Cost-aware Pre-training for Multiclass Cost-sensitive Deep Learning

Yu-An Chung
Department of CSIE
National Taiwan University
b01902040@ntu.edu.tw

Hsuan-Tien Lin
Department of CSIE
National Taiwan University
htlin@csie.ntu.edu.tw

Shao-Wen Yang
Intel Labs
Intel Corporation
shao-wen.yang@intel.com

Abstract

Deep learning has been one of the most prominent machine learning techniques nowadays, being the state-of-the-art on a broad range of applications where automatic feature extraction is needed. Many such applications also demand varying costs for different types of mis-classification errors, but it is not clear whether or how such cost information can be incorporated into deep learning to improve performance. In this work, we first design a novel loss function that embeds the cost information for the training stage of cost-sensitive deep learning. We then show that the loss function can also be integrated into the pre-training stage to conduct cost-aware feature extraction more effectively. Extensive experimental results justify the validity of the novel loss function for making existing deep learning models cost-sensitive, and demonstrate that our proposed model with cost-aware pre-training and training outperforms non-deep models and other deep models that digest the cost information in other stages.

1 Introduction

In many real-world machine learning applications [Tan, 1993; Chan and Stolfo, 1998; Fan *et al.*, 2000; Zhang and Zhou, 2010; Jan *et al.*, 2011], classification errors may come with different costs; namely, some types of mis-classification errors may be (much) worse than others. For instance, when classifying bacteria [Jan *et al.*, 2011], the cost of classifying a Gram-positive species as a Gram-negative one should be higher than the cost of classifying the species as another Gram-positive one because of the consequence on treatment effectiveness. Different costs are also useful for building a realistic face recognition system, where a government staff being mis-recognized as an impostor causes only little inconvenience, but an imposer mis-recognized as a staff can result in serious damage [Zhang and Zhou, 2010]. It is thus important to take into account the de facto *cost* of every type of error rather than only measuring the error rate and penalizing all types of errors equally.

The classification problem that mandates the learning algorithm to consider the cost information is called cost-sensitive classification. Amongst cost-sensitive classification algorithms, the binary classification ones [Elkan, 2001; Zadrozny *et al.*, 2003] are somewhat mature with re-weighting the training examples [Zadrozny *et al.*, 2003] being one major approach, while the multiclass classification ones are continuing to attract research attention [Domingos, 1999; Margineantu, 2001; Abe *et al.*, 2004; Tu and Lin, 2010].

This work focuses on multiclass cost-sensitive classification, whose algorithms can be grouped into three categories [Abe *et al.*, 2004]. The first category makes the prediction procedure cost-sensitive [Kukar and Kononenko, 1998; Domingos, 1999; Zadrozny and Elkan, 2001], generally done by equipping probabilistic classifiers with Bayes decision theory. The major drawback is that probability estimates can often be inaccurate, which in term makes cost-sensitive performance unsatisfactory. The second category makes the training procedure cost-sensitive, which is often done by transforming the training examples according to the cost information [Chan and Stolfo, 1998; Domingos, 1999; Zadrozny *et al.*, 2003; Beygelzimer *et al.*, 2005; Langford and Beygelzimer, 2005]. However, the transformation step cannot take the particularities of the underlying classification model into account and thus sometimes has room for improvement. The third category specifically extends one particular classification model to be cost-sensitive, such as support vector machine [Tu and Lin, 2010] or neural network [Kukar and Kononenko, 1998; Zhou and Liu, 2006]. Given that deep learning stands as an important class of models with its special properties to be discussed below, we aim to design cost-sensitive deep learning algorithms within the third category while borrowing ideas from other categories.

Deep learning models, or neural networks with deep architectures, are gaining increasing research attention in recent years. Training a deep neural network efficiently and effectively, however, comes with many challenges, and different models deal with the challenges differently. For instance, conventional fully-connected deep neural networks (DNN) generally initialize the network with an unsupervised pre-training stage before the actual training stage to avoid being trapped in a bad local minimal, and the unsupervised pre-training stage has been successfully carried out by stacked autoencoders [Vincent *et al.*, 2010; Krizhevsky and Hinton, 2011; Baldi, 2012]. Deep belief networks [Hinton *et al.*, 2006; Le Roux and Bengio, 2008] shape the network as a gener-

ative model and commonly take restricted Boltzmann machines [Le Roux and Bengio, 2008] for pre-training. Convolutional neural networks (CNN) [LeCun *et al.*, 1998] mimic the visual perception process of human based on special network structures that result in less need for pre-training, and are considered the most effective deep learning models in tasks like image or speech recognition [Ciresan *et al.*, 2011; Krizhevsky *et al.*, 2012; Abdel-Hamid *et al.*, 2014].

While some existing works have studied cost-sensitive neural networks [Kukar and Kononenko, 1998; Zhou and Liu, 2006], none of them have focused on cost-sensitive deep learning to the best of our knowledge. That is, we are the first to present cost-sensitive deep learning algorithms, with the hope of making deep learning more realistic for applications like bacteria classification and face recognition. In Section 2, we first formalize the cost-sensitive classification problem and review related deep learning works. Then, in Section 3, we start with a baseline algorithm that makes the prediction procedure cost-sensitive (first category). The features extracted from the training procedure of such an algorithm, however, are cost-blind. We then initiate a pioneering study on how the cost information can be digested in the training procedure (second category) of DNN and CNN. We design a novel loss function that matches the needs of neural network training while embedding the cost information. Furthermore, we argue that for DNN pre-trained with stacked autoencoders, the cost information should not only be used for the training stage, but also the *pre-training* stage. We then propose a novel pre-training approach for DNN (third category) that mixes unsupervised pre-training with a cost-aware loss function. Experimental results on deep learning benchmarks and standard cost-sensitive classification settings in Section 4 verified that the proposed algorithm based on cost-sensitive training and cost-aware pre-training indeed yields the best performance, outperforming non-deep models as well as a broad spectrum of deep models that are either cost-insensitive or cost-sensitive in other stages. Finally, we conclude in Section 5.

