Cost-Sensitive Deep Learning with Layer-Wise Cost Estimation

Yu-An Chung
Massachusetts Institute of Technology
Cambridge, MA, U.S.A.
andyyuan@mit.edu

Shao-Wen Yang
Amazon
Seattle, WA, U.S.A.
swyang@amazon.com

Hsuan-Tien Lin
National Taiwan University
Taipei, Taiwan
htlin@csie.ntu.edu.tw

***Abstract*—While deep neural networks have succeeded in several applications, such as image classification, object detection, and speech recognition, by reaching very high classification accuracies, it is important to note that many real-world applications demand varying costs for different types of misclassification errors, thus requiring cost-sensitive classification algorithms. Current models of deep neural networks for cost-sensitive classification are restricted to some specific network structures and limited depth. In this paper, we propose a novel framework that can be applied to deep neural networks with any structure to facilitate their learning of meaningful representations for cost-sensitive classification problems. Furthermore, the framework allows end-to-end training of deeper networks directly. The framework is designed by augmenting auxiliary neurons to the output of each hidden layer for layer-wise cost estimation, and including the total estimation loss within the optimization objective. Experimental results on public benchmark data sets with two cost information settings demonstrate that the proposed framework outperforms state-of-the-art cost-sensitive deep learning models.**

***Index Terms*—cost-sensitive classification, deep neural networks, cost-sensitive deep learning**

I. Introduction

Deep learning has shown great success on a broad range of applications such as image classification [1, 2, 3, 4] and speech recognition [5]. Problems in such applications belong to a large class of regular classification in which each type of misclassification error is penalized equally.

Nevertheless, using accuracy as the evaluation metric for learning does not always produce the most useful classification system in the real world. In fact, many real-world applications [6, 7, 8, 9, 10] demand varying costs for different types of misclassification errors. For example, different costs are useful for building a realistic face recognition system [9, 11, 12, 13], in which a government staff being misrecognized as an impostor causes only a slight inconvenience; however, an imposer misrecognized as a staff can result in serious damage. Even in a simple digit recognition task, varying costs can be helpful in representing the nature of the task, as it is common and understandable to classify an ill-written 7 as 1 but classifying a 7 as a 4 would be laughable. Such applications call for cost-sensitive learning, which aims to identify the best classifier under the application-demanded costs.

Much research effort has been made to study cost-sensitive classification algorithms. In the works of [14, 15, 16], the researchers proposed to equip probabilistic classifiers with Bayes decision theory to enable the classifiers to consider the cost information during prediction. Some other studies extended existing cost-insensitive classification algorithms to be cost-sensitive, such as support vector machine [17]. Recently, as deep neural networks (DNN) have become state-of-the-art on a broad range of machine learning applications [5, 3, 4], researchers are attempting to make DNN cost-sensitive [18].

One successful DNN for cost-sensitive classification, called Cost-Sensitive DNN (CSDNN), has been recently proposed by [18]. The training process of CSDNN consists of two steps. The first step is to initialize the DNN by layer-wise pretraining using a cost-sensitive variant of the conventional auto-encoder [19]. The second step involves the fine-tuning of the DNN with a loss function that incorporates the cost information. The final CSDNN is thus cost-sensitive in both pretraining and training stages, and is shown to be a state-of-the-art algorithm that outperforms other existing cost-sensitive classification algorithms and some deep learning alternatives.

While CSDNN is state-of-the-art, its design is based on the conventional fully-connected DNN with sigmoid activation functions and experiences two issues. First, the design restricts the applicability to more modern structures such as convolutional [20, 1] and pooling layers. Second, the sigmoid function suffers from the problem of diminishing gradients when the network deepens, even after careful pretraining.

