Online Incremental Feature Learning with Denoising Autoencoders

Guanyu Zhou
Department of EECS
University of Michigan
Ann Arbor, MI 48109
guanyuz@umich.edu

Kihyuk Sohn
Department of EECS
University of Michigan
Ann Arbor, MI 48109
kihyuks@umich.edu

Honglak Lee
Department of EECS
University of Michigan
Ann Arbor, MI 48109
honglak@eecs.umich.edu

Abstract

While determining model complexity is an important problem in machine learning, many feature learning algorithms rely on cross-validation to choose an optimal number of features, which is usually challenging for online learning from a massive stream of data. In this paper, we propose an incremental feature learning algorithm to determine the optimal model complexity for large-scale, online datasets based on the denoising autoencoder. This algorithm is composed of two processes: *adding* features and *merging* features. Specifically, it adds new features to minimize the objective function's residual and merges similar features to obtain a compact feature representation and prevent over-fitting. Our experiments show that the proposed model quickly converges to the optimal number of features in a large-scale online setting. In classification tasks, our model outperforms the (non-incremental) denoising autoencoder, and deep networks constructed from our algorithm perform favorably compared to deep belief networks and stacked denoising autoencoders. Further, the algorithm is effective in recognizing new patterns when the data distribution changes over time in the massive online data stream.

1 Introduction

In recent years, there has been much interest in online learning algorithms [1, 9, 29, 8] with the purpose of developing an intelligent agent that can perform life-long learning in complex real environments. To that end, many large-scale learning algorithms have also been proposed. In particular, Support Vector Machines (SVMs) [5] have been one of the most popular discriminative methods for large-scale learning which includes selective sampling to reduce the number of support vectors [4, 24] and incremental support vectors to learn an SVM with a small number of examples in the early phase of training [7, 23]. Typically, linear SVMs are used in large-scale settings due to their efficient training as well as the scalability to large-scale datasets. However, this approach is limited in that the feature mapping has to be fixed during the training; therefore, it cannot be adapted to the training data.

Alternatively, there are several methods for jointly training the feature mapping and the classifier, such as via multi-task learning [31, 6], transfer learning [2, 22, 17], non-parametric Bayesian learning [1, 10], and deep learning [16, 3, 28, 27, 20]. Among these, we are interested in deep learning approaches that have shown promise in learning features from complex, high-dimensional unlabeled and labeled data. Specifically, we present a large-scale feature learning algorithm based on the denoising autoencoder (DAE) [32]. The DAE is a variant of autoencoders [3] that extracts features by adding perturbations to the input data and attempting to reconstruct the original data. This process learns features that are robust to input noise and useful for classification.

Despite the promise, determining the optimal model complexity, i.e., the number of features for DAE, still remains a nontrivial question. When there are too many features, for example, the model may overfit the data or converge very slowly. On the other hand, when there are too few features, the model may underfit due to the lack of relevant features. Further, finding an optimal feature set size becomes even more difficult for large-scale or online datasets whose distribution may change over time, since the cross-validation may be challenging given a limited amount of time or compu-

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume XX of JMLR: W&CP XX.

tational resources.

To address this problem, we propose an incremental algorithm to learn features from the large-scale online data by adaptively incrementing the features depending on the data and the existing features, using DAE as a basic building block. Specifically, new features are added and trained to minimize the residual of the objective function in generative tasks (e.g., minimizing reconstruction error) or discriminative tasks (e.g., minimizing supervised loss function). At the same time, similar features are merged to avoid redundant feature representations. Experimental results show that our incremental feature learning algorithms perform favorably compared to the non-incremental feature learning algorithms, including the standard DAE, the deep belief network (DBN), and the stacked denoising autoencoder (SDAE), on classification tasks with large datasets. Moreover, we show that incremental feature learning is more effective in quickly recognizing new patterns than the non-incremental algorithms when the data distribution (e.g., the ratio of labels) is highly non-stationary.