2 Background

We will formalize the multiclass cost-sensitive classification problem before introducing deep learning and related works.

2.1 Multiclass Cost-sensitive Classification

We first introduce the multiclass classification problem and then extend it to the cost-sensitive setting. The K-class classification problem comes with a size-N training set $S = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, where each input vector $\mathbf{x}_n$ is within an input space $\mathcal{X}$, and each label y_n is within a label space $\mathcal{Y} = \{1, 2, ..., K\}$. The goal of multiclass classification is to train a classifier $g\colon \mathcal{X} \to \mathcal{Y}$ such that the expected error $[\![y \neq g(\mathbf{x})]\!]$ on test examples $(\mathbf{x}, y)$ is small.[1]

Multiclass cost-sensitive classification extends multiclass classification by penalizing each type of mis-classification error differently based on some given costs. Specifically, consider a K by K cost matrix $\mathbf{C}$, where each entry $\mathbf{C}(y, k) \in [0, \infty)$ denotes the cost for predicting a class-y example as class k and naturally $\mathbf{C}(y, y) = 0$. The goal of cost-sensitive classification is to train a classifier g such that the expected cost $\mathbf{C}(y, g(\mathbf{x}))$ on test examples is small.

The cost-matrix setting is also called cost-sensitive classification with class-dependent costs. Another popular setting is to consider example-dependent costs, which means coupling an additional cost vector $\mathbf{c} \in [0, \infty)^K$ with each example $(\mathbf{x}, y)$, where the k-th component $\mathbf{c}[k]$ denotes the cost for classifying $\mathbf{x}$ as class k. During training, each $\mathbf{c}_n$ that accompanies $(\mathbf{x}_n, y_n)$ is also fed to the learning algorithm to train a classifier g such that the expected cost $\mathbf{c}[g(\mathbf{x})]$ is small with respect to the distribution that generates $(\mathbf{x}, y, \mathbf{c})$ tuples. The cost-matrix setting can be cast as a special case of the cost-vector setting by defining the cost vector in $(\mathbf{x}, y, \mathbf{c})$ as row y of the cost matrix $\mathbf{C}$. In this work, we will eventually propose a cost-sensitive deep learning algorithm that works under the more general cost-vector setting.

2.2 Neural Network and Deep Learning

There are many deep learning models that are successful for different applications nowadays [Lee *et al.*, 2009; Krizhevsky and Hinton, 2011; Ciresan *et al.*, 2011; Krizhevsky *et al.*, 2012; Simonyan and Zisserman, 2014]. In this work, we first study the fully-connected deep neural network (DNN) for multiclass classification as a starting point of making deep learning cost-sensitive. The DNN consists of H hidden layer and parameterizes each layer $i \in \{1, 2, ..., H\}$ by $\theta_i = \{\mathbf{W}_i, \mathbf{b}_i\}$, where $\mathbf{W}_i$ is a fully-connected weight matrix and $\mathbf{b}_i$ is a bias vector that enter the neurons. That is, the weight matrix and bias vector applied on the input are stored within $\theta_1 = \{\mathbf{W}_1, \mathbf{b}_1\}$. For an input feature vector $\mathbf{x}$, the H hidden layers of the DNN describe a complex feature transform function by computing $\phi(\mathbf{x}) = s(\mathbf{W}_H \cdot s(\cdots s(\mathbf{W}_2 \cdot s(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) \cdots) + \mathbf{b}_H)$, where $s(z) = \frac{1}{1+\exp(-z)}$ is the component-wise logistic function. Then, to perform multiclass classification, an extra softmax layer, parameterized by $\theta_{\text{sm}} = \{\mathbf{W}_{\text{sm}}, \mathbf{b}_{\text{sm}}\}$, is placed after the H-th hidden layer. There are K neurons in the softmax layer, where the j-th neuron comes with weights $\mathbf{W}_{\text{sm}}^{(j)}$ and bias $\mathbf{b}_{\text{sm}}^{(j)}$ and is responsible for estimating the probability of class j given $\mathbf{x}$:

$$P(y = j|\mathbf{x}) = \frac{\exp(\phi(\mathbf{x})^T \mathbf{W}_{\text{sm}}^{(j)} + \mathbf{b}_{\text{sm}}^{(j)})}{\sum_{k=1}^K \exp(\phi(\mathbf{x})^T \mathbf{W}_{\text{sm}}^{(k)} + \mathbf{b}_{\text{sm}}^{(k)})}. \quad (1)$$

Based on the probability estimates, the classifier trained from the DNN is naturally $g(\mathbf{x}) = \text{argmax}_{1 \leq k \leq K} P(y = k|\mathbf{x})$.