In this paper, we resolve these issues by proposing a novel framework for cost-sensitive deep learning. To build a cost-sensitive DNN for a K-class cost-sensitive classification problem, the proposed framework replaces the layer-wise pretraining step with layer-wise cost estimation, in which K additional neurons are added to the output of each hidden layer. These K additional neurons serve as auxiliary units that help the DNN learn meaningful representations towards estimating the costs in each layer. The DNN is then trained by solving a joint optimization problem on the weighted sum of the loss functions associated with the auxiliary units. Experiments conducted on four benchmark data sets and two cost information settings validate that the proposed framework outperforms CSDNN. The proposed framework can be easily and effectively attached to deep neural networks with ReLU [21] activation functions or convolutional neural networks like AlexNet [1], as shown in the longer version of this work [22]. The benefits of performance and generality make the proposed framework a favorable choice in practice.

The idea of using additional neurons as auxiliary units has been studied by several existing deep learning works, such as the well-known GoogLeNet [3], which takes the additional neurons as intermediate classifiers in selected hidden layers as regularizers. Deeply-Supervised Nets [23] adds additional neurons as intermediate classifiers to all hidden layers and reported that the nodes not only serve as regularizers but also allow improved convergence behavior. BranchyNet [24] considers auxiliary neurons at hidden layers as early exit points of prediction to speed up testing time. Nevertheless, all the previous works focus on using regular (cost-insensitive) classifiers as auxiliary units. To the best of our knowledge, our proposed framework is the first work that tackles cost-sensitive deep learning with layer-wise auxiliary units.

The rest of the paper is organized as follows. In Section II, we formally define the cost-sensitive classification problem and introduce related works. Then, we propose our framework in Section III, and validate the framework with real-world data sets in Section IV. Finally, we conclude in Section V.

II. Preliminary

We start by formalizing the cost-sensitive problem in Section II-A. We then introduce some important cost-sensitive deep learning works in Section II-B.

A. Cost-Sensitive Classification

In a K-class regular classification problem, a size-N training set $S = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$ is given, where each input vector $\mathbf{x}_n$ is within an input space $\mathcal{X} \subseteq \mathbb{R}^D$, and each label y_n is within a label space $\mathcal{Y} = \{1, 2, ..., K\}$. Regular classification aims at using S to train a classifier $g \colon \mathcal{X} \to \mathcal{Y}$ such that the expected error $[\![y \neq g(\mathbf{x})]\!]$ on the test examples $(\mathbf{x}, y)$ is small.[1] That is, each type of misclassification error is charged with the same penalty.

We consider a general cost-vector setting [14, 17] of cost-sensitive classification when designing the proposed framework. The cost-vector setting represents the cost information by coupling an additional cost vector $\mathbf{c} \in [0, \infty)^K$ with each example $(\mathbf{x}, y)$, where the k-th component $\mathbf{c}[k]$ of the cost vector $\mathbf{c}$ denotes the cost of predicting $\mathbf{x}$ as class k, and naturally $\mathbf{c}[y] = 0$. Consider a cost-sensitive training set $S_c = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^{N}$, cost-sensitive classification aims at using S_c to train a classifier $g_c \colon \mathcal{X} \to \mathcal{Y}$ such that the expected cost $\mathbf{c}[g_c(\mathbf{x})]$ on the test examples $(\mathbf{x}, y, \mathbf{c})$ is small.

A special case of the cost-vector setting is the cost-matrix setting, where the cost information is encoded by a $K \times K$ cost matrix $\mathbf{C}$ and each entry $\mathbf{C}(y, k) \in [0, \infty)$ indicates the cost for predicting a class-y example as class k. The information within the cost matrix can be simply cast as the cost vectors by defining the cost vector in $(\mathbf{x}, y, \mathbf{c})$ as the y-th row of the cost matrix $\mathbf{C}$. The cost-matrix setting, albeit less general, allows real-world applications to specify their demanded costs more easily. We follow many earlier cost-sensitive classification works [14, 15, 25, 17, 18] to take the cost-matrix setting when conducting benchmark experiments.

B. Deep Learning for Cost-Sensitive Classification

Nowadays, most DNNs are designed to solve regular classification problems [3, 4]. Those DNNs usually consist of several hidden layers with a softmax layer of K neurons at the end. Each input vector $\mathbf{x}$ propagates through different hidden layers and is transformed into different levels of latent representations. The softmax layer converts the last latent representation into per-class probability estimation, and takes the class with the highest estimated probability as the prediction $g(\mathbf{x})$ of the network.