2 Preliminaries

In this paper, we use the denoising autoencoder [32] as a building block for incremental feature learning. Based on the idea that good features should be robust to input corruption, the DAE tries to recover the input data $\mathbf{x}$ from encoded representation $f(\widetilde{\mathbf{x}})$ of corrupted data $\widetilde{\mathbf{x}}$ via a decoding function $g(\mathbf{h})$. More precisely, there are four components in the DAE:

- A conditional distribution $q(\widetilde{\mathbf{x}}|\mathbf{x})$, that stochastically perturbs input $\mathbf{x}$ to a corrupted version $\widetilde{\mathbf{x}}$.
- An encoding function $f(\widetilde{\mathbf{x}}) \equiv \mathbf{h} \in \mathbb{R}^N$, which gives a representation of input data $\mathbf{x}$.
- A decoding function $g(\mathbf{h}) \equiv \widehat{\mathbf{x}} \in \mathbb{R}^D$, which recovers the data from the representation $\mathbf{h}$.
- A differentiable cost function $\mathcal{L}(\mathbf{x})$ computes the dissimilarity between input and reconstruction.

Throughout the paper, we consider a case in which the input and the hidden variables are binary (or bounded between 0 and 1), i.e., $\mathbf{x} \in [0,1]^D$ and $\mathbf{h} \in [0,1]^N$. As described in [32], we consider the conditional distribution $q(\widetilde{\mathbf{x}}|\mathbf{x}) = \prod_{i=1}^{D} q(\widetilde{x}_i|x_i)$ that randomly sets some of the coordinates to zeros. By corrupting the input elements, the DAE attempts to learn informative representations that can successfully recover the missing coordinates to reconstruct the original input data. In other words, the DAE is trained to *fill in the missing values* introduced by corruption.

The DAE perturbs the input $\mathbf{x}$ to $\widetilde{\mathbf{x}}$ and maps it to the hidden representation, or abusing notation, the *feature* $\mathbf{h}$ through the encoding function defined as follows:

$$\mathbf{h} = f(\widetilde{\mathbf{x}}) = sigm(\mathbf{W}\widetilde{\mathbf{x}} + \mathbf{b}),$$

where $\mathbf{W} \in \mathbb{R}^{N \times D}$ is a weight matrix, $\mathbf{b} \in \mathbb{R}^N$ is a hidden bias vector, and $sigm(s) = \frac{1}{1+\exp(-s)}$ is the sigmoid function. Then, we reconstruct $\widehat{\mathbf{x}} \in [0,1]^D$ from $\mathbf{h}$ using the decoding function:

$$\widehat{\mathbf{x}} = g(\mathbf{h}) = sigm(\mathbf{W}^T\mathbf{h} + \mathbf{c}),$$

where $\mathbf{c} \in \mathbb{R}^D$ is an input bias vector. Here, we implicitly assumed tied weights for encoding and decoding functions, but we can consider separate weights as well.

In this work, we use cross-entropy as a cost function:

$$\mathcal{L}(\mathbf{x}) = \mathcal{H}(\mathbf{x}, \widehat{\mathbf{x}}) = -\sum_{i=1}^{D} x_i \log \widehat{x}_i + (1 - x_i) \log(1 - \widehat{x}_i)$$

Finally, the objective is to learn model parameters that minimize the cost function

$$\{\widehat{\mathbf{W}}, \widehat{\mathbf{b}}, \widehat{\mathbf{c}}\} = \arg\min_{\mathbf{W},\mathbf{b},\mathbf{c}} \sum_j \mathcal{L}(\mathbf{x}^{(j)}),$$

and this can be optimized by gradient descent (or conjugate gradient, L-BFGS, etc.) with respect to all parameters for encoding and decoding functions.

3 Incremental Feature Learning

In many cases, the number of hidden units N is fixed during the training of DAE and the optimal value is determined by cross-validation. However, cross-validation is computationally prohibitive and often infeasible for large datasets. Furthermore, it may not perform well when the distribution of training data significantly changes over time.

To address these issues, we present an adaptive feature learning algorithm that can handle large-scale datasets without explicitly performing cross-validation over the number of features on the whole dataset. In detail, our incremental feature learning algorithm is composed of two key processes: (1) adding new feature mappings to the existing feature set and (2) merging parts of the existing feature mappings that are considered redundant. We describe the details of the proposed algorithm in this section.

3.1 Overview

The incremental feature learning algorithm addresses the following two issues: (1) *when* to add and merge features and (2) *how* to add and merge features. To deal with these problems systematically, we define an objective function to determine when to add or merge

Algorithm 1 Incremental feature learning

repeat
  Compute the objective $\mathcal{L}(\mathbf{x})$ for input $\mathbf{x}$.
  Collect hard examples into a subset B (i.e., $B \leftarrow B \cup \{\mathbf{x}\}$ if $\mathcal{L}(\mathbf{x}) > \mu$).
  if $|B| > \tau$ **then**
    Select $2\Delta M$ candidate features and merge them into ΔM features (Section 3.3).
    Add ΔN new features by greedily optimizing with respect to the subset B (Section 3.2).
    Set $B = \emptyset$ and update ΔN and ΔM.
  end if
  Fine-tune all the features jointly with in current batch of data (i.e., optimize via gradient descent with respect to all parameters).
until convergence

features and how to learn the new features. Specifically, we form a set B which is composed of hard training examples whose objective function values are greater than a certain threshold (μ) and use these examples as input data to greedily initialize the new features. However, some of the collected hard examples may be irrelevant to the task of interest. Therefore, we only begin adding features when there are enough examples collected in B (i.e., $|B| > \tau$). In our implementation, we set μ as the average of the objective function values for recent 10,000 training examples; the τ was set also to 10,000. These values were not tuned. Note that it is not atypical to use 10,000 examples for training when using conjugate gradient or L-BFGS (e.g., [21]).

After incrementing new features (i.e., initializing new features and adding them to the feature set), we jointly train all features with the upcoming training examples. Note that our feature learning algorithm still optimizes the original objective function for the training data, and the subset B is used only for initializing the feature increments. The overall algorithm is schematically shown in Figure 1, and a pseudo-code is provided in Algorithm 1. In the following section, we will describe adding new features and merging similar existing features based both on the generative and discriminative objective functions.

Notations. In this paper, we denote D for input dimension, N for the number of (existing) features, and K for the number of class labels. Based on these notations, the parameters for generative training are defined as the (tied) weight matrix $\mathbf{W} \in \mathbb{R}^{N \times D}$, the hidden bias $\mathbf{b} \in \mathbb{R}^N$, and the input bias $\mathbf{c} \in \mathbb{R}^D$. Similarly, the parameters for discriminative training are composed of the weight matrix $\mathbf{\Gamma} \in \mathbb{R}^{K \times N}$ between hidden and output units, the output bias $\boldsymbol{\nu} \in \mathbb{R}^K$, as well as the weight matrix $\mathbf{W}$ and the hidden bias $\mathbf{b}$. We use θ to denote all parameters $\{\mathbf{W}, \mathbf{b}, \mathbf{c}, \mathbf{\Gamma}, \boldsymbol{\nu}\}$.

For incremental feature learning, we use $\mathcal{N}$ to denote *new* features and $\mathcal{O}$ to denote existing or *old* features. For example, $f_{\mathcal{O}}(\widetilde{\mathbf{x}}) \equiv \mathbf{h}_{\mathcal{O}} \in [0,1]^N$ represents an encoding function with existing features, and $f_{\mathcal{N}}(\widetilde{\mathbf{x}}) \equiv \mathbf{h}_{\mathcal{N}} \in [0,1]^{\Delta N}$ denotes an encoding function with newly added features. A combined encoding function is then written as $f_{\mathcal{O}\cup\mathcal{N}}(\widetilde{\mathbf{x}}) = [\mathbf{h}_{\mathcal{O}}; \mathbf{h}_{\mathcal{N}}] \in [0,1]^{N+\Delta N}$. Likewise, $\theta_{\mathcal{O}}$ refers to the existing parameters, i.e., $\{\mathbf{W}_{\mathcal{O}}, \mathbf{b}_{\mathcal{O}}, \mathbf{c}, \mathbf{\Gamma}_{\mathcal{O}}, \boldsymbol{\nu}\}$, and $\theta_{\mathcal{N}} = \{\mathbf{W}_{\mathcal{N}}, \mathbf{b}_{\mathcal{N}}, \mathbf{c}, \mathbf{\Gamma}_{\mathcal{N}}, \boldsymbol{\nu}\}$ denotes the parameters for new features.

3.2 Adding features

In this section, we describe an efficient learning algorithm for adding features with different types of objective functions. The basic idea for efficient training is that only the new features and corresponding parameters $\theta_{\mathcal{N}}$ are trained to minimize the objective function while keeping $\theta_{\mathcal{O}}$ fixed. In the following subsections, we expand on the different training criteria, such as generative, discriminative, and hybrid objectives.

3.2.1 Generative training

A generative objective function measures an average reconstruction error between the input $\mathbf{x}$ and the reconstruction $\widehat{\mathbf{x}}$. The cross-entropy function, assuming that the input and output values are both within interval $[0,1]$, is used as an objective function. The optimization problem is posed as:

$$\min_{\mathbf{W}_{\mathcal{N}}, \mathbf{b}_{\mathcal{N}}} \quad \frac{1}{|B|} \sum_{i \in B} \mathcal{L}_{gen}(\mathbf{x}^{(i)}), \tag{1}$$

where $\mathcal{L}_{gen}(\mathbf{x}) = \mathcal{H}(\mathbf{x}, \widehat{\mathbf{x}})$. For incremental feature learning, the DAE optimizes the new features to reconstruct the data $\mathbf{x}$ from the corrupted data $\widetilde{\mathbf{x}}$. The encoding and decoding functions for the new features can be written as:

$$\mathbf{h}_{\mathcal{N}} = sigm(\mathbf{W}_{\mathcal{N}} \widetilde{\mathbf{x}} + \mathbf{b}_{\mathcal{N}}), \tag{2}$$

$$\widehat{\mathbf{x}} = sigm(\mathbf{W}_{\mathcal{N}}^T \mathbf{h}_{\mathcal{N}} + \mathbf{W}_{\mathcal{O}}^T \mathbf{h}_{\mathcal{O}} + \mathbf{c}). \tag{3}$$

The key idea here is that, although the output $\widehat{\mathbf{x}}$ depends on both new features $\mathbf{h}_{\mathcal{N}}$ and old features $\mathbf{h}_{\mathcal{O}}$ as described in equation (3), the training of incremental feature $\theta_{\mathcal{N}}$ that minimizes the *residual* of the objective function is still highly efficient because $\theta_{\mathcal{O}}$ is fixed during the training. From another point of view, we can interpret $\mathbf{W}_{\mathcal{O}}^T \mathbf{h}_{\mathcal{O}}$ as a part of the bias. Thus, we can rewrite equation (3) as:

$$\widehat{\mathbf{x}} = sigm(\mathbf{W}_{\mathcal{N}}^T \mathbf{h}_{\mathcal{N}} + c_d(\mathbf{h}_{\mathcal{O}})), \tag{4}$$

where $c_d(\mathbf{h}_{\mathcal{O}}) \equiv \mathbf{W}_{\mathcal{O}}^T \mathbf{h}_{\mathcal{O}} + \mathbf{c}$ is viewed as a *dynamic* decoding bias. Since the parameters for existing fea-

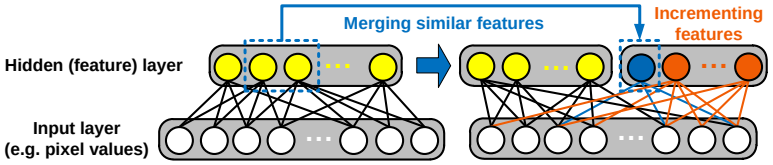


Figure 1: Illustration of single layer incremental feature learning. During the training, our incremental feature learning algorithm optimizes the parameters of new features (the orange units and the blue units on the right), while holding the other features (yellow units outside the dotted blue box on the left). The orange units are incremental features, and the blue unit is the merged feature from the similar existing features depicted as yellow units in the dotted blue box. See text for details.

tures and the corresponding activations are not changing during the training of new features, we compute $\mathbf{h}_{\mathcal{O}}$ once and recall the $c_d(\mathbf{h}_{\mathcal{O}})$ repeatedly for each training example without additional computational cost. In this way, we can efficiently optimize the new parameters greedily via stochastic gradient descent.

3.2.2 Discriminative training

A discriminative objective function computes an average classification loss between the actual label $\mathbf{y} \in [0,1]^K$ and the predicted label $\widehat{\mathbf{y}} \in [0,1]^K$. More precisely, the cross-entropy is used as a performance measure and we pose an optimization problem as follows:

$$\min_{\mathbf{W}_{\mathcal{N}},\mathbf{b}_{\mathcal{N}},\mathbf{\Gamma}_{\mathcal{N}}} \quad \frac{1}{|B|}\sum_{i\in B}\mathcal{L}_{disc}(\mathbf{x}^{(i)},\mathbf{y}^{(i)}), \tag{5}$$

where $\mathcal{L}_{disc}(\mathbf{x},\mathbf{y}) = \mathcal{H}(\mathbf{y},\widehat{\mathbf{y}}(\mathbf{x}))$. Note that the label $\mathbf{y}$ is a binary vector with a softmax unit that allows one element to be 1 out of K dimensions for K-way classification problem (i.e., $y_k = 1$ if and only if the example is in the k-th class). Formally speaking, the discriminative model predicts $\widehat{\mathbf{y}}$ as a posterior probability of class labels via the softmax activation function

$$\widehat{\mathbf{y}} = \text{softmax}(\boldsymbol{\nu} + \mathbf{\Gamma}_{\mathcal{O}} f_{\mathcal{O}}(\widetilde{\mathbf{x}}) + \mathbf{\Gamma}_{\mathcal{N}} f_{\mathcal{N}}(\widetilde{\mathbf{x}})), \tag{6}$$

where $\text{softmax}(\mathbf{a})_k = \frac{\exp(a_k)}{\sum_{k'}\exp(a_{k'})}, k = 1,...,K$ for $\mathbf{a} \in \mathbb{R}^K$. A similar interpretation for $\boldsymbol{\nu}+\mathbf{\Gamma}_{\mathcal{O}} f_{\mathcal{O}}(\widetilde{\mathbf{x}})$, as in the generative training, is possible, and therefore, we can efficiently train the new parameters $\{\mathbf{W}_{\mathcal{N}}, \mathbf{b}_{\mathcal{N}}, \mathbf{\Gamma}_{\mathcal{N}}\}$ using gradient descent.

3.2.3 Hybrid training

Considering the discriminative model as a single objective function has a risk of overfitting. As a remedy, we can use a hybrid objective function that combines the discriminative and generative loss function as follows:

$$\mathcal{L}_{hybrid}(\mathbf{x},\mathbf{y}) = \mathcal{L}_{disc}(\mathbf{x},\mathbf{y}) + \lambda\mathcal{L}_{gen}(\mathbf{x}). \tag{7}$$

In hybrid objective function, we further normalize both loss functions with respect to their target dimensions. In our experiments, we found that any value of $\lambda \in [0.1, 0.4]$ gave roughly the best performance.

3.3 Merging features

As described in the previous section, incremental feature mappings can be efficiently trained to improve the generative and discriminative objectives for online datasets. However, monotonically adding features could potentially result in many redundant features and overfitting. To deal with this problem, we consider merging similar features to produce more compact feature representations, which can mitigate the problem of overfitting as well.

Our merging process is done in two steps: we select a pair of candidate features and merge them to a single feature. Detailed descriptions are given below:

- Select a set of candidate features to be merged, $\mathcal{M} = \{m_1, m_2\} \subset \mathcal{O}$, and replace $f_{\mathcal{O}}$ by $f_{\mathcal{O}\setminus\mathcal{M}}$ (i.e., *remove* m_1-th and m_2-th features from $\mathcal{O}$).
- Add a new feature mapping $f_{\mathcal{N}}$ that *replaces* the merged features (See also Section 3.2).

The candidate feature pair and the new feature mapping can be determined by solving the following optimization problem:

$$\min_{\theta_{\mathcal{N}},\mathcal{M}} \quad \frac{1}{|B|}\sum_{i\in B}\mathcal{L}_{hybrid}(\mathbf{x}^{(i)},\mathbf{y}^{(i)}). \tag{8}$$

However, optimizing (8) jointly with respect to $\theta_{\mathcal{N}}$ and $\mathcal{M}$ is computationally expensive because the complexity of an exhaustive search over all pairs of candidate features increases quadratically in the number of features. As an approximation, we decompose the original optimization problem into a two-step process, where we first find the most *similar* feature pair for merging candidates, and then solve the optimization problem with $\theta_{\mathcal{N}}$ only. In addition to being efficient in training, this approximate optimization process also minimizes the upper bound of the objective function in equation (8). The process is described below:

- Find a pair of features whose distance is minimal: $\widehat{\mathcal{M}} = \arg\min_{\{m_1,m_2\}} d(\mathbf{W}_{m_1}, \mathbf{W}_{m_2})$.
- Add new features to $\theta_{\mathcal{O}\setminus\mathcal{M}}$ by solving $\widehat{\theta}_{\mathcal{N}} = \arg\min_{\theta_{\mathcal{N}}} \frac{1}{|B|} \sum_{i\in B} \mathcal{L}_{hybrid}(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$.

Given the candidate features to merge, a newly added feature can be trained via gradient descent as described in the previous section. Unlike section 3.2, however, we initialize the new feature parameters as a weighted average (i.e., linear combination) of two candidate feature parameters for faster convergence as

$$\theta_{\mathcal{N}} = \frac{\sum_{\mathbf{x}\in B} P(h_{m_1}|\mathbf{x};\theta_{m_1})\theta_{m_1} + P(h_{m_2}|\mathbf{x};\theta_{m_2})\theta_{m_2}}{\sum_{\mathbf{x}\in B} P(h_{m_1}|\mathbf{x};\theta_{m_1}) + P(h_{m_2}|\mathbf{x};\theta_{m_2})}.$$

3.4 General formulations

In this section, we discuss variations of our algorithm given four criteria: (1) encoding and decoding functions of DAE depending on the type of input data, (2) discriminative objectives, (3) distance functions for determining merged candidates, and (4) an extension to a deep incremental network.

Encoding and decoding functions. In previous sections, we developed our algorithm based on sigmoid functions for both encoder and decoder. However, other linear and non-linear functions (e.g., *tanh*, *linear*, or *rectified-linear*) can be adapted. For example, we can use a linear decoding function with a mean squared error loss for real-valued input.

Discriminative training. In section 3.2.2, we used a softmax classifier with cross-entropy for discriminative tasks. However, it can easily be generalized to other classifiers. For example, SVM can be adopted to our algorithm by defining a hinge-loss (primal objective of the SVM) at the output layer of the network.

Merging process. We used cosine distance (normalized inner product) for all the experiments. Alternatively, we can consider other similarity functions, such as KL divergence, intersection kernel, and Chi-square kernel, to select candidate features to merge. Furthermore, we note that in addition to our merging technique as described in Sec. 3.3, other merging methods or pruning techniques, such as "forward search," "backward search," or pruning infrequently activated features, can be used to reduce model complexity.

Deep incremental network. The proposed incremental learning method can be readily adapted to deep architectures. For example, we can train the deep incremental network that determines the optimal feature set size of multiple layers in a greedy way by fixing the number of features in the lower layers and incrementing the features only at the top-layer sequentially.

4 Connection to Boosting and Related Work

In this section, we provide a comparison between incremental feature learning and boosting algorithms because they share similar motivations and properties. Boosting (e.g., [18, 13, 30, 25, 15, 14, 33]) is a learning framework that trains a single strong learner from a set of weak learners. Incremental feature learning and boosting algorithms are similar in that they add weak learners (or *feature mapping* corresponding to newly-added hidden units in the context of feature learning) to improve the existing classifiers. However, incremental feature learning determines the number of feature increments based on the performance, while typical boosting methods introduce the pre-determined number of weak learners at each stage. Further, the merging process makes our algorithm robust to overfitting and to be more adaptive to the online stream of data whose distribution (e.g., data or class labels) may fluctuate over time, since it removes the redundant weak learners effectively.

Importantly, the weak learners in boosting algorithms are typically used as classifiers (for predicting class labels); whereas in our setting, they can be feature mappings to improve classification performance or to learn representations from input data, which makes our algorithm flexibly applicable to unsupervised, semi-supervised, or self-taught learning [26] settings. Similar to our approach, Chen et al. [11] presented an incremental feature learning algorithm, called boosted sparse coding, that adds a single feature at a time using sparse coding objectives. Our incremental learning algorithm is different in that (1) we add multiple features at a time based on several performance criteria (i.e., generative/discriminative/hybrid objectives); (2) we have a merging process that helps avoid overfitting; and (3) we optimize the whole network (i.e., via fine-tuning) after initializing new features.

In addition to boosting, our model is also related to cascade correlation [12]. A cascade correlation neural network adds a new hidden unit as a new layer at each step, which will depend on all previously added units. Thus, a cascade correlation neural network cannot effectively learn a large number of hidden units (which would mean a very deep network). However, in our proposed model, all hidden units (assuming one hidden-layer model) can be independently computed given input data, which makes our training more efficient. In addition, instead of simply adding hidden units, our model is more adaptive to the input data by adding or merging features. In our experiments, the cascade correlation neural network performed well with dozens of hidden units; however, in more complicated tasks, our model always outperformed the cas-

cade correlation by a large margin.

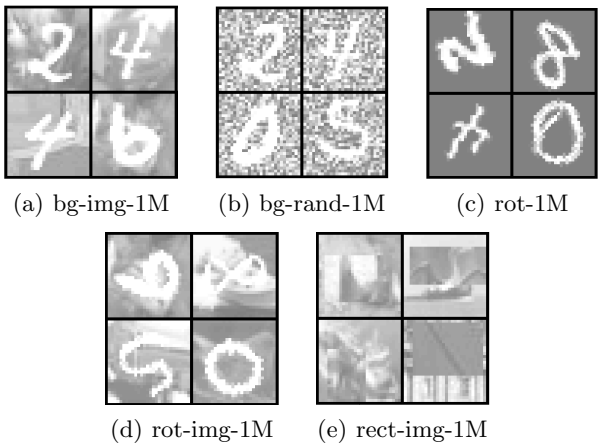
5 Experimental Results

5.1 Experimental setup

We evaluate the performance of the proposed learning algorithm on the large-scale extended datasets based on the MNIST variation and rect-images datasets.[1] The MNIST variation dataset is composed of digit images with different variation types, such as rotation, random, or image background, and their combination. For rect-images dataset, each image contains a single rectangle with different aspect ratios filled with an image that is different from its background image. Since the original datasets have a limited number of training examples, we extended the dataset sizes to 1 million. Samples from these datasets are shown in Figure 2. We note that our extended datasets are more challenging since we used more diverse background images (randomly selected from 2000 images from CIFAR-10 [19]), whereas the original MNIST variation datasets used background patches extracted from 20 images downloaded from the internet. (See supplementary material for further discussion about the datasets.)

In all experiments, we used a 20% corruption rate for the DAE, and we implemented all algorithms via Jacket GPU library running on nVidia GTX 470. For optimization, we used L-BFGS[2] with a minibatch size of 1,000, since a similar setting has shown good performance in training autoencoders without tuning the learning rate [21]. For all feature learning algorithms, we trained features by pre-training on the first 12,000 training examples, followed by supervised fine-tuning in an online setting.

For incremental feature learning, the number of feature increments ΔN_t and the number of merged features ΔM_t at time t are adaptively determined by monitoring performance. In this study, we mainly considered two types of update rules. With the first type of update rule, when the performance of current online data batch is far from the convergence, we increase ΔN_t to accelerate. On the other hand, if the performance difference between two consecutive batches is very small, we regard it as a convergence and reduce ΔN_t to decelerate the pace. To balance the model, we keep ΔM_t proportional to ΔN_t during updates. Alternatively, with the second type of update rule, we considered a combination of AIC and BIC to control the number of features. This way, the model starting with a large number of features can still recover from potential overfitting by penalizing the model complexity. Note that all results reported in this section are based on the latter update rule.



(a) bg-img-1M (b) bg-rand-1M (c) rot-1M

(d) rot-img-1M (e) rect-img-1M

Figure 2: Samples from the extended MNIST datasets and the extended *rect* dataset.

We also evaluated the classification accuracy with other variants of update rules (for ΔN & ΔM) and found that the performance was fairly robust to the choice of update rules. For more details about selecting the number of incremental and merged features, see the supplementary material.

5.2 Evaluation on generative criteria

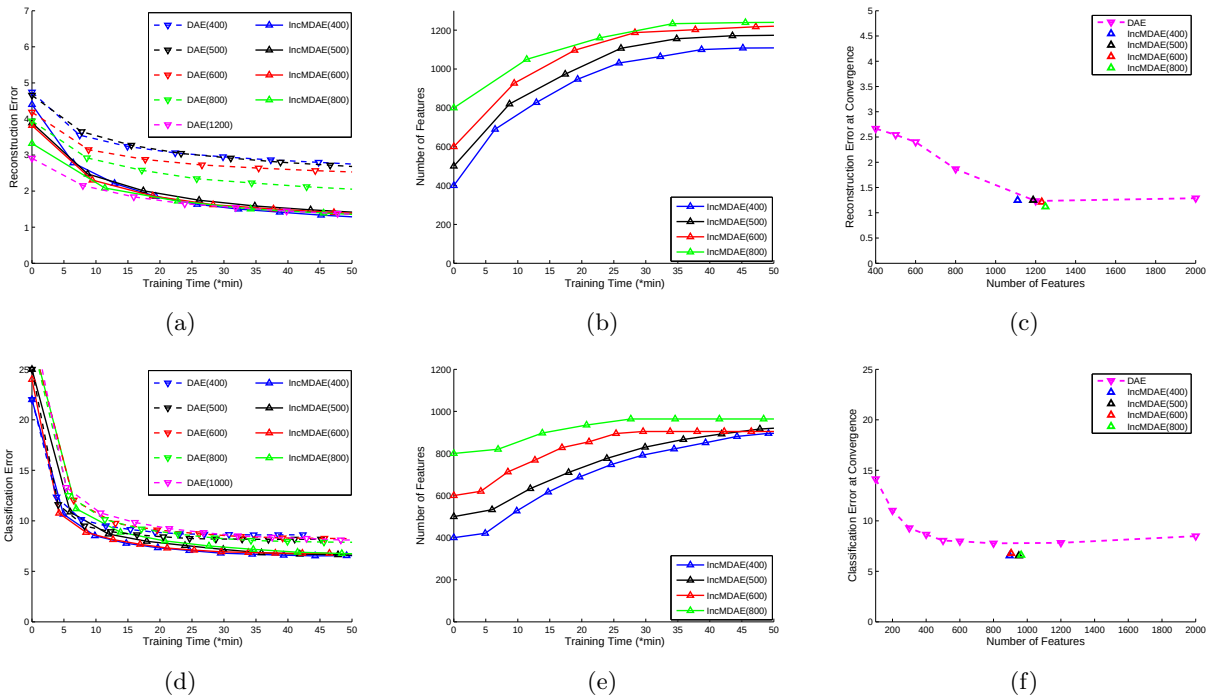
In this experiment, we compare our algorithm to the non-incremental DAE in generative criteria. Specifically, we tested on the bg-img-1M dataset and show the online reconstruction error and the number of features over training time in Figure 3(a) and 3(b), respectively. We report the results with 400, 500, 600 and 800 initial features for both incremental and non-incremental learning. In addition, we report the result of non-incremental learning with 1200 features, which is approximately the optimal feature set size obtained from incremental feature learning. In Figure 3(b) and 3(c), we observe that the incremental feature learning algorithm could find an approximately optimal number of features regardless of its initial feature set size, and it led to the smaller or comparable reconstruction error compared to its non-incremental counterpart.

5.3 Evaluation on discriminative criteria

To evaluate discriminative performance of our algorithm, we ran experiments using a hybrid objective function, a combination of generative and discriminative objective functions (with $\lambda = 0.2$). First, we compare the classification performance of the incremental feature learning algorithm with that of its non-incremental counterpart on bg-img-1M dataset.

[1] http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/MnistVariations

[2] http://www.di.ens.fr/~mschmidt/Software/



(a) (b) (c)

(d) (e) (f)

Figure 3: (a,d) Test (online) reconstruction error and classification error (on the bg-img-1M dataset) of incremental (IncMDAE) and non-incremental DAE. (b,e) The number of features for incremental learning with generative and hybrid criteria. (c,f) Test reconstruction error and classification error at convergence. The numbers in parentheses refer to the initial number of features.

Similar to section 5.2, we report the classification errors in Figure 3(d) using 400, 500, 600 and 800 initial features for both incremental and non-incremental algorithms and additionally plot the result of non-incremental learning with 1000 features. Overall, as Figure 3(d), 3(e) and 3(f) indicate, we observed trends similar to the results using the generative criteria. Figure 3(d) and 3(f) shows that incremental DAEs find an approximately optimal number of features (regardless of the initial feature set sizes) