad spectrum of applications, it is thus important to design CSMLC algorithms, which takes the criteria (cost) into account during learning and can thus adapt to different costs easily. State-of-the-art CSMLC algorithms include condensed filter tree (CFT) (Li and Lin, 2014) and probabilistic classifier chain (PCC) (Cheng et al., 2010). PCC extends CC to CSMLC by making Bayes optimal predictions according to the criterion. CFT is also extended from CC, but achieves cost-sensitivity by converting the criterion to importance weights when training each binary classifier within CC. The conversion step in CFT generally requires knowing the predictions of all classifiers, which has readily been stored within the memory or RethinkNet. Thus, RethinkNet can be easily combined with the importance-weighting idea within CFT to achieve cost-sensitivity.

Extensive experiments across real-world data sets validate that RethinkNet indeed improves MLC performance across different evaluation criteria and is superior to state-of-the-art MLC and CSMLC algorithms. Furthermore, for image labeling, experimental results demonstrate that RethinkNet outperforms both CNN-RNN and Att-RNN. The results justify the usefulness of RethinkNet.

The paper is organized as follows. Section 2 sets up the problem and introduces concurrent RNN models. Section 3 illustrates the proposed RethinkNet framework. Section 4 contains extensive experimental results to demonstrate the benefits of RethinkNet. Finally, Section 5 concludes our findings.

2. Preliminary

In the multi-label classification (MLC) problem, the goal is to attach multiple labels to a feature vector $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$. Let there be a total of K labels. The labels are represented by a label vector $\mathbf{y} \in \mathcal{Y} \subseteq \{0,1\}^K$, where the k-th bit $\mathbf{y}[k] = 1$ if and only if the k-th label is relevant to $\mathbf{x}$. We call $\mathcal{X}$ and $\mathcal{Y}$ the feature space and the label space, respectively.

During training, a MLC algorithm takes the training data set $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ that contains N examples with $\mathbf{x}_n \in \mathcal{X}$ and $\mathbf{y}_n \in \mathcal{Y}$ to learn a classifier $f \colon \mathcal{X} \rightarrow \mathcal{Y}$. The classifier maps a feature vector $\mathbf{x} \in \mathcal{X}$ to its predicted label vector in $\mathcal{Y}$. For testing, $\bar{N}$ test examples $\{(\bar{\mathbf{x}}_n, \bar{\mathbf{y}}_n)\}_{n=1}^{\bar{N}}$ are drawn from the same distribution that generated the training data set $\mathcal{D}$. The goal of an MLC algorithm is to learn a classifier f such that the predicted

vectors $\{\hat{\mathbf{y}}_n\}_{n=1}^{\bar{N}} = \{f(\bar{\mathbf{x}}_n)\}_{n=1}^{\bar{N}}$ such that $\{\hat{\mathbf{y}}_n\}_{n=1}^{\bar{N}}$ are close to the ground truth vectors $\{\bar{\mathbf{y}}_n\}_{n=1}^{\bar{N}}$.

The closeness between label vectors is measured by the evaluation criteria. Different applications require possibly different evaluation criteria, which calls for a more general setup called cost-sensitive multi-label classification (CSMLC). In this work, we follow the setting from previous works (Li and Lin, 2014; Cheng et al., 2010) and consider a specific family of evaluation criteria. This family of criteria measures the closeness between a single predicted vector $\hat{\mathbf{y}}$ and a single ground truth vector $\mathbf{y}$. To be more specific, these criteria can be written as a cost function $C\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, where $C(\mathbf{y}, \hat{\mathbf{y}})$ represents the cost (difference) of predicting $\mathbf{y}$ as $\hat{\mathbf{y}}$. This way, classifier f can be evaluated by the average cost $\frac{1}{\bar{N}} \sum_{n=1}^{\bar{N}} C(\bar{\mathbf{y}}_n, \hat{\mathbf{y}}_n)$ on the test examples.

In CSMLC setting, we assume the criterion for evaluation to be known before training. That is, CSMLC algorithms can learn f with both the training data set $\mathcal{D}$ and the cost function C, and should be able to adapt to different C easily. By using this additional cost information, CSMLC algorithms aim at minimizing the expected cost on the test data set $\mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}}[C(\mathbf{y}, f(\mathbf{x}))]$. Common cost functions are listed as follows. Note that some common 'cost' functions use higher output to represent a better prediction—we call those *score* functions to differentiate them from usual cost (*loss*) functions that use lower output to represent a better prediction.

- Hamming loss: $C_H(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{K} \sum_{k=1}^{K} [\![\mathbf{y}[k] \neq \hat{\mathbf{y}}[k]]\!]$ where $[\![\cdot]\!]$ is the indicator function.

- F1 score: $C_F(\mathbf{y}, \hat{\mathbf{y}}) = \frac{2\|\mathbf{y} \cap \hat{\mathbf{y}}\|_1}{\|\mathbf{y}\|_1 + \|\hat{\mathbf{y}}\|_1}$, where $\|\mathbf{y}\|_1$ actually equals the number of 1'ss in label vector $\mathbf{y}$.

- Accuracy score: $C_A(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\|\mathbf{y} \cap \hat{\mathbf{y}}\|_1}{\|\mathbf{y} \cup \hat{\mathbf{y}}\|_1}$

- Rank loss: $C_R(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{\mathbf{y}[i] > \mathbf{y}[j]} \left([\![\hat{\mathbf{y}}[i] < \hat{\mathbf{y}}[j]]\!] + \frac{1}{2} [\![\hat{\mathbf{y}}[i] = \hat{\mathbf{y}}[j]]\!] \right)$

2.1. Related Work

There are many different families of MLC algorithms. In this work, we focuses on the chain-based algorithms, which make predictions label by label sequentially and each prediction take previously-predicted labels as inputs.

Classifier chain (CC) (Read et al., 2009) is the most classic algorithm in the chain-based family. CC learns a sub-classifier per label. In particular, CC predicts the label one by one, the prediction of previous labels are fed to the next sub-classifier to predict the next label. CC allows sub-classifiers to utilize label correlation by building correlations between sub-classifiers.

However, deciding the label ordering for CC can be difficult and the label ordering is crucial to the performance (Read et al., 2009, 2014; Goncalves et al., 2013; Liu and Tsang, 2015). The sub-classifier in the later part of the chain can receive more information from other sub-classifiers while others receive less. Algorithms have been developed to solve the label ordering problem by using different ways to search for a better ordering of the labels. These algorithms include one that uses monte carlo methods (Read et al., 2014) and genetic algorithm (Goncalves et al., 2013). A recent work called dynamic programming based classifier chain (CC-DP) (Liu and Tsang, 2015) is proposed that by using dynamic

programming. However, the time complexity for CC-DP is still as large as $O(K^3Nd)$ and the derivation is limited using support vector machine (SVM) as sub-classifier.

For the CSMLC setup, there are two algorithms developed based on CC, the probabilistic classifier chain (PCC) (Cheng et al., 2010) and condense filter tree (CFT) (Li and Lin, 2014). PCC learns a CC classifier during training. During testing, PCC make a bayes optimal decision with respect to the given cost function. This step of making inference can be time consuming, thus efficient inference rule for each cost function are needed to be derived individually. Efficient inference rule for F1 score and Rank loss are derived in (Dembczynski et al., 2012, 2011). Albeit, there is no known inference rule for Accuracy score. For CFT, it transforms the cost information into instance weight. Through a multi-step training process, CFT gradually fine-tune the weight assigned and itself cost-sensitive during training. CFT does not require the inference rule to be derived for each cost function, but the multi-step training process is still time consuming. Also CFT is not able to be combined with deep learning architectures for image or sound data sets.

CC can be interpreted as a deep learning architecture (Read and Hollmén, 2014). Each layer of the deep learning architecture predicts a label and passes the prediction to the next layer. By turning the idea of building correlations between sub-classifiers into maintaining a memory between sub-classifiers, the convolutional neural network recurrent neural network (CNN-RNN) (Wang et al., 2016) algorithm further adapt recurrent neural network (RNN) to generalize the deep learning architecture for CC. CNN-RNN treats each prediction of the label as a time step in the RNN. CNN-RNN also demonstrated that with this architecture, they are able to incorporate with convolutional neural networks (CNN) and produces experimental results that outperforms traditional MLC algorithms. The order-free RNN with Visual Attention (Att-RNN) (Chen et al., 2017) algorithms is an improvement over CNN-RNN. Att-RNN incorporated the attention model (Xu et al., 2015) to enhance their performance and interoperability. Also, Att-RNN uses the beam search method to search for a better ordering of the label. But both Att-RNN and CNN-RNN are not cost-sensitive, thus they are unable to utilize the cost information.

To summarize, there are three key aspects in developing a MLC algorithm. Whether the algorithm is able to effectively utilize the label correlation information, whether the algorithm is able to consider the cost information and whether the algorithm is extendable to deep learning structures for modern application. In terms of utilizing label correlation information, current chain based MLC algorithms has to solve the label ordering due to the sequential nature of the chain. The first label and the last label in the chain are destined to receive different amount of information from other labels. Chain based algorithms are generally made extendable to other deep learning architectures by adapting RNN. However, there is currently no MLC algorithm in the chain based family that are designed both considering cost information as well as being extendable with other deep learning architectures. In the next section, we will introduce the RNN to understand how it is designed.

2.2. Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is a class of neural network model that are designed to solve sequence prediction problem. In sequence prediction problem, let there be B iterations. There is an output for each iteration. RNN uses memory to pass information from one

iteration to the next iteration. RNN learns two transformations and all iterations shares these two transformations. The feature transformation $\mathbf{U}(\cdot)$ takes in the feature vector and transforms it to an output space. The memory transformation $\mathbf{W}(\cdot)$ takes in the output from the previous iteration and transform it to the same output space as the output of $\mathbf{U}$. For $1 \leq i \leq B$, we use $\mathbf{x}^{(i)}$ to represent the feature vector for the i-th iteration, and use $\mathbf{o}^{(i)}$ to represent its output vector. Formally, for $2 \leq i \leq B$, and let $\sigma(\cdot)$ be the activation unction, the RNN model can be written as $\mathbf{o}^{(1)} = \sigma(U(\mathbf{x}^{(1)})), \quad \mathbf{o}^{(i)} = \sigma(U(\mathbf{x}^{(i)}) + W(\mathbf{o}^{(i-1)}))$.

The basic variation of RNN is called simple RNN (SRN) (Elman, 1990; Jordan, 1997). SRN assumes $\mathbf{W}$ and $\mathbf{U}$ to be linear transformation. SRN is able to link information between iterations, but it can be hard to train due to the decay of gradient (Hochreiter et al., 2001). Several variations of RNN had been proposed to solve this problem. Long short term memory (LSTM) (Hochreiter and Schmidhuber, 1997) and gated recurrent unit (GRU) (Cho et al., 2014) solve this problem by redesigning the neural network architecture. Iterative RNN (IRNN) (Le et al., 2015) proposed that the problem can be solved by different initialization of the memory matrix in SRN.

In sum, RNN provides a foundation for sharing information between a sequence of label predictions using memory. In the next section, we will demonstrate how we utilize RNN to develope a novel chain based MLC algorithms that addresses the three key aspects for MLC algorithm mentioned in the previous subsection.

3. Proposed Model

The idea of improving prediction result by iteratively polishing the prediction is the "rethinking" process. This process can be taken as a sequence prediction problem and RethinkNet adopts RNN to model this process.

Figure 1 illustrates how RethinkNet is designed. RethinkNet is composed of an RNN layer and a dense (fully connected) layer. The dense layer learns a label embedding to transform the output of RNN layer to label vector. The RNN layer is used to model the "rethinking" process. The RNN layer goes through a total of B iterations. All iterations in RNN share the same feature vector since they are solving the same MLC problem. The output of the RNN layer at t-th iteration is $\hat{\mathbf{o}}^{(t)}$, which represents the embedding of the label vector $\hat{\mathbf{y}}^{(t)}$. Each $\hat{\mathbf{o}}^{(t)}$ is passed down to $(t+1)$-th iteration in the RNN layer.

Each iteration in RNN layer represents to a rethink iteration. In the first iteration, RethinkNet makes a prediction base on the feature vector alone, which targets at labels that are easier to identify. The first prediction $\hat{\mathbf{y}}^{(1)}$ is similar to BR, which predicts each label independently without the information of other labels. From the second iteration, RethinkNet begins to use the result from the previous iteration to make better predictions $\hat{\mathbf{y}}^{(2)} \ldots \hat{\mathbf{y}}^{(B)}$. $\hat{\mathbf{y}}^{(B)}$ is taken as the final prediction $\hat{\mathbf{y}}$. As RethinkNet polishes the prediction, difficult labels would eventually be labeled more accurately.

3.1. Modeling Label Correlation

RethinkNet models label correlation in the memory of the RNN layer. To simplify the illustration, we assume that the activation function $\sigma(\cdot)$ is sigmoid function and the dense layer is an identity transformation. SRN is used in the RNN layer because other forms of RNN share similar property since they are originated from SRN. In SRN, the memory and feature
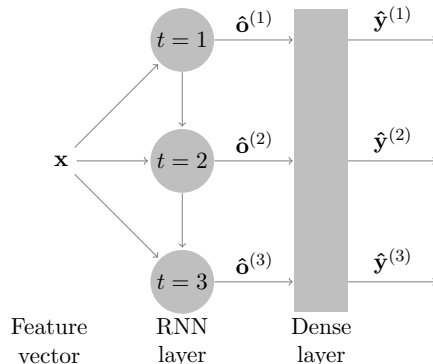
Figure 1: The architecture of the proposed RethinkNet model.

transformations are represented as matrices $\mathbf{W} \in R^{K \times K}$ and $\mathbf{U} \in R^{K \times d}$ respectively. The RNN layer output $\hat{\mathbf{o}}^{(t)}$ will be a label vector with length K.

Under the setting, the predicted label vector is $\hat{\mathbf{y}}^{(t)} = \hat{\mathbf{o}}^{(t)} = \sigma(\mathbf{U}\mathbf{x} + \mathbf{W}\hat{\mathbf{o}}^{(t-1)})$. This equation can be separated into two parts, the feature term $\mathbf{U}\mathbf{x}$, which makes the prediction like BR, and the memory term $\mathbf{W}\hat{\mathbf{o}}^{(t-1)}$, which transforms the previous prediction to the current label vector space. This memory transformation serves as the model for label correlation. $\mathbf{W}[i, j]$ represents i-th row and j-th column of $\mathbf{W}$ and it represents the correlation between i-th and j-th label. The prediction of j-th label is the combination of $(\mathbf{U}\mathbf{x})[j]$ and $\hat{\mathbf{o}}^{(t)}[j] = \sum_{i=1}^{K} \hat{\mathbf{o}}^{(t-1)}[i] * \mathbf{W}[i, j]$. If we predict $\hat{\mathbf{o}}^{(t-1)}[i]$ as relevant at $(t-1)$-th iteration and $\mathbf{W}[i, j]$ is high, it indicates that the j-th label is more likely to be relevant. If $\mathbf{W}[i, j]$ is negative, this indicates that the i-th label and j-th label may be negatively correlated.

Figure 2 plots the learned memory transformation matrix and the correlation coefficient of the labels. We can clearly see that RethinkNet is able to capture the label correlation information, although we also found that such result in some data set can be noisy. The finding suggests that $\mathbf{W}$ may carry not only label correlation but also other data set factors. For example, the RNN model may learn that the prediction of a certain label does not come with a high accuracy. Therefore, even if another label is highly correlated with this one, the model would not give it a high weight.
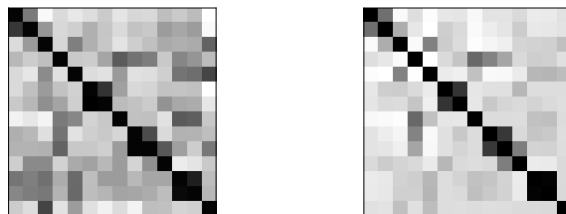
(a) memory transform   (b) correlation coefficient

Figure 2: The trained memory transformation matrix $\mathbf{W}$ with SRN and the correlation coefficient of the yeast data set. Each cell represents the correlation between two labels. Each row of the memory weight is normalized for the diagonal element to be 1 so it can be compared with correlation coefficient.

3.2. Cost-Sensitive Reweighted Loss Function

Cost information is another important piece of information that should be considered when solving an MLC problem. Different cost function values each label differently, so we should set the importance of each label differently. One way to encode such property is to weight each label in the loss function according to its importance. The problem would become how to estimate the label importance.

The difference between a label predicted correctly and incorrectly under the cost function can be used to estimate the importance of the label. To evaluate the importance of a single label, knowing other labels is required for most costs. We leverage the sequential nature of RethinkNet where temporary predictions are made between each of the iterations. Using the temporary prediction to fill out all other labels, we will be able to estimate the importance of each label.

The weight of each label is designed as equation (1). For $t = 1$, where no prior prediction exists, the labels are set with equal importance. For $t > 1$, we use $\hat{\mathbf{y}}_n^{(t)}[i]_0$ and $\hat{\mathbf{y}}_n^{(t)}[i]_1$ to represent the label vector $\hat{\mathbf{y}}_n^{(t)}$ when the i-th label is set to 0 and 1 respectively. The weight of the i-th label is the cost difference between $\hat{\mathbf{y}}_n^{(t)}[i]_0$ and $\hat{\mathbf{y}}_n^{(t)}[i]_1$. This weighting approach can be used to estimate the effect of each label under current prediction with the given cost function. Such method echos the design of CFT (Li and Lin, 2014).

$$\mathbf{w}_n^{(1)}[i] = 1, \quad \mathbf{w}_n^{(t)}[i] = |C(\mathbf{y}_n, \hat{\mathbf{y}}_n^{(t-1)}[i]_0) - C(\mathbf{y}_n, \hat{\mathbf{y}}_n^{(t-1)}[i]_1)| \tag{1}$$

In usual MLC setting, the loss function adopted for neural network is binary cross-entropy. To accept the weight in loss function, we formulated the weighted binary cross-entropy as Equation (2). For $t = 1$, the weight for all labels are set to 1 since there is no prediction to reference. For $t = 2, \dots K$, the weights are updated using the previous prediction. Note that when the given cost function is Hamming loss, the labels in each iteration are weighted the same and the weighting is reduced to the same as in BR.

$$\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{B} \sum_{i=1}^{K} -\mathbf{w}_n^{(t)}[i](\mathbf{y}_n[i] \log p(\hat{\mathbf{y}}_n^{(t)}[i]) + (1 - \mathbf{y}_n[i]) \log(1 - p(\hat{\mathbf{y}}_n^{(t)}[i]))) \tag{2}$$

Table 1: Comparison between MLC algorithms.

algorithm	memory content	cost-sensitivity	feature extraction
BR	-	-	-
CC	former prediction	-	-
CC-DP	optimal ordered prediction	-	-
PCC	former prediction	v	-
CFT	former prediction	v	-
CNN-RNN	former prediction in RNN	-	CNN
Att-RNN	former prediction in RNN	-	CNN + attention
RethinkNet	full prediction in RNN	v	general NN

Table 1 shows a comparison between MLC algorithms. RethinkNet is able to consider both the label correlation and the cost information. Its structure allows it to be extended easily with other neural network for advance feature extraction, so it can be easily adopted to solve image labeling problems. In Section 4, we will demonstrate that these advantages can be turned into better performance.

4. Experiments

The experiments are evaluated on 11 real-world data sets (Tsoumakas et al., 2011). The data set statistics are shown in Table 2. The data set is split with 75% training and 25% testing randomly and the feature vectors are scaled to $[0, 1]$. Experiments are repeated 10 times with the mean and standard error (ste) of the testing loss/score recorded. The results are evaluated with Hamming loss, Rank loss, F1 score, Accuracy score (Li and Lin, 2014). We use $\downarrow$ to indicate the lower value for the criterion is better and $\uparrow$ to indicate the higher value is better.

RethinkNet is implemented with TENSORFLOW (Abadi et al., 2015). The RNN layer can be interchanged with different variations of RNN including SRN, LSTM GRU and IRNN. A 25% dropout on the memory matrix of RNN is added. A single fully-connected layer is used for the dense layer and Nesterov Adam (Nadam) (Dozat, 2016) is used to optimize the model. The model is trained until converges or reach $1,000$ epochs and the batch size is fixed to 256. We added L2 regularization to the training parameters and the regularization strength is search within $(10^{-8}, \ldots, 10^{-1})$ with three-fold cross-validation. The implementation of RethinkNet can be found here [1].

Table 2: Statistics on multi-label data sets

data set	feature dim.	label dim.	data points	cardinality	density
emotions	72	6	593	1.869	0.311
scene	2407	6	2407	1.074	0.179
yeast	2417	14	2417	4.237	0.303
birds	260	19	645	1.014	0.053
tmc2007	500	22	28596	2.158	0.098
arts1	23146	26	7484	1.654	0.064
medical	120	45	978	1.245	0.028
enron	1001	53	1702	3.378	0.064
bibtex	1836	159	7395	2.402	0.015
CAL500	68	174	502	26.044	0.150
Corel5k	499	374	5000	3.522	0.009

4.1. Rethinking

In Section 3, we claim that RethinkNet is able to improve through iterations of rethinking. We justify our claim with this experiment. In this experiment, we use the simplest form of RNN, SRN, in the RNN layer of RethinkNet and the dimensionality of the RNN layer is fixed to 128. We set the number of rethink iterations $B = 5$ and plot the training and testing loss/score on Figure 3.

From the figure, we can see that for cost functions like Rank loss, F1 score, Accuracy score, which relies more on utilizing label correlation, achieved significant improvement over the increase of rethink iteration. Hamming loss is a criterion that evaluates each label independently and algorithms that does not consider label correlation like BR perform well on such criterion (Read et al., 2009). The result shows that the performance generally converges at around the third rethink iteration. For efficiency, the rest of experiments will be fixed with $B = 3$.

---

1. https://github.com/yangarbiter/multilabel-learn

(a) scene

(b) yeast

(c) medical

(d) CAL500
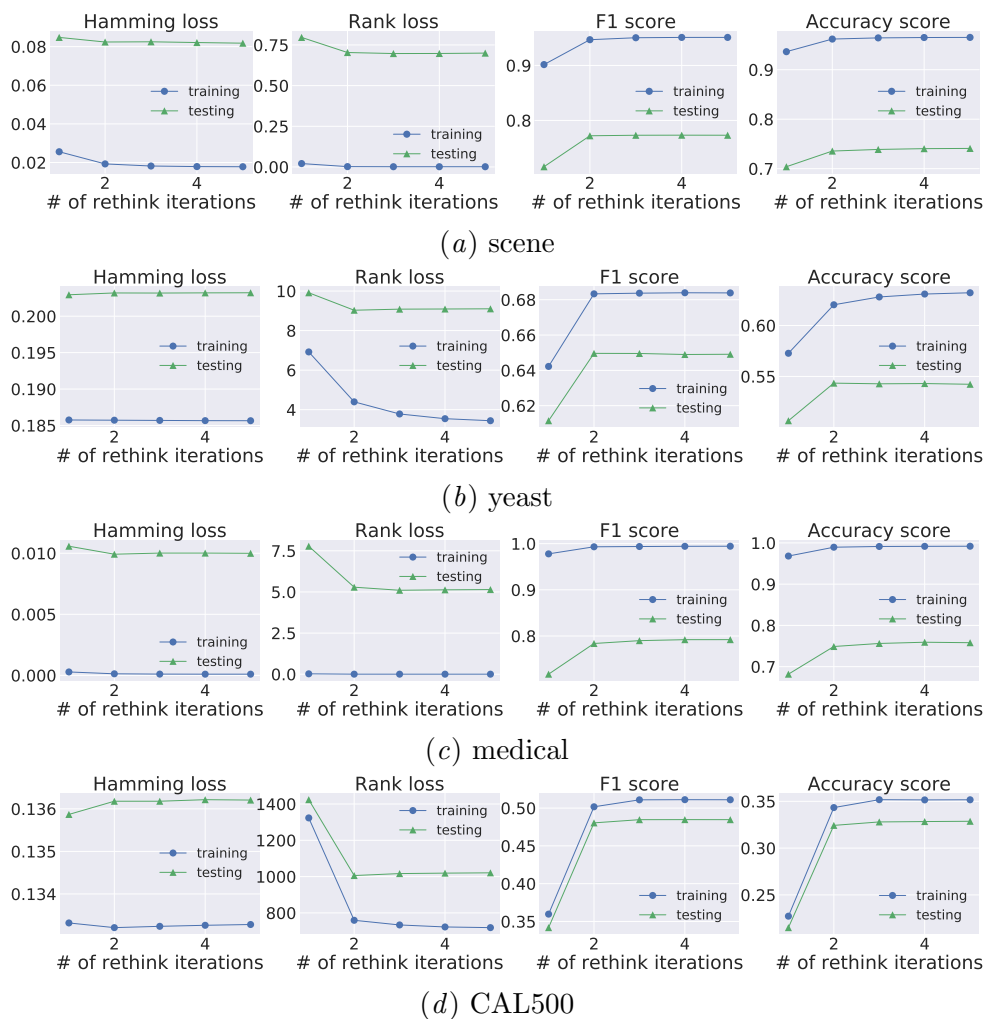
Figure 3: The average performance versus number of rethink iteration.

To further observe the rethinking process, we also trained RethinkNet on the MSCOCO (Lin et al., 2014) data set and observe its behavior on real-world images. The detailed experimental setup can be found in Section 4.4. Take Figure 4 as example. In the first iteration, RethinkNet predict label 'person', 'cup', 'fork', 'bowl', 'chair', 'dining table' exists in the figure. These are labels that are easier to detect. Using the knowledge this may be a scene on a dining table, the probability that there exist 'knife' or 'spoon' should be increased. In the second iteration, RethinkNet further predicted that 'bottle', 'knife', 'spoon' are also in the example. In the third iteration, RethinkNet found that the bottle should not be in the figure and exclude it from the prediction.

4.2. Effect of Reweighting

We conducted this experiment to verify the cost-sensitive reweighting can really use the cost information to reach a better performance. The performance of RethinkNet with and

Figure 4: An example from the MSCOCO data set with ground truth labels 'person', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'cake', 'chair', 'dining table'.

without reweighting under Rank loss, F1 score and Accuracy score is compared. Hamming loss is the same before and after reweighting so it is not shown in the result. Table 3 lists the mean and standard error (ste) of each experiment and it demonstrates that on almost all data sets, reweighting the loss function for RethinkNet yields better result.

Table 3: Experimental results (mean $\pm$ ste) of the performance in Rank loss ($\downarrow$), F1 score ($\uparrow$), Accuracy score ($\uparrow$) of non-reweighted and reweighted RethinkNet (best ones are bold)

	Rank loss		F1 score		Accuracy score	
data set	non-reweighted	reweighted	non-reweighted	reweighted	non-reweighted	reweighted
emotions	$3.48 \pm .13$	$\mathbf{1.82 \pm .25}$	$.652 \pm .012$	$\mathbf{.687 \pm .006}$	$.574 \pm .007$	$\mathbf{.588 \pm .007}$
scene	$2.50 \pm .03$	$\mathbf{.72 \pm .01}$	$.750 \pm .005$	$\mathbf{.772 \pm .005}$	$.721 \pm .008$	$\mathbf{.734 \pm .005}$
yeast	$13.3 \pm .05$	$\mathbf{9.04 \pm .09}$	$.612 \pm .005$	$\mathbf{.648 \pm .004}$	$.500 \pm .005$	$\mathbf{.538 \pm .004}$
birds	$8.21 \pm .43$	$\mathbf{4.24 \pm .32}$	$\mathbf{.237 \pm .011}$	$.236 \pm .012$	$\mathbf{.195 \pm .013}$	$.193 \pm .008$
tmc2007	$9.59 \pm .32$	$\mathbf{5.37 \pm .02}$	$\mathbf{.754 \pm .004}$	$.748 \pm .009$	$\mathbf{.691 \pm .003}$	$.690 \pm .002$
Arts1	$19.6 \pm .05$	$\mathbf{13.0 \pm .2}$	$.351 \pm .005$	$\mathbf{.365 \pm .003}$	$.304 \pm .005$	$\mathbf{.315 \pm .004}$
medical	$27.2 \pm .2$	$\mathbf{5.6 \pm .2}$	$.793 \pm .006$	$\mathbf{.795 \pm .004}$	$\mathbf{.761 \pm .006}$	$.760 \pm .006$
enron	60.3 ± 2.5	$\mathbf{39.7 \pm .5}$	$.544 \pm .007$	$\mathbf{.604 \pm .007}$	$.436 \pm .006$	$\mathbf{.480 \pm .004}$
Corel5k	$654. \pm 1.$	$\mathbf{524. \pm 2.}$	$.169 \pm .002$	$\mathbf{.257 \pm .001}$	$.118 \pm .001$	$\mathbf{.164 \pm .002}$
CAL500	$1545. \pm 17.$	$\mathbf{997. \pm 12.}$	$.363 \pm .003$	$\mathbf{.484 \pm .003}$	$.231 \pm .005$	$\mathbf{.328 \pm .002}$
bibtex	$186. \pm 1.$	$\mathbf{115. \pm 1.}$	$.390 \pm .002$	$\mathbf{.398 \pm .002}$	$.320 \pm .002$	$\mathbf{.329 \pm .002}$

4.3. Compare with Other MLC Algorithms

We compare RethinkNet with other state-of-the-art MLC and CSMLC algorithms in this experiment. The competing algorithms includes the binary relevance (BR), probabilistic classifier chain (PCC), classifier chain (CC), dynamic programming based classifier chain (CC-DP), condensed filter tree (CFT). To compare with the RNN structure used in CNN-RNN (Wang et al., 2016), we implemented a classifier chains using RNN (CC-RNN) as competitor. CC-RNN is CNN-RNN without the CNN layer since we are dealing with general data sets. BR is implemented using a feed-forward neural network with a 128 neurons hidden layer. We coupled both CC-RNN and RethinkNet with a 128 neurons LSTM. CC-RNN and BR are trained using same approach as RethinkNet. Training K independent feed-forward neural network is too computationally heavy, so we coupled CFT, PCC, CC

with L2-regularized logistic regression. CC-DP is coupled with SVM as it is derived for it. We adopt the implementation from SCIKIT-LEARN (Pedregosa et al., 2011) for both the L2-regularized logistic regression and linear SVM. The regularization strength for these models are searched within $(10^{-4}, 10^{-3}, \ldots, 10^4)$ with three-fold cross-validation. PCC does not have inference rule derived for Accuracy score and we use the F1 score inference rule as an alternative in view of the similarity in the formula. Other parameters not mentioned are kept with default of the implementation.

The experimental results are shown on Table 6 and the t-test results are on Table 4. Note that we cannot get the result of CC-DP in two weeks on the data sets Corel5k, CAL500 and bibtex so they are not listed. In terms of average ranking and t-test results, RethinkNet yields a superior performance. On Hamming loss, all algorithms are generally competitive. For Rank loss, F1 score and Accuracy score, CSMLC algorithms (RethinkNet, PCC, CFT) begin to take the lead. Even the parameters of cost-insensitive algorithms are tuned on the target evaluation criteria, they are not able to compete with cost-sensitive algorithms. This demonstrates the importance of developing cost-sensitive algorithms.

All three CSMLC algorithms has similar performance on Rank loss and RethinkNet performs slightly better on F1 score. For Accuracy score, since PCC is not able to directly utilize the cost information of, this makes PCC performs slightly poor.

When comparing with deep structures (RethinkNet, CC-RNN, BR), only BR is competitive under Hamming loss with RethinkNet. On all other settings, RethinkNet is able to outperform the other two competitors. CC-RNN learns an RNN with sequence length being the number of labels (K). When K gets large, the depth of CC-RNN can go very deep making it hard to train with fixed learning rate in our setting and failed to perform well on these data sets. This demonstrates that RethinkNet is a better designed deep structure to solve CSMLC problem.

Table 4: RethinkNet versus the competitors based on t-test at 95% confidence level

(#win/#tie/#loss)	PCC	CFT	CC-DP	CC	CC-RNN	BR
hamming ($\downarrow$)	6/1/4	3/4/4	5/2/1	6/1/4	8/3/0	3/6/2
rank loss ($\downarrow$)	5/1/5	5/2/4	7/1/0	10/1/0	10/1/0	10/1/0
f1 ($\uparrow$)	6/2/3	5/4/2	5/2/1	8/3/0	10/1/0	9/2/0
acc ($\uparrow$)	7/1/3	5/4/2	5/1/2	7/4/0	9/2/0	9/2/0
total	24/5/15	18/14/12	22/6/4	31/9/4	37/7/0	31/11/2

Table 5: Experimental results on MSCOCO data set.

	baseline	CNN-RNN	Att-RNN	RethinkNet
hamming ($\downarrow$)	0.0279	0.0267	0.0270	**0.0234**
rank loss ($\downarrow$)	60.4092	56.6088	43.5248	**35.2552**
f1 ($\uparrow$)	0.5374	0.5759	0.6309	**0.6622**
acc ($\uparrow$)	0.4469	0.4912	0.5248	**0.5724**

Table 6: Experimental results (mean ± ste) on different criteria (best results in bold)

			Hamming loss ↓			
data set	RethinkNet	PCC	CFT	CC-DP	CC	CC-RNN	BR
emotions	.191 ± .005[2]	.219 ± .005[6]	.194 ± .003[4]	.213 ± .004[5]	.219 ± .005[7]	.192 ± .004[3]	**.190 ± .004[1]**
scene	**.081 ± .001[1]**	.101 ± .001[5]	.095 ± .001[4]	.104 ± .002[7]	.101 ± .001[6]	.087 ± .001[2]	.087 ± .003[2]
yeast	**.205 ± .001[1]**	.218 ± .001[6]	**.205 ± .002[1]**	.214 ± .002[4]	.218 ± .001[7]	.215 ± .002[5]	.205 ± .002[2]
birds	**.048 ± .001[1]**	.050 ± .001[3]	.051 ± .001[6]	.050 ± .002[3]	.050 ± .001[3]	.053 ± .002[7]	.049 ± .001[2]
tmc2007	**.046 ± .000[1]**	.058 ± .000[5]	.057 ± .000[4]	.058 ± .000[5]	.058 ± .000[5]	.047 ± .001[2]	.048 ± .000[3]
Arts1	.062 ± .001[5]	.060 ± .000[2]	.060 ± .000[2]	.065 ± .001[6]	.060 ± .000[2]	.068 ± .001[7]	**.058 ± .000[1]**
medical	.010 ± .000[1]	.010 ± .000[1]	.011 ± .000[5]	**.010 ± .000[1]**	.010 ± .000[1]	.023 ± .000[7]	.011 ± .000[6]
enron	.047 ± .000[4]	.046 ± .000[1]	**.046 ± .000[1]**	.047 ± .000[4]	.046 ± .000[1]	.059 ± .000[7]	.048 ± .000[6]
Corel5k	.009 ± .000[1]	**.009 ± .000[1]**	.009 ± .000[1]	− ± −	.009 ± .000[1]	.009 ± .000[1]	.009 ± .000[1]
CAL500	**.137 ± .001[1]**	.138 ± .001[3]	.138 ± .001[3]	− ± −	.138 ± .001[3]	.149 ± .001[6]	.137 ± .001[1]
bibtex	.013 ± .000[2]	.013 ± .000[2]	.013 ± .000[2]	− ± −	.013 ± .000[2]	.015 ± .000[6]	**.012 ± .000[1]**
avg. rank	**1.82**	3.18	3.00	4.38	3.45	4.82	2.36

			Rank loss ↓			
data set	RethinkNet	PCC	CFT	CC-DP	CC	CC-RNN	BR
emotions	**1.48 ± .04[1]**	1.63 ± .05[3]	1.59 ± .03[2]	3.64 ± .02[4]	3.64 ± .02[4]	3.64 ± .02[4]	3.64 ± .02[4]
scene	**.72 ± .01[1]**	.88 ± .03[2]	.96 ± .04[3]	2.59 ± .01[5]	2.61 ± .01[6]	2.49 ± .04[4]	2.61 ± .01[6]
yeast	8.89 ± .11[2]	9.76 ± .08[3]	**8.83 ± .09[1]**	13.23 ± .04[5]	13.16 ± .07[4]	19.47 ± .04[7]	13.23 ± .04[5]
birds	**4.32 ± .27[1]**	4.66 ± .18[2]	4.90 ± .20[3]	8.51 ± .28[4]	8.51 ± .28[4]	8.51 ± .28[4]	8.51 ± .28[4]
tmc2007	5.22 ± .04[3]	4.32 ± .01[2]	**3.89 ± .01[1]**	12.32 ± .03[6]	12.14 ± .03[5]	21.44 ± .02[7]	11.39 ± .04[4]
Arts1	13.0 ± .1[3]	**12.2 ± .1[1]**	12.9 ± .0[2]	19.7 ± .0[4]	19.7 ± .0[4]	19.7 ± .0[4]	19.7 ± .0[4]
medical	5.3 ± .1[2]	**4.4 ± .1[1]**	6.0 ± .2[3]	27.2 ± .1[4]	27.3 ± .1[5]	27.3 ± .1[5]	27.3 ± .1[5]
enron	**40.1 ± .6[1]**	42.8 ± .6[3]	42.2 ± .6[2]	49.0 ± .5[5]	48.7 ± .5[4]	82.3 ± .5[7]	52.8 ± .4[6]
Corel5k	527. ± 2.[3]	**426. ± 1.[1]**	460. ± 2.[2]	− ± −	654. ± 1.[5]	653. ± 1.[4]	654. ± 1.[5]
CAL500	**1040. ± 8.[1]**	1389. ± 10.[3]	1234. ± 10.[2]	− ± −	1599. ± 13.[5]	1915. ± 10.[6]	1568. ± 9.[4]
bibtex	114. ± 1.[3]	**99. ± 1.[1]**	112. ± 1.[2]	− ± −	186. ± 1.[4]	186. ± 1.[4]	186. ± 1.[4]
avg. rank	**1.91**	2	2.09	4.63	4.55	5.09	4.64

			F1 score ↑			
data set	RethinkNet	PCC	CFT	CC-DP	CC	CC-RNN	BR
emotions	**.690 ± .007[1]**	.654 ± .004[3]	.655 ± .006[2]	.616 ± .008[7]	.620 ± .008[6]	.649 ± .007[4]	.639 ± .009[5]
scene	**.765 ± .003[1]**	.734 ± .004[3]	.730 ± .003[5]	.711 ± .005[6]	.710 ± .004[7]	.742 ± .004[2]	.731 ± .006[4]
yeast	**.651 ± .003[1]**	.598 ± .003[4]	.646 ± .003[2]	.617 ± .003[3]	.587 ± .003[6]	.577 ± .007[7]	.593 ± .012[5]
birds	.235 ± .016[2]	**.251 ± .011[1]**	.217 ± .009[5]	.208 ± .008[6]	.225 ± .008[3]	.087 ± .006[7]	.221 ± .008[4]
tmc2007	**.765 ± .002[1]**	.683 ± .001[6]	.718 ± .001[4]	.676 ± .001[7]	.684 ± .001[5]	.732 ± .009[3]	.740 ± .001[2]
Arts1	.385 ± .006[3]	**.425 ± .002[1]**	.411 ± .003[2]	.375 ± .003[4]	.365 ± .004[5]	.076 ± .002[7]	.359 ± .003[6]
medical	.790 ± .004[3]	**.812 ± .004[1]**	.780 ± .006[4]	.799 ± .004[2]	.778 ± .007[5]	.333 ± .010[7]	.755 ± .006[6]
enron	**.601 ± .003[1]**	.557 ± .002[3]	.599 ± .004[2]	.539 ± .004[6]	.556 ± .004[4]	.424 ± .011[7]	.548 ± .004[5]
Corel5k	.232 ± .003[3]	.233 ± .001[2]	**.259 ± .001[1]**	− ± −	.156 ± .002[5]	.000 ± .000[6]	.164 ± .001[4]
CAL500	**.485 ± .002[1]**	.405 ± .002[3]	.477 ± .002[2]	− ± −	.347 ± .003[5]	.048 ± .001[6]	.360 ± .004[4]
bibtex	.394 ± .005[2]	**.425 ± .002[1]**	.393 ± .003[3]	− ± −	.393 ± .002[4]	.000 ± .000[6]	.385 ± .003[5]
avg. rank	**1.73**	2.55	2.91	5.13	5.00	5.64	4.55

			Accuracy score ↑			
data set	RethinkNet	PCC	CFT	CC-DP	CC	CC-RNN	BR
emotions	**.600 ± .007[1]**	.556 ± .006[4]	.566 ± .006[3]	.534 ± .008[7]	.538 ± .008[6]	.568 ± .006[2]	.545 ± .009[5]
scene	**.737 ± .003[1]**	.693 ± .005[7]	.700 ± .004[4]	.699 ± .005[5]	.699 ± .004[6]	.718 ± .006[2]	.707 ± .008[3]
yeast	**.541 ± .004[1]**	.482 ± .003[7]	.533 ± .003[2]	.514 ± .003[3]	.486 ± .003[5]	.484 ± .008[6]	.495 ± .008[4]
birds	.205 ± .009[7]	.211 ± .009[6]	.592 ± .010[3]	**.596 ± .009[1]**	.592 ± .010[2]	.525 ± .013[5]	.589 ± .011[4]
tmc2007	**.700 ± .003[1]**	.578 ± .001[7]	.618 ± .001[4]	.587 ± .001[6]	.595 ± .001[5]	.634 ± .033[3]	.667 ± .002[2]
Arts1	.320 ± .003[5]	.351 ± .002[2]	**.370 ± .003[1]**	.337 ± .003[3]	.326 ± .003[4]	.071 ± .002[7]	.308 ± .002[6]
medical	.754 ± .004[3]	**.780 ± .004[1]**	.751 ± .006[4]	.771 ± .004[2]	.750 ± .008[5]	.304 ± .007[7]	.728 ± .008[6]
enron	.482 ± .003[1]	.429 ± .004[6]	.480 ± .004[2]	.437 ± .004[5]	.452 ± .004[3]	.324 ± .011[7]	.441 ± .004[4]
Corel5k	.161 ± .002[2]	.148 ± .001[3]	**.168 ± .001[1]**	− ± −	.111 ± .001[5]	.000 ± .000[6]	.113 ± .001[4]
CAL500	**.326 ± .001[1]**	.255 ± .001[3]	.320 ± .001[2]	− ± −	.218 ± .002[5]	.027 ± .002[6]	.230 ± .003[4]
bibtex	.327 ± .002[4]	**.353 ± .002[1]**	.328 ± .002[2]	− ± −	.327 ± .002[3]	.000 ± .000[6]	.326 ± .002[5]
avg. rank	**2.45**	4.27	2.55	4.00	4.45	5.18	4.27

4.4. Comparison on Image Data Set

The CNN-RNN and Att-RNN algorithms are designed to solve image labeling problems. The purpose of this experiment is to understand how RethinkNet performs on such task comparing with CNN-RNN and Att-RNN. We use the data set MSCOCO (Lin et al., 2014) and the training testing split provided by them. Pre-trained Resnet-50 (He et al., 2015) is adopted for feature extraction. The competing models include logistic regression as baseline, CNN-RNN, Att-RNN, and RethinkNet. The implementation of Att-RNN is from

the original author and other models are implemented with KERAS. The models are fine tuned with the pre-trained Resnet-50. The results on testing data set are shown on Table 5. The result shows that RethinkNet is able to outperform state-of-the-art deep learning models that are designed for image labeling.

4.5. Using Different Variations of RNN

In this experiment, we compare the performance of RethinkNet using different forms of RNN on the RNN layer in RethinkNet. The competitors includes SRN, LSTM, GRU, and IRNN. We tuned the label embedding dimensionality so that the total number of trainable parameters are around 200,000 for each form of RNN. The results are evaluated on two more commonly seen cost functions, Rank loss and F1 score, and shown on Table 7.

Different variations of RNN differs in the way they manipulate the memory. In terms of testing result, we can see that SRN and LSTM are two better choices. GRU and IRNN tends to overfit too much causing their testing performance to drop. Among SRN and LSTM, SRN tends to have a slightly larger discrepancy between training and testing performance. We can also observed that many data sets performs better with the same variation of RNN across cost functions. This indicates that different data set may require different form of memory manipulation.

Table 7: Experimental results with different RNN for RethinkNet. Evaluated in Rank loss ↓ and F1 score ↑ (best results are in bold)

Rank loss ↓								
data set	SRN		GRU		LSTM		IRNN	
	training	testing	training	testing	training	testing	training	testing
emotions	$1.06 \pm .51$	$1.81 \pm .26$	$.45 \pm .13$	$1.54 \pm .04$	$.68 \pm .06$	$\mathbf{1.50 \pm .06}$	$.00 \pm .00$	$1.60 \pm .04$
scene	$.001 \pm .000$	$\mathbf{.706 \pm .012}$	$.001 \pm .001$	$.708 \pm .015$	$.002 \pm .000$	$.715 \pm .006$	$.001 \pm .000$	$.763 \pm .005$
yeast	$.34 \pm .05$	$9.69 \pm .10$	$.70 \pm .06$	$9.93 \pm .16$	3.67 ± 1.09	$\mathbf{9.18 \pm .16}$	$.01 \pm .01$	$10.17 \pm .10$
birds	$.02 \pm .01$	$4.29 \pm .28$	$.00 \pm .00$	$4.44 \pm .31$	$.01 \pm .01$	$\mathbf{4.25 \pm .28}$	$.00 \pm .00$	$4.34 \pm .30$
tmc2007	$.11 \pm .03$	$\mathbf{5.01 \pm .07}$	$.12 \pm .03$	$5.25 \pm .04$	$.11 \pm .05$	$5.17 \pm .07$	$.07 \pm .01$	$5.13 \pm .05$
Arts1	$.7 \pm .1$	$13.3 \pm .2$	$5.8 \pm .2$	$13.1 \pm .2$	$5.5 \pm .0$	$\mathbf{13.0 \pm .1}$	$.2 \pm .0$	$14.3 \pm .2$
medical	$.00 \pm .00$	$\mathbf{4.75 \pm .22}$	$.00 \pm .00$	$5.85 \pm .27$	$.01 \pm .00$	$5.40 \pm .35$	$.00 \pm .00$	$6.13 \pm .42$
enron	$.4 \pm .0$	$39.7 \pm .4$	$.4 \pm .0$	$39.1 \pm .6$	$.4 \pm .0$	$\mathbf{38.8 \pm .5}$	$.4 \pm .0$	$39.0 \pm .5$
Corel5k	$0. \pm 0.$	$524. \pm 2.$	$0. \pm 0.$	$532. \pm 2.$	$0. \pm 0.$	$526. \pm 2.$	$0. \pm 0.$	$534. \pm 2.$
CAL500	$893. \pm 124.$	$1035. \pm 21.$	$377. \pm 11.$	$1101. \pm 13.$	$544. \pm 16.$	$1053. \pm 11.$	$8. \pm 1.$	$1358. \pm 14.$
bibtex	$0. \pm 0.$	$117. \pm 1.$	$0. \pm 0.$	$121. \pm 2.$	$0. \pm 0.$	$122. \pm 1.$	$0. \pm 0.$	$\mathbf{109. \pm 3.}$
F1 score ↑								
data set	SRN		GRU		LSTM		IRNN	
	training	testing	training	testing	training	testing	training	testing
emotions	$.794 \pm .023$	$.682 \pm .010$	$.811 \pm .022$	$.680 \pm .003$	$.788 \pm .004$	$\mathbf{.690 \pm .006}$	$.836 \pm .051$	$.681 \pm .008$
scene	$.919 \pm .002$	$\mathbf{.769 \pm .003}$	$.961 \pm .034$	$.757 \pm .003$	$.857 \pm .025$	$.753 \pm .011$	$.931 \pm .027$	$.764 \pm .004$
yeast	$.724 \pm .027$	$.641 \pm .005$	$.687 \pm .008$	$.643 \pm .005$	$.709 \pm .002$	$\mathbf{.651 \pm .003}$	$.691 \pm .022$	$.640 \pm .004$
birds	$.513 \pm .020$	$.235 \pm .014$	$.546 \pm .008$	$.243 \pm .015$	$.508 \pm .014$	$.240 \pm .013$	$.552 \pm .006$	$\mathbf{.248 \pm .015}$
tmc2007	$.990 \pm .002$	$\mathbf{.771 \pm .003}$	$.991 \pm .002$	$.764 \pm .003$	$.982 \pm .004$	$.758 \pm .004$	$.983 \pm .003$	$.757 \pm .001$
Arts1	$.763 \pm .009$	$\mathbf{.364 \pm .005}$	$.395 \pm .026$	$.323 \pm .003$	$.406 \pm .033$	$.320 \pm .004$	$.522 \pm .090$	$.344 \pm .009$
medical	$.995 \pm .001$	$\mathbf{.793 \pm .005}$	$.988 \pm .000$	$.792 \pm .002$	$.976 \pm .001$	$.791 \pm .006$	$.999 \pm .000$	$.786 \pm .009$
enron	$.689 \pm .004$	$.605 \pm .003$	$.695 \pm .003$	$\mathbf{.610 \pm .003}$	$.665 \pm .003$	$.603 \pm .003$	$.740 \pm .008$	$.608 \pm .007$
Corel5k	$.340 \pm .002$	$\mathbf{.260 \pm .002}$	$.325 \pm .002$	$.255 \pm .003$	$.475 \pm .018$	$.230 \pm .005$	$.409 \pm .016$	$.221 \pm .009$
CAL500	$.491 \pm .006$	$.474 \pm .004$	$.507 \pm .002$	$.483 \pm .004$	$.506 \pm .001$	$\mathbf{.485 \pm .002}$	$.493 \pm .002$	$.478 \pm .001$
bibtex	$.995 \pm .001$	$.391 \pm .005$	$.860 \pm .050$	$.385 \pm .004$	$.854 \pm .006$	$.379 \pm .003$	$.928 \pm .022$	$\mathbf{.399 \pm .003}$

5. Conclusion

Classic multi-label classification (MLC) algorithms predict labels as a sequence to model the label correlation. However, these approaches face the problem of ordering the labels

in the sequence. In this paper, we reformulate the sequence prediction problem to avoid the issue. By mimicking the human rethinking process, we propose a novel cost-sensitive multi-label classification (CSMLC) algorithm called RethinkNet. RethinkNet takes the process of gradually polishing its prediction as the sequence to predict. We adopt the recurrent neural network (RNN) to predict the sequence, and the memory in the RNN can then be used to store the label correlation information. In addition, we also modified the loss function to take in the cost information, and thus make RethinkNet cost-sensitive. Extensive experiments demonstrate that RethinkNet is able to outperform other MLC and CSMLC algorithms on general data sets. On image data set, RethinkNet is also able to exceed state-of-the-art image labeling algorithms in performance. The results suggest that RethinkNet is a promising algorithm for solving CSMLC using neural network.

References

Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

Moshe Bar. Visual objects in context. *Nature reviews. Neuroscience*, 5(8):617, 2004.

Shang-Fu Chen, Yi-Chen Chen, Chih-Kuan Yeh, and Yu-Chiang Frank Wang. Order-free RNN with visual attention for multi-label classification. *arXiv preprint arXiv:1707.05495*, 2017.

Weiwei Cheng, Eyke Hüllermeier, and Krzysztof J Dembczynski. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML*, 2010.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Krzysztof Dembczynski, Wojciech Kotlowski, and Eyke Hüllermeier. Consistent multilabel ranking through univariate losses. In *ICML*, 2012.

Krzysztof J Dembczynski, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. An exact algorithm for f-measure maximization. In *NIPS*, 2011.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Eduardo Corrêa Goncalves, Alexandre Plastino, and Alex A Freitas. A genetic algorithm for optimizing the label ordering in multi-label classifier chains. In *ICTAI*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. 2001.

Sheng-Jun Huang and Zhi-Hua Zhou. Multi-label learning by exploiting label correlations locally. In *AAAI*, 2012.

Michael I Jordan. Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495, 1997.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Chun-Liang Li and Hsuan-Tien Lin. Condensed filter tree for cost-sensitive multi-label classification. In *ICML*, 2014.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

Weiwei Liu and Ivor Tsang. On the optimality of classifier chain for multi-label classification. In *NIPS*, 2015.

F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Jesse Read and Jaakko Hollmén. A deep interpretation of classifier chains. In *IDA*, 2014.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. In *ECML-PKDD*, pages 254–269, 2009.

Jesse Read, Luca Martino, and David Luengo. Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, pages 1535–1546, 2014.

Konstantinos Trohidis, Grigorios Tsoumakas, George Kalliris, and Ioannis P. Vlahavas. Multi-label classification of music into emotions. In *ISMIR*, 2008.

Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *Data mining and knowledge discovery handbook*, pages 667–685. 2009.

Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Jozef Vilcek, and Ioannis Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011.

Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. In *CVPR*, 2016.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.