Traditionally, the parameters $\{\{\theta_i\}_{i=1}^H, \theta_{\text{sm}}\}$ of the DNN are optimized by the back-propagation algorithm, which is essentially gradient descent, with respect to the negative log-likelihood loss function over the training set S:

$$L_{\text{NLL}}(S) = \sum_{n=1}^{N} -\ln(P(y = y_n|\mathbf{x}_n)). \quad (2)$$

The strength of the DNN, through multiple layers of nonlinear transforms, is to extract sophisticated features automatically and implement complex functions. However, the training of the DNN is non-trivial because of non-convex optimization and gradient diffusion problems, which degrade the

[1] $[\![\cdot]\!]$ is 1 when the inner condition is true, and 0 otherwise.

test performance of the DNN when adding too many layers. [Hinton *et al.*, 2006] first proposed a greedy layer-wise pre-training approach to solve the problem. The layer-wise pre-training approach performs a series of feature extraction steps from the bottom (input layer) to the top (last hidden layer) to capture higher level representations of original features along the network propagation.

In this work, we shall improve a classical yet effective unsupervised pre-training strategy, *stacked denoising auto-encoders* [Vincent *et al.*, 2010], for the DNN. Denoising auto-encoder (DAE) is an extension of regular auto-encoder. An auto-encoder is essentially a (shallow) neural network with one hidden layer, and consists of two parameter sets: $\{\mathbf{W}, \mathbf{b}\}$ for mapping the (normalized) input vector $\mathbf{x} \in [0,1]^d$ to the d'-dimensional latent representation $\mathbf{h}$ by $\mathbf{h} = s(\mathbf{W}\cdot\mathbf{x}+\mathbf{b}) \in [0,1]^{d'}$; $\{\mathbf{W}', \mathbf{b}'\}$ for reconstructing an input vector $\tilde{\mathbf{x}}$ from $\mathbf{h}$ by $\tilde{\mathbf{x}} = s(\mathbf{W}' \cdot \mathbf{h} + \mathbf{b}')$. The auto-encoder is trained by minimizing the total cross-entropy loss $L_{CE}(S)$ over S, defined as

$$-\sum_{n=1}^{N}\sum_{j=1}^{d}\Big(\mathbf{x}_n[j]\ln\tilde{\mathbf{x}}_n[j] + (1-\mathbf{x}_n[j])\ln(1-\tilde{\mathbf{x}}_n[j])\Big), \quad (3)$$

where $\mathbf{x}_n[j]$ denotes the j-th component of $\mathbf{x}_n$ and $\tilde{\mathbf{x}}_n[j]$ is the corresponding reconstructed value.

The DAE extends the regular auto-encoder by randomly adding noise to inputs $\mathbf{x}_n$ before mapping to the latent representation, such as randomly setting some components of $\mathbf{x}_n$ to 0. Several DAEs can then be stacked to form a deep network, where each layer receives its input from the latent representation of the previous layer. For the DNN, initializing with stacked DAEs is known to perform better than initializing with stacked regular auto-encoders [Vincent *et al.*, 2010] or initializing randomly. Below we will refer the DNN initialized with stacked DAEs and trained (fine-tuned) by back-propagation with (2) as the SDAE, while restricting the DNN to mean the model that is initialized randomly and trained with (2).

In this work, we will also extend another popular deep learning model, the convolutional neural network (CNN), for cost-sensitive classification. The CNN is based on a locally-connected network structure that mimics the visual perception process [LeCun *et al.*, 1998]. We will consider a standard CNN structure specified in Caffe[2] [Jia *et al.*, 2014], which generally does not rely on a pre-training stage. Similar to the DNN, we consider the CNN with a softmax layer for multi-class classification.

2.3 Cost-sensitive Neural Network

Few existing works have studied cost-sensitive classification using neural networks [Kukar and Kononenko, 1998; Zhou and Liu, 2006]. [Zhou and Liu, 2006] focused on studying the effect of sampling and threshold-moving to tackle the class imbalance problem using neural network as a core classifier rather than proposing general cost-sensitive neural network algorithms. [Kukar and Kononenko, 1998] proposed four approaches of modifying neural networks for cost-sensitivity. The first two approaches train a usual multiclass classification neural network, and then make the prediction stage of the trained network cost-sensitive by including the costs in the prediction formula; the third approach modifies the learning rate of the training algorithm base on the costs; the fourth approach, called MIN (minimization of the misclassification costs), modifies the loss function of neural network training directly. Among the four proposed algorithms, MIN consistently achieves the lowest test cost [Kukar and Kononenko, 1998] and will be taken as one of our competitors. Nevertheless, none of the existing works, to the best of our knowledge, have conducted careful study on cost-sensitive algorithms for deep neural networks.

[2]https://github.com/BVLC/caffe/tree/master/examples/cifar10

3 Cost-sensitive Deep Learning

Before we start describing our proposed algorithm, we highlight a naïve algorithm. For the DNN/SDAE/CNN that estimate the probability with (1), when given the full picture of the cost matrix, a cost-sensitive prediction can be obtained using Bayes optimal decision, which computes the expected cost of classifying an input vector $\mathbf{x}$ to each class and predicts the label that reaches the lowest expected cost:

$$g(\mathbf{x}) = \operatorname*{argmin}_{1\leqslant k\leqslant K}\sum_{y=1}^{K} P(y|\mathbf{x})\mathbf{C}(y,k). \quad (4)$$

We will denote these algorithms as $\mathrm{DNN}_{\mathrm{Bayes}}$, $\mathrm{SDAE}_{\mathrm{Bayes}}$ and $\mathrm{CNN}_{\mathrm{Bayes}}$, respectively. These algorithms do not include the costs in the pre-training nor training stages. Also, those algorithms require knowing the full cost matrix, and cannot work under the cost-vector setting.

3.1 Cost-sensitive Training

The DNN essentially decomposes the multiclass classification problem to per-class probability estimation problems via the well-known one-versus-all (OVA) decomposition. [Tu and Lin, 2010] proposed the one-sided regression algorithm that extends OVA for support vector machine (SVM) to a cost-sensitive SVM by considering per-class regression problems. In particular, if regressors $r_k(\mathbf{x}) \approx \mathbf{c}[k]$ can be learned properly, a reasonable prediction can be made by

$$g_r(\mathbf{x}) \equiv \operatorname*{argmin}_{1\leqslant k\leqslant K} r_k(\mathbf{x}). \quad (5)$$

[Tu and Lin, 2010] further argued that the loss function of the regressor r_k with respect to $\mathbf{c}[k]$ should be one-sided. That is, $r_k(\mathbf{x})$ is allowed to underestimate the smallest cost $\mathbf{c}[y]$ and to overestimate other costs. Define $z_{n,k} = 2[\![\mathbf{c}_n[k] = \mathbf{c}_n[y_n]]\!] - 1$ for indicating whether $\mathbf{c}_n[k]$ is the smallest within $\mathbf{c}_n$. The cost-sensitive SVM [Tu and Lin, 2010] minimizes a regularized version of the total one-sided loss $\xi_{n,k} = \max(z_{n,k}\cdot(r_k(\mathbf{x}_n) - \mathbf{c}_n[k]), 0)$, where r_k are formed by (kernelized) linear models. With such a design, the cost-sensitive SVM enjoys the following property [Tu and Lin, 2010]:

$$\mathbf{c}_n[g_r(\mathbf{x}_n)] \leqslant \sum_{k=1}^{K}\xi_{n,k}. \quad (6)$$

That is, an upper bound $\sum_{k=1}^{K}\xi_{n,k}$ of the total cost paid by g_r on $\mathbf{x}_n$ is minimized within the cost-sensitive SVM.

If we replace the softmax layer of the DNN or the CNN with regression outputs (using the identity function instead of the logistic one for outputting), we can follow [Tu and Lin, 2010] to make DNN and CNN cost-sensitive by letting each output neuron estimate $\mathbf{c}[k]$ as r_k and predicting with (5). The training of the cost-sensitive DNN and CNN can also be done by minimizing the total one-sided loss. Nevertheless, the one-sided loss is not differentiable at some points, and back-propagation (gradient descent) cannot be directly applied. We thus derive a *smooth* approximation of $\xi_{n,k}$ instead. Note that the new loss function should not only approximate $\xi_{n,k}$ but also be an upper bound of $\xi_{n,k}$ to keep enjoying the bounding property of (6). [Lee and Mangasarian, 2001] has shown a smooth approximation $u+\frac{1}{\alpha}\cdot\ln(1+\exp(-\alpha u)) \approx \max(u,0)$ when deriving the smooth SVM. Taking $\alpha = 1$ leads to $\text{LHS} = \ln(1+\exp(u))$, which is trivially an upper bound of $\max(u,0)$ because $\ln(1+\exp(u)) > u$, and $\ln(1+\exp(u)) > \ln(1) = 0$. Based on the approximation, we define

$$\delta_{n,k} \equiv \ln(1+\exp(z_{n,k}\cdot(r_k(\mathbf{x}_n) - \mathbf{c}_n[k]))). \quad (7)$$

$\delta_{n,k}$ is not only a smooth approximation of $\xi_{n,k}$ that enjoys the differentiable property, but also an upper bound of $\xi_{n,k}$ to keep the bounding property of (6) held. That is, we can still ensure a small total cost by minimizing the newly defined smooth one-sided regression (SOSR) loss over all examples:

$$L_{\text{SOSR}}(S) = \sum_{n=1}^{N}\sum_{k=1}^{K}\delta_{n,k}. \quad (8)$$

We will refer to these algorithms, which replace the softmax layer of the DNN/SDAE/CNN with a regression layer parameterized by $\theta_{\text{SOSR}} = \{\mathbf{W}_{\text{SOSR}}, \mathbf{b}_{\text{SOSR}}\}$ and minimize (8) with back-propagation, as DNN_{SOSR}, $\text{SDAE}_{\text{SOSR}}$ and CNN_{SOSR}. These algorithms work with the cost-vector setting. They include costs in the training stage, but not the pre-training stage.

3.2 Cost-aware Pre-training

For multiclass classification, the pre-training stage, either in a totally unsupervised or partially supervised manner [Bengio *et al.*, 2007], has been shown to improve the performance of the DNN and several other deep models [Bengio *et al.*, 2007; Hinton *et al.*, 2006; Erhan *et al.*, 2010]. The reason is that pre-training usually helps initialize a neural network with better weights that prevent the network from getting stuck in poor local minima. In this section, we propose a cost-aware pre-training approach that leads to a novel cost-sensitive deep neural network (CSDNN) algorithm.

CSDNN is designed as an extension of $\text{SDAE}_{\text{SOSR}}$. Instead of pre-training with SDAE, CSDNN takes stacked *cost-sensitive auto-encoders* (CAE) for pre-training instead. For a given cost-sensitive example $(\mathbf{x}, y, \mathbf{c})$, CAE tries not only to denoise and reconstruct the original input $\mathbf{x}$ like DAE, but also to digest the cost information by *reconstructing the cost vector* $\mathbf{c}$. That is, in addition to $\{\mathbf{W}, \mathbf{b}\}$ and $\{\mathbf{W}', \mathbf{b}'\}$ for DAE, CAE introduces an extra parameter set $\{\mathbf{W}'', \mathbf{b}''\}$ fed to regression neurons from the hidden representation. Then, we can mix the two loss functions L_{CE} and L_{SOSR} with a balancing coefficient $\beta \in [0,1]$, yielding the following loss function for CAE over S:

$$L_{\text{CAE}}(S) = (1-\beta)\cdot L_{\text{CE}}(S) + \beta \cdot L_{\text{SOSR}}(S) \quad (9)$$

The mixture step is a widely-used technique for multi-criteria optimization [Hillermeier, 2001], where β controls the balance between reconstructing the original input $\mathbf{x}$ and the cost vector $\mathbf{c}$. A positive β makes CAE cost-aware during its feature extraction, while a zero β makes CAE degenerate to DAE. Similar to DAEs, CAEs can then be stacked to initialize a deep neural network before the weights are fine-tuned by back-propagation with (8). The resulting algorithm is named CSDNN, which is cost-sensitive in both the pre-training stage (by CAE) and the training stage (by (8)), and can work under the general cost-vector setting. The full algorithm is listed in Algorithm 1.

Algorithm 1 CSDNN

Input: Cost-sensitive training set $S = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^N$
1: **for** each hidden layer $\theta_i = \{\mathbf{W}_i, \mathbf{b}_i\}$ **do**
2: Learn a CAE by minimizing (9).
3: Take $\{\mathbf{W}_i, \mathbf{b}_i\}$ of CAE as θ_i.
4: **end for**
5: Fine-tune the network parameters $\{\{\theta_i\}_{i=1}^H, \theta_{\text{SOSR}}\}$ by minimizing (8) using back-propagation.
Output: The fine-tuned deep neural network with (5) as g_r.

CSDNN is essentially $\text{SDAE}_{\text{SOSR}}$ with DAEs replaced by CAEs with the hope of more effective cost-aware feature extraction. We can also consider $\text{SCAE}_{\text{Bayes}}$ which does the same for $\text{SDAE}_{\text{Bayes}}$. The CNN, due to its special network structure, generally does not rely on stacked DAEs for pre-training, and hence cannot be extended by stacked CAEs.

As discussed, DAE is a degenerate case of CAE. Another possible degeneration is to consider CAE with less complete cost information. For instance, a naïve cost vector defined by $\hat{\mathbf{c}}_n[k] = [\![y_n \neq k]\!]$ encodes the label information (whether the prediction is erroneous with respect to the demanded label) but not the complete cost information. To study whether it is necessary to take the complete cost information into account in CAE, we design two variant algorithms that replace the cost vectors in CAEs with $\hat{\mathbf{c}}_n[k]$, which effectively makes those CAEs *error-aware*. Then, $\text{SCAE}_{\text{Bayes}}$ becomes $\text{SEAE}_{\text{Bayes}}$ (with E standing for error); CSDNN becomes $\text{SEAE}_{\text{SOSR}}$.

4 Experiments

In the previous section, we have derived many cost-sensitive deep learning algorithms, each with its own specialty. They can be grouped into two series: those minimizing with (2) and predicting with (4) are Bayes-series algorithms ($\text{DNN}_{\text{Bayes}}$, $\text{SDAE}_{\text{Bayes}}$, $\text{SEAE}_{\text{Bayes}}$, $\text{SCAE}_{\text{Bayes}}$, and $\text{CNN}_{\text{Bayes}}$); those minimizing with (8) and predicting with (5) are SOSR-series algorithms (DNN_{SOSR}, $\text{SDAE}_{\text{SOSR}}$, $\text{SEAE}_{\text{SOSR}}$, $\text{CSDNN} \equiv \text{SCAE}_{\text{SOSR}}$, CNN_{SOSR}). Note that the Bayes-series can only be applied to the cost-matrix setting while the SOSR-series can deal with the cost-vector setting. The two

series help understand whether it is beneficial to consider the cost information in the training stage.

Within each series, CNN is based on a locally-connected structure, while DNN, SDAE, SEAE and SCAE are fully-connected and differ by how pre-training is conducted, ranging from none, unsupervised, error-aware, to cost-aware. The wide range helps understand the effectiveness of digesting the cost information in the pre-training stage.

Next, the two series will be compared with the blind-series algorithms ($\text{DNN}_{\text{blind}}$, $\text{SDAE}_{\text{blind}}$, and $\text{CNN}_{\text{blind}}$), which are the existing algorithms that do not incorporate the cost information at all, to understand the importance of taking the cost information into account. The two series will also be compared against two baseline algorithms: CSOSR [Tu and Lin, 2010], a non-deep algorithm that our proposed SOSR-series originates from; MIN [Kukar and Kononenko, 1998], a neural-network algorithm that is cost-sensitive in the training stage like the SOSR-series but with a different loss function. The algorithms along with highlights on where the cost information is digested are summarized in Table 1.

4.1 Setup

We conducted experiments on MNIST, bg-img-rot (the hardest variant of MNIST provided in [Larochelle *et al.*, 2007]), SVHN [Netzer *et al.*, 2011], and CIFAR-10 [Krizhevsky and Hinton, 2009]. The first three datasets belong to handwritten digit recognition and aim to classify each image into a digit of 0 to 9 correctly; CIFAR-10 is a well-known image recognition dataset which contains 10 classes such as car, ship and animal. For all four datasets, the training, validation, and testing split follows the source websites; the input vectors in the training set are linearly scaled to $[0, 1]$, and the input vectors in the validation and testing sets are scaled accordingly.

The four datasets are originally collected for multiclass classification and contain no cost information. We adopt the most frequently-used benchmark in cost-sensitive learning, the randomized proportional setup [Abe *et al.*, 2004], to generate the costs. The setup is for the cost-matrix setting. It 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 $[0, 10\frac{|\{n:y_n=k\}|}{|\{n:y_n=y\}|}]$. The randomized proportional setup generates the cost information that takes the class distribution of the dataset into account, charging a higher cost (in expectation) for mis-classifying a minority class, and can thus be used to deal with imbalanced classification problems. Note that we take this benchmark cost-matrix setting to give prediction-stage cost-sensitive algorithms like the Bayes-series a fair chance of comparison. We find that the range of the costs can affect the numerical stability of the algorithms, and hence scale all the costs by the maximum value within $\mathbf{C}$ during training in our implementation. The reported test results are based on the unscaled $\mathbf{C}$.

Arguably one of the most important use of cost-sensitive classification is to deal with imbalanced datasets. Nevertheless, the four datasets above are somewhat balanced, and the randomized proportional setup may generate similar cost for each type of mis-classification error. To better meet the real-world usage scenario, we further conducted experiments to evaluate the algorithms with imbalanced datasets. In particular, for each dataset, we construct a variant dataset by randomly picking four classes and removing 70% of the examples that belong to those four classes. We will name these imbalanced variants as $\text{MNIST}_{\text{imb}}$, $\text{bg-img-rot}_{\text{imb}}$, SVHN_{imb}, and $\text{CIFAR-10}_{\text{imb}}$, respectively.

All experiments were conducted using Theano. For algorithms related to DNN and SDAE, we selected the hyperparameters by following [Vincent *et al.*, 2010]. The β in (9), needed by SEAE and SCAE algorithms, was selected among $\{0, 0.05, 0.1, 0.25, 0.4, 0.75, 1\}$. As mentioned, for CNN, we considered a standard structure in Caffe [Jia *et al.*, 2014].

4.2 Experimental Results

The average test cost of each algorithm along with the standard error is shown in Table 2. The best result[3] per dataset among all algorithms is highlighted in bold.

Is it necessary to consider costs? $\text{DNN}_{\text{blind}}$ and $\text{SDAE}_{\text{blind}}$ performed the worst on almost all the datasets. While $\text{CNN}_{\text{blind}}$ was slightly better than those two, it never reached the best performance for any dataset. The results indicate the necessity of taking the cost information into account.

Is it necessary to go deep? The two existing cost-sensitive baselines, CSOSR and MIN, outperformed the cost-blind algorithms often, but were usually not competitive to cost-sensitive deep learning algorithms. The results validate the importance of studying cost-sensitive deep learning.

Is it necessary to incorporate costs during training? SOSR-series models, especially under the imbalanced scenario, generally outperformed their Bayes counterparts. The results demonstrate the usefulness of the proposed (7) and (8) and the importance of incorporating the cost information during the training stage.

Is it necessary to incorporate costs during pre-training? CSDNN outperformed both $\text{SEAE}_{\text{SOSR}}$ and $\text{SDAE}_{\text{SOSR}}$, and $\text{SDAE}_{\text{SOSR}}$ further outperformed DNN_{SOSR}. The results show that for the fully-connected structure where pre-training is needed, our newly proposed cost-aware pre-training with CAE is indeed helpful in making deep learning cost-sensitive.

Which is better, CNN_{SOSR} **or** CSDNN**?** The last two columns in Table 2 show that CSDNN is competitive to CNN_{SOSR}, with both algorithms usually achieving the best performance. CSDNN is slightly better on two datasets. Note that CNNs are known to be powerful for image recognition tasks, which match the datasets that we have used. Hence, it is not surprising that CNN can reach promising performance with our proposed SOSR loss (8). Our efforts not only make CNN cost-sensitive, but also result in the CSDNN algorithm that makes the full-connected deep neural network cost-sensitive with the help of cost-aware pre-training via CAE.

Is the mixture loss necessary? To have more insights on CAE, we also conducted experiments to evaluate the perfor-

[3]The selected CSDNN that achieved the test cost listed in Table 2 is composed of 3 hidden layers, and each hidden layer consists of 3000 neurons.

Table 1: cost-awareness of algorithms (O: cost-aware; E: error-aware; X: cost-blind)

Stage \ Algorithm	$\text{DNN}_{\text{blind}}$	$\text{SDAE}_{\text{blind}}$	$\text{CNN}_{\text{blind}}$	CSOSR	MIN	$\text{DNN}_{\text{Bayes}}$	$\text{SDAE}_{\text{Bayes}}$	$\text{SEAE}_{\text{Bayes}}$	$\text{SCAE}_{\text{Bayes}}$	$\text{CNN}_{\text{Bayes}}$	DNN_{SOSR}	$\text{SDAE}_{\text{SOSR}}$	$\text{SEAE}_{\text{SOSR}}$	CSDNN	CNN_{SOSR}
pre-training	none	X	none	none	none	none	X	E	O	none	none	X	E	O	none
training	X	X	X	O	O	X	X	X	X	X	O	O	O	O	O
prediction	X	X	X	X	X	O	O	O	O	O	X	X	X	X	X

Table 2: Average test cost

Dataset \ Algorithm	$\text{DNN}_{\text{blind}}$	$\text{SDAE}_{\text{blind}}$	$\text{CNN}_{\text{blind}}$	CSOSR	MIN	$\text{DNN}_{\text{Bayes}}$	$\text{SDAE}_{\text{Bayes}}$	$\text{SEAE}_{\text{Bayes}}$	$\text{SCAE}_{\text{Bayes}}$	$\text{CNN}_{\text{Bayes}}$	DNN_{SOSR}	$\text{SDAE}_{\text{SOSR}}$	$\text{SEAE}_{\text{SOSR}}$	CSDNN	CNN_{SOSR}
MNIST	0.11 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.00	0.10 ± 0.003	0.10 ± 0.00	$\mathbf{0.09 \pm 0.00}$	0.09 ± 0.00	0.09 ± 0.00	$\mathbf{0.09 \pm 0.00}$	0.10 ± 0.00	0.09 ± 0.00	0.09 ± 0.00	$\mathbf{0.09 \pm 0.00}$	$\mathbf{0.08 \pm 0.00}$
bg-img-rot	3.33 ± 0.06	3.28 ± 0.07	3.05 ± 0.07	3.25 ± 0.06	3.02 ± 0.06	2.95 ± 0.07	2.66 ± 0.07	2.85 ± 0.07	2.54 ± 0.07	2.40 ± 0.07	3.21 ± 0.07	2.99 ± 0.07	3.00 ± 0.07	$\mathbf{2.34 \pm 0.07}$	$\mathbf{2.29 \pm 0.07}$
SVHN	1.58 ± 0.03	1.40 ± 0.03	0.91 ± 0.03	1.17 ± 0.03	1.19 ± 0.03	1.07 ± 0.03	0.93 ± 0.03	0.94 ± 0.03	0.88 ± 0.03	$\mathbf{0.85 \pm 0.03}$	1.02 ± 0.03	0.92 ± 0.03	0.99 ± 0.03	$\mathbf{0.83 \pm 0.03}$	$\mathbf{0.82 \pm 0.03}$
CIFAR-10	3.46 ± 0.04	3.26 ± 0.05	2.51 ± 0.04	3.30 ± 0.04	3.19 ± 0.05	2.80 ± 0.05	2.52 ± 0.05	2.68 ± 0.05	2.38 ± 0.04	2.34 ± 0.05	2.74 ± 0.05	2.48 ± 0.04	2.52 ± 0.05	$\mathbf{2.24 \pm 0.05}$	$\mathbf{2.25 \pm 0.04}$
$\text{MNIST}_{\text{imb}}$	0.32 ± 0.01	0.31 ± 0.01	0.19 ± 0.01	0.26 ± 0.01	0.27 ± 0.01	0.23 ± 0.01	0.20 ± 0.01	0.20 ± 0.01	$\mathbf{0.18 \pm 0.01}$	$\mathbf{0.18 \pm 0.01}$	0.22 ± 0.01	0.20 ± 0.01	0.19 ± 0.01	$\mathbf{0.17 \pm 0.01}$	$\mathbf{0.17 \pm 0.01}$
$\text{bg-img-rot}_{\text{imb}}$	15.9 ± 0.70	13.8 ± 0.70	5.04 ± 0.67	8.55 ± 0.70	8.40 ± 0.69	7.19 ± 0.69	5.10 ± 0.70	4.95 ± 0.70	4.73 ± 0.70	$\mathbf{4.49 \pm 0.68}$	6.89 ± 0.70	4.99 ± 0.69	4.86 ± 0.69	$\mathbf{4.16 \pm 0.68}$	$\mathbf{4.39 \pm 0.69}$
SVHN_{imb}	1.79 ± 0.01	1.60 ± 0.01	0.31 ± 0.01	1.05 ± 0.01	0.99 ± 0.01	0.53 ± 0.01	0.33 ± 0.01	0.34 ± 0.01	0.29 ± 0.01	0.28 ± 0.01	0.51 ± 0.01	0.31 ± 0.01	0.31 ± 0.01	$\mathbf{0.26 \pm 0.01}$	0.28 ± 0.01
$\text{CIFAR-10}_{\text{imb}}$	19.1 ± 0.09	17.7 ± 0.09	7.29 ± 0.08	10.1 ± 0.09	11.2 ± 0.09	8.16 ± 0.09	7.48 ± 0.09	7.25 ± 0.08	6.97 ± 0.09	6.81 ± 0.09	7.86 ± 0.08	7.44 ± 0.09	7.14 ± 0.09	$\mathbf{6.48 \pm 0.09}$	6.63 ± 0.08

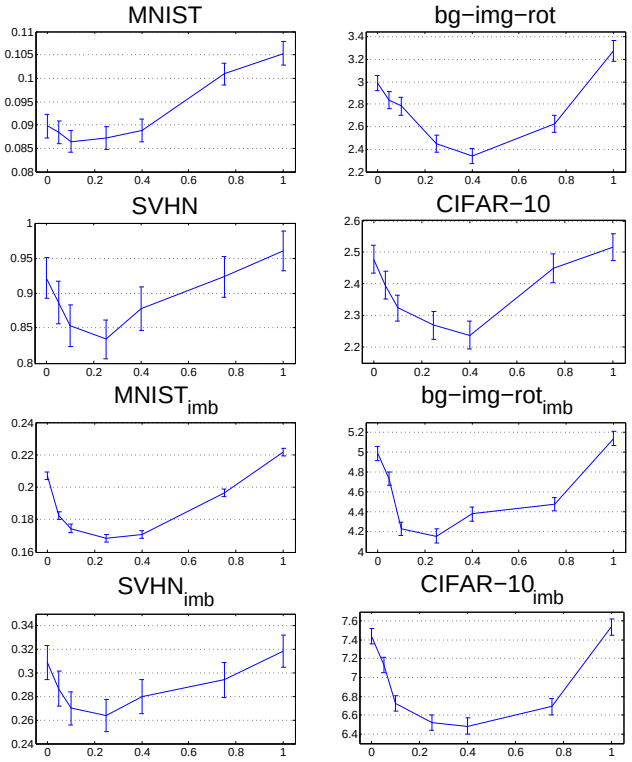

Figure 1: Relation between β and test cost (note that $\text{SDAE}_{\text{SOSR}}$ is the data point with $\beta = 0$).

mance of CSDNN for $\beta \in [0, 1]$. When $\beta = 0$, CSDNN degenerates to $\text{SDAE}_{\text{SOSR}}$; when $\beta = 1$, each CAE of CSDNN performs fully cost-aware pre-training to fit the cost vectors. The results are displayed in Figure 1, showing a roughly U-shaped curve. The curve implies that some $\beta \in [0, 1]$ that best balances the tradeoff between denoising and cost-awareness can be helpful. The results validate the usefulness of the proposed mixture loss (9) for pre-training.

5 Conclusion

We proposed a novel deep learning algorithm CSDNN for multiclass cost-sensitive classification with deep learning. Existing baselines and other alternatives within the blind-series, the Bayes-series and the SOSR-series were extensively compared with CSDNN carefully to validate the importance of each component of CSDNN. The experimental results demonstrate that incorporating the cost information into both the pre-training and the training stages leads to promising performance of CSDNN, outperforming those baselines and alternatives. One key component of CSDNN, namely the SOSR loss for cost-sensitivity in the training stage, is shown to be helpful in improving the performance of CNN. The results justify the importance of the proposed SOSR loss for training and the CAE approach for pre-training.

6 Acknowledgement

We thank the anonymous reviewers 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

[Abdel-Hamid *et al.*, 2014] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 22(10):1533–1545, 2014.

[Abe *et al.*, 2004] Naoki Abe, Bianca Zadrozny, and John Langford. An iterative method for multi-class cost-sensitive learning. In *KDD*, 2004.

[Baldi, 2012] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. *Unsupervised and Transfer Learning Challenges in Machine Learning*, 2012.

[Bengio *et al.*, 2007] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, 2007.

[Beygelzimer *et al.*, 2005] Alina Beygelzimer, Varsha Dani, Tom Hayes, John Langford, and Bianca Zadrozny. Error limiting reductions between classification tasks. In *ICML*, 2005.

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

[Ciresan *et al.*, 2011] Dan C Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI*, 2011.

[Domingos, 1999] Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *KDD*, 1999.

[Elkan, 2001] Charles Elkan. The foundations of cost-sensitive learning. In *IJCAI*, 2001.

[Erhan *et al.*, 2010] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *JMLR*, 11:625–660, 2010.

[Fan *et al.*, 2000] Wei Fan, Wenke Lee, Salvatore J. Stolfo, and Matthew Miller. A multiple model cost-sensitive approach for intrusion detection. In *ECML*, 2000.

[Hillermeier, 2001] Claus Hillermeier. *Nonlinear multiobjective optimization*. Birkhauser, 2001.

[Hinton *et al.*, 2006] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[Jan *et al.*, 2011] Te-Kang Jan, Hsuan-Tien Lin, Hsin-Pai Chen, Tsung-Chen Chern, Chung-Yueh Huang, Bing-Cheng Wen, Chia-Wen Chung, Yung-Jui Li, Ya-Ching Chuang, Li-Li Li, Yu-Jiun Chan, Juen-Kai Wang, Yuh-Lin Wang, Chi-Hung Lin, and Da-Wei Wang. Cost-sensitive classification on pathogen species of bacterial meningitis by Surface Enhanced Raman Scattering. In *BIBM*, 2011.

[Jia *et al.*, 2014] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

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

[Krizhevsky and Hinton, 2011] Alex Krizhevsky and Geoffrey E. Hinton. Using very deep autoencoders for content-based image retrieval. In *ESANN*, 2011.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[Kukar and Kononenko, 1998] Matjaz Kukar and Igor Kononenko. Cost-sensitive learning with neural networks. In *ECAI*, 1998.

[Langford and Beygelzimer, 2005] John Langford and Alina Beygelzimer. Sensitive error correcting output codes. In *COLT*, 2005.

[Larochelle *et al.*, 2007] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML*, 2007.

[Le Roux and Bengio, 2008] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural Computation*, 20:1631–1649, 2008.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

[Lee and Mangasarian, 2001] Yuh-Jye Lee and O. L. Mangasarian. SSVM: A smooth support vector machine. *Computational Optimization and Applications*, 20:5–22, 2001.

[Lee *et al.*, 2009] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, 2009.

[Margineantu, 2001] Dragos D. Margineantu. Methods for cost-sensitive learning. In *IJCAI*, 2001.

[Netzer *et al.*, 2011] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, 2011.

[Simonyan and Zisserman, 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Tan, 1993] Ming Tan. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, 13:7–33, 1993.

[Tu and Lin, 2010] Han-Hsing Tu and Hsuan-Tien Lin. One-sided support vector regression for multiclass cost-sensitive classification. In *ICML*, 2010.

[Vincent *et al.*, 2010] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, 11:3371–3408, 2010.

[Zadrozny and Elkan, 2001] Bianca Zadrozny and Charles Elkan. Learning and making decisions when costs and probabilities are both unknown. In *KDD*, 2001.

[Zadrozny *et al.*, 2003] Bianca Zadrozny, John Langford, and Naoki Abe. Cost-sensitive learning by cost-proportionate example weighting. In *ICDM*, 2003.

[Zhang and Zhou, 2010] Yin Zhang and Zhi-Hua Zhou. Cost-sensitive face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32:1758–1769, 2010.

[Zhou and Liu, 2006] Zhi-Hua Zhou and Xu-Ying Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *Knowledge and Data Engineering, IEEE Transactions on*, 18:63–77, 2006.