On the other hand, only few works have explored cost-sensitive classification using shallow or deep neural networks. Prior to the prevailing of deep learning, [26] pioneered the study of making neural networks cost-sensitive by sampling and threshold-moving to tackle the class imbalance problem; [14] proposed four approaches of modifying neural networks for cost-sensitivity. Recently, [18] attempted to make cost-sensitive neural networks deeper by proposing a cost-sensitive deep learning algorithm called Cost-Sensitive DNN (CSDNN). In terms of the network structure, CSDNN starts with a DNN with fully-connected layers, but replaces the softmax layer at the end of the DNN by a cost-estimation layer. Each of the K neurons in the cost-estimation layer provides per-class cost estimation with regression instead of per-class probability estimation. Then, the class with the lowest estimated cost can be naturally taken as the prediction $g_c(\mathbf{x})$ of the network. [18] proposed to train the structure with a cost-sensitive loss function L_{OSR} on the cost-estimation layer.[2]

[18] then found that the performance of the network can be further improved by careful pretraining, and proposed a Cost-Sensitive Auto-Encoder (CSAE) to pretrain the structure above in a layer-wise manner. CSAE operates similar to a conventional auto-encoder [19], which is a shallow neural network that maps any input $\mathbf{x}$ to a representation such that the output $\tilde{\mathbf{x}}$ is a close reconstruction of the original input. The reconstruction error is commonly measured by cross-entropy loss, denoted by L_{CE}. What makes CSAE different is that the shallow network is augmented with K additional output neurons for cost estimation. That is, CSAE attempts to not only reconstruct $\mathbf{x}$ but also digest the cost information by estimating the cost vector $\mathbf{c}$. The attempt is represented with a mixture loss $(1-\beta) \cdot L_{\text{CE}} + \beta \cdot L_{\text{OSR}}$ with a balancing coefficient $\beta \in [0, 1]$ on the output layer of CSAE. When $\beta = 0$, CSAE degrades to a conventional auto-encoder.

Figure 1 illustrates how CSAE is used to pretrain CSDNN. With the pretraining, each layer in CSDNN carries some ability to estimate the costs. That is, the pretraining makes the latent representations *cost-aware*. [18] reported that such initialization indeed allows CSDNN to converge to a better optima and to reach state-of-the-art performance.

[1]The boolean operation $[\![\cdot]\!]$ is 1 if the condition is true, and 0 otherwise.

[2]The term L_{OSR} stands for One-Sided Regression and roots from a cost-sensitive SVM work [17]. Details are omitted here for lack of space.

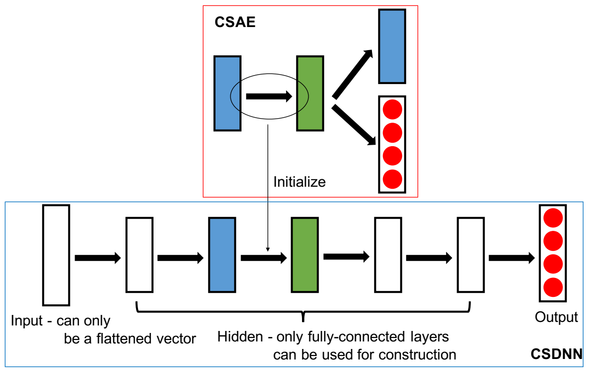
CSAE
Initialize
Input - can only be a flattened vector
Hidden - only fully-connected layers can be used for construction
Output
CSDNN

Fig. 1. CSAE pretraining for CSDNN [18]

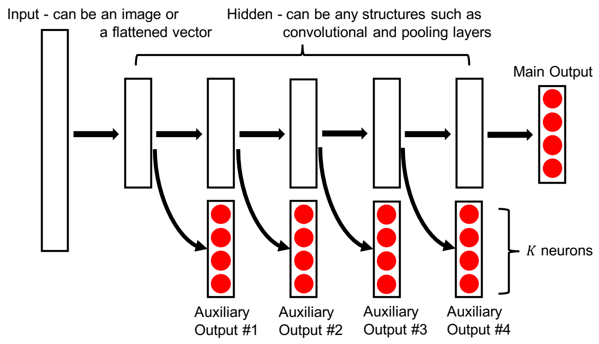
Input - can be an image or a flattened vector
Hidden - can be any structures such as convolutional and pooling layers
Main Output
K neurons
Auxiliary Output #1
Auxiliary Output #2
Auxiliary Output #3
Auxiliary Output #4

Fig. 2. a DNN with five hidden layers dressed with the proposed Auxiliary Cost-Sensitive Targets (AuxCST) framework

III. Proposed Framework

While CSDNN is state-of-the-art, its design is based on fully-connected layers with sigmoid activation functions and suffers from some issues. We discuss the issues behind CSDNN that motivate us to propose a better framework in Section III-A, and present the framework in Section III-B.

A. Motivation

Arguably the key idea within CSDNN is pretraining with CSAE. To understand the issues behind CSDNN, we first review the necessity of pretraining for general deep learning. In earlier years, neural networks used sigmoid or hyperbolic-tangent activation functions for non-linear transformation in the hidden layers [27, 28, 29, 19]. Both functions, which exhibit flatness in part of their curves, can cause the gradients of the network to be small. As the depth of the network increases, the small gradients in the latter layers of the network make the gradients in the earlier layers even smaller during back-propagation, a phenomenon known as the diminishing gradients. Earlier works by [29] and [19] tackled the diminishing-gradient problem by proposing a greedy layer-wise pretraining strategy. Pretraining helped mitigate the problem to some degree, but the problem would resurface as the network deepens if we stick with the same activation functions.

In recent years, another route to resolve the diminishing-gradient problem is to consider other activation functions, such as the rectifier linear unit (ReLU) [21]. As ReLU does not suffer from the diminishing-gradient problem as much as sigmoid or hyperbolic-tangent activation functions, pretraining is no longer necessary [30]. Nowadays, ReLU and many of its variants [31, 32] become the mainstream activation functions in modern deep learning studies [33, 4].

CSDNN [18] intended to conduct cost-sensitive deep learning by mimicking what [19] did for regular deep learning: using sigmoid activation functions, and adopting greedy layerwise pretraining. Thus, CSDNN carries the same problem of diminishing gradients when the network deepens, as our experimental results in Section IV will demonstrate. To keep cost-sensitive deep learning up to date with modern deep learning studies, it is then necessary to conduct cost-sensitive deep learning with other routes, such as adopting ReLU and removing the pretraining stage.

Nevertheless, directly removing the pretraining stage in CSDNN throws away one important benefit of CSAE in making the latent representations cost-aware. Next, we present our proposed framework to rescue the benefit.

B. Layer-Wise Cost Estimation

Our key goal is to construct a DNN that can simultaneously enjoy the benefit of cost-aware representation extraction (similar to that provided by CSAE), and the flexibility of using any structures. CSAE achieved cost-aware representation extraction by using K additional neurons in the auto-encoder for cost estimation. Our key idea is to also use K additional neurons for cost estimation, but instead of adding them to the auto-encoders that are separated from the DNN, we propose to directly put K neurons into each layer of the DNN. That is, we propose to replace CSAEs by "merging" their additional neurons with the DNN of our interest. The proposed structure is illustrated with Figure 2. By dressing the original DNN with K additional neurons in each layer that serve as auxiliary outputs, the extracted latent representations carry some ability to estimate the costs, thus achieving cost-aware representation extraction almost effortlessly.

As shown in Figure 2, in addition to augmenting K additional neurons to each layer of the DNN, we follow CSDNN and replace the output layer of the DNN with a cost-estimation layer. Then, the only remaining task is to train the "upgraded" DNN with a proper loss function. We consider a simple weighted-mixture loss function of the main one-sided regression loss function at the output layer, and the auxiliary one-sided regression loss functions at the hidden layers. In particular, let $L_{\text{OSR}}^{(i)}$ denote the auxiliary loss function for the output of the i-th hidden layer and $L_{\text{OSR}}^{(*)}$ denote the main loss function at the output layer, we train the upgraded DNN with:

$$\sum_{i=1}^{H-1} \alpha_i \cdot L_{\text{OSR}}^{(i)} + L_{\text{OSR}}^{(*)}, \tag{1}$$

where H is the number of hidden layers in the DNN, and α_i is the balancing coefficient for $L_{\mathrm{OSR}}^{(i)}$.[3]

With the proposed structural addons and the mixture loss function, we are now ready to present the full framework in Algorithm 1. The framework will be named as Auxiliary Cost-Sensitive Targets (AuxCST). While the novel framework appears simple, it carries many practical benefits. With the framework, we can now flexibly use ReLU or other activation functions and thus avoid diminishing-gradient problem. We can also build cost-sensitive DNN with any structures, such as image inputs with convolutional and pooling layers. Furthermore, we can apply this framework directly on any state-of-the-art DNN structures such as ResNet [4] for solving large-scale cost-sensitive classification problems.

Algorithm 1 Auxiliary Cost-Sensitive Targets (AuxCST)

```
Input: your favorite regular DNN or any off-the-shelf one [1, 3, 4] with $H$ hidden layers; balancing coefficients $\{\alpha_i\}_{i=1}^{H-1}$
1: Replace the softmax layer at the end of DNN with $K$ regression neurons and loss function $L_{\mathrm{OSR}}^{(*)}$
2: for $i = 1, 2, \ldots, H-1$ do
3:     Add $K$ additional regression neurons with loss function $L_{\mathrm{OSR}}^{(i)}$ to the output of the $i$-th hidden layer and connect them fully to the $i$-th hidden layer
4: end for
5: Train the new DNN by back-propagation on (1)
```

IV. Experiments

Three sets of experiments are conducted to validate the usefulness of the proposed AuxCST framework. Four benchmark data sets are used for the experiments: MNIST, CIFAR-10 [34], and CIFAR-100 [34]. For all data sets, the training and testing splits follow the source websites; the input vectors in training set are linearly scaled to $[0, 1]$, and the input vectors in the testing sets are scaled accordingly.

The data sets were originally collected for regular (cost-insensitive) classification and thus contain no cost information. We adopt the most frequently-used benchmark in cost-sensitive learning, the randomized proportional setup [25], to generate the costs. For a regular data set $S = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, the setup first generates a $K \times K$ matrix $\mathbf{C}$, and sets the diagonal entries $\mathbf{C}(y, y)$ to 0 while sampling the non-diagonal entries $\mathbf{C}(y, k)$ uniformly from $[0, 10\frac{|\{n:y_n=k\}|}{|\{n:y_n=y\}|}]$. Then, for each example $(\mathbf{x}_n, y_n)$ in S, its cost vector $\mathbf{c}_n$ is defined as the y_n-th row of matrix $\mathbf{C}$. The randomized proportional setup generates the cost information that takes the class distribution of the data set into account, charging a higher cost (in expectation) for misclassifying a minority class, and can thus be used to deal with imbalanced classification problems.

Arguably one of the most important use of cost-sensitive classification is to deal with imbalanced data sets. Nevertheless, the first three data sets MNIST, CIFAR-10, and CIFAR-100 are somewhat balanced, and the randomized proportional setup may generate similar cost for each type of misclassification error. To better meet the real-world usage scenario and increase the diversity of data sets, we further conduct experiments to evaluate the algorithms with imbalanced data sets. In particular, for each of the first three data sets MNIST, CIFAR-10, and CIFAR-100, we construct a variant data set by randomly picking 40% of the classes and removing 70% of the examples that belong to those 40% classes. We will name these imbalanced variants as $\mathrm{MNIST_{imb}}$, $\mathrm{CIFAR\text{-}10_{imb}}$, and $\mathrm{CIFAR\text{-}100_{imb}}$, respectively.

Our first experiment in Section IV-A intends to investigate the relationship between the balancing coefficient α_i in (1) for using AuxCST and the performance. Our second experiment in Section IV-B compares DNN equipped with AuxCST framework with state-of-the-art CSDNN [18]) to show the usefulness of AuxCST. For the first and the second experiments, the cost information was generated by the randomized proportional setup. In each of the experiments, we will describe the goal of the experiment, present the experimental results, and provide discussions and conclusions.

A. How does α_i affect AuxCST?

In our proposed Auxiliary Cost-Sensitive Targets (AuxCST) framework, K additional neurons are added in parallel to each of the hidden layer in DNN. As an example $\mathbf{x}$ propagates through the network, in addition to the final prediction layer, the DNN also outputs K values in each hidden layer. Same with the final prediction layer, these additional K neurons in each hidden layer also aim to estimate the per-class costs, and are coupled with L_{OSR}. The final objective function for optimizing the entire DNN is a weighted sum of the main one-sided loss for the final prediction layer and the auxiliary one-sided loss for all hidden layers, and has the form (1).

In this experiment, we would like to investigate the relationship between the selection of α_i in (1) and the performance (average test costs) of AuxCST framework. To simplify the experiment, we keep all coefficients α_i to identical values, that is, $\alpha_1 = \alpha_2 = ... = \alpha_{H-1} = \alpha$, and (1) becomes:

$$\alpha \cdot \sum_{i=1}^{H-1} L_{\mathrm{OSR}}^{(i)} + L_{\mathrm{OSR}}^{(*)}, \tag{2}$$

and we increase the value of α from 0 to 1 by a step 0.1. We show the results of the imbalanced version of MNIST here, while similar results have been observed for MNIST, CIFAR-10, CIFAR-100 and their imbalanced variants. Their cost information is generated by randomized proportional setup.

We constructed fully-connected DNN with varying numbers of hidden layers $H = \{1, 2, 3, 4, 5\}$, where each hidden layer consists of 1024 neurons. Note that our proposed AuxCST framework can be applied to DNN consists of any kind of layers, but since our goal in current experiment is not to pursue

[3]There is no need to consider $L_{\mathrm{OSR}}^{(H)}$ for the outputs of the last hidden layer, as the main loss function $L_{\mathrm{OSR}}^{(*)}$ readily conducts cost estimation.

the best performance but to investigate more about AuxCST, we choose to use only fully-connected layers for constructing DNN in order to reduce the amount of hyper-parameters.

The results are shown in Figure 3. We plot 5 curves (because we tested with 5 kinds of numbers of hidden layers), where the x-axis is the value of α, and the y-axis is the corresponding average test costs achieved. Note that when $\alpha = 0$, it means that the DNN does not make use of AuxCST framework. From the figure, no matter how many hidden layers there are, roughly U-shaped curves could be observed, and the lowest average test costs were achieved when α fell in the range $0.2 \sim 0.5$, implying that α within this range best balanced layer-wise and final cost estimation terms.

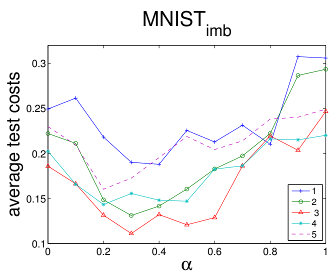

Fig. 3. The figure depicts the relationship between the selection of α for using AuxCST framework and the performance achieved on $\mathrm{MNIST_{imb}}$ with five curves. The numbers in the legend are the number of hidden layers, and each number corresponds to a curve.

B. Compare with state-of-the-art

In this experiment, we build two DNNs with and without AuxCST framework and compare them to state-of-the-art Cost-sensitive Deep Neural Network (CSDNN) [18]. We emphasize the two major drawbacks of CSDNN here:

1) CSDNN uses sigmoid functions for non-linear transformations, and this will eventually results in diminishing gradients when the network grows deeper.
2) CSDNN can be applied to DNN that consists of only fully-connected layers, this puts limits on its potential to be extended and applied to more challenging tasks that require modern neural components such as convolution and pooling layers.

To give CSDNN a fair chance of comparison, the two DNNs we build also consist of only fully-connected layers, and ReLU is used as activation function. The first DNN is equipped with AuxCST by setting $\alpha_i = 0.2$, as 0.2 was found to be one of the best value balancing for (2) in Section IV-A, we will refer to this DNN as AuxDNN. The second DNN, which will be referred to as NaiveDNN, did not make use of AuxCST and was directly optimized by L_{OSR}, it is equivalent to setting $\alpha = 0$ in (2).

The experimental results are displayed in Figure 4. The x-axis is the number of hidden layers and the y-axis is the corresponding average test costs achieved. As we can observe from Figure 4, when the number of hidden layers was less than or equal to three, CSDNN outperformed NaiveDNN probably because CSAE were doing cost-aware feature extraction relatively well, which accorded to the experimental results in [18]. When the number of hidden layers exceeded three, all of the three models began to suffer from overfitting, causing their average test costs to increase. However, by looking at CSDNN and NaiveDNN, it was interesting to observe that although the average test costs of both models increased, the extent of increment of CSDNN was larger than that of NaiveDNN. We inferred that this phenomenon was ascribed to the diminishing gradients caused by sigmoid functions used in CSDNN, and although CSAE had done their best to mitigate this problem when the network was relatively shallow, CSDNN can still not escape the fate of diminishing gradients when the network grew deeper. This phenomenon could not be observed in [18] because the deepest network they built had only three hidden layers. As for AuxDNN, it significantly outperformed both CSDNN and NaiveDNN regardless of the number of hidden layers, this further demonstrated the usefulness of our proposed AuxCST framework.

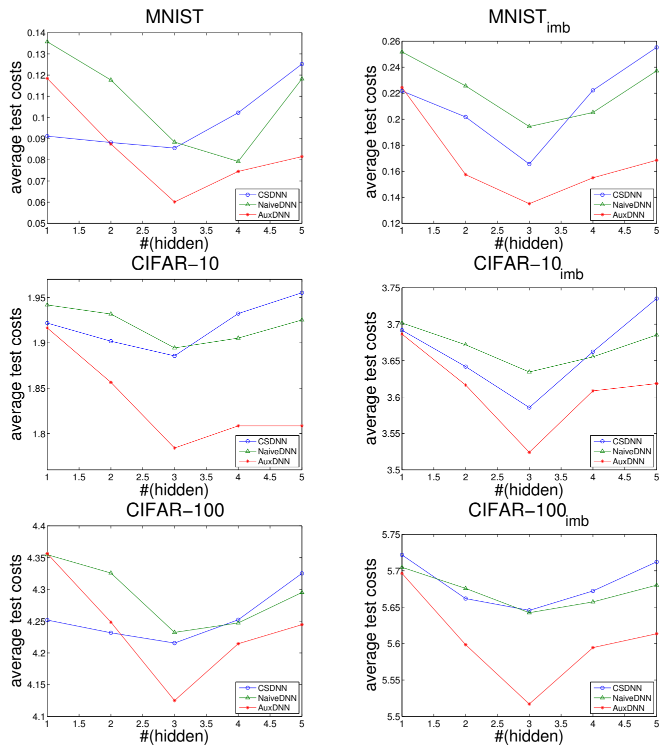

Fig. 4. The six sub-figures display the performance of the three competing DNNs on MNIST, CIFAR-10, CIFAR-100, $\mathrm{MNIST_{imb}}$, $\mathrm{CIFAR\text{-}10_{imb}}$, and $\mathrm{CIFAR\text{-}100_{imb}}$, where each curve corresponds to one competitor.

V. Conclusion and Future Work

We propose a novel framework Auxiliary Cost-Sensitive Targets (AuxCST) for general end-to-end cost-sensitive deep learning. Different from the previous approaches, the framework can be applied to DNN that consists of any structures to tackle challenging cost-sensitive classification problems. Extensive experimental results demonstrate the usefulness of the proposed framework for making any advanced DNN models cost-sensitive. In the future, we will build a deeper network with AuxCST framework to tackle ImageNet cost-sensitive classification problem.

Acknowledgement

We thank members of the NTU Computational Learning Laboratory and the anonymous reviewers for valuable comments. The work was partially supported by the Ministry of Science and Technology of Taiwan via MOST 107-2628-E-002-008-MY3 and 108-2119-M-007-010.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[2] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *CVPR*, 2012.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[5] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *TASLP*, vol. 20, no. 1, pp. 30–42, 2012.

[6] M. Tan, "Cost-sensitive learning of classification knowledge and its applications in robotics," *Machine Learning*, vol. 13, no. 1, pp. 7–33, 1993.

[7] P. K. Chan and S. J. Stolfo, "Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection," in *KDD*, 1998.

[8] W. Fan, W. Lee, S. Stolfo, and M. Miller, "A multiple model cost-sensitive approach for intrusion detection," in *ECML*, 2000.

[9] Y. Zhang and Z.-H. Zhou, "Cost-sensitive face recognition," *TPAMI*, vol. 32, no. 10, pp. 1758–1769, 2010.

[10] T.-K. Jan, H.-T. Lin, H.-P. Chen, T.-C. Chern, C.-Y. Huang, B.-C. Wen, C.-W. Chung, Y.-J. Li, Y.-C. Chuang, L.-L. Li, Y.-J. Chan, J.-K. Wang, Y.-L. Wang, C.-H. Lin, and D.-W. Wang, "Cost-sensitive classification on pathogen species of bacterial meningitis by Surface Enhanced Raman Scattering," in *BIBM*, 2011.

[11] J. Lu and Y.-P. Tan, "Cost-sensitive subspace learning for face recognition," in *CVPR*, 2010.

[12] L. Zhang, H. Li, X. Zhou, B. Huang, and L. Shang, "Cost-sensitive sequential three-way decision for face recognition," in *RSEISP*, 2014.

[13] G. Zhang, H. Sun, Z. Ji, Y.-H. Yuan, and Q. Sun, "Cost-sensitive dictionary learning for face recognition," *Pattern Recognition*, vol. 60, pp. 613–629, 2016.

[14] M. Kukar and I. Kononenko, "Cost-sensitive learning with neural networks," in *ECAI*, 1998.

[15] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive," in *KDD*, 1999.

[16] B. Zadrozny and C. Elkan, "Learning and making decisions when costs and probabilities are both unknown," in *KDD*, 2001.

[17] H.-H. Tu and H.-T. Lin, "One-sided support vector regression for multi-class cost-sensitive classification," in *ICML*, 2010.

[18] Y.-A. Chung, H.-T. Lin, and S.-W. Yang, "Cost-aware pre-training for multiclass cost-sensitive deep learning," in *IJCAI*, 2016.

[19] Y. Bengio, "Learning deep architectures for ai," *Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[22] Y.-A. Chung, S.-W. Yang, and H.-T. Lin, "Cost-sensitive deep learning with layer-wise cost estimation," NTU, Tech. Rep., 2016, https://arxiv.org/abs/1611.05134.

[23] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *AISTATS*, 2015.

[24] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *ICPR*, 2016.

[25] N. Abe, B. Zadrozny, and J. Langford, "An iterative method for multi-class cost-sensitive learning," in *KDD*, 2004.

[26] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *TKDE*, vol. 18, no. 1, pp. 63–77, 2006.

[27] J. Dayhoff, *Neural Network Architectures: An Introduction*. Van Nostrand Reinhold Co., 1990.

[28] S. Lawrence, L. Giles, A. C. Tsoi, and A. Back, "Face recognition: A convolutional neural-network approach," *TNN*, vol. 8, no. 1, pp. 98–113, 1997.

[29] G. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[30] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *AISTATS*, 2011.

[31] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.

[33] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML, Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[34] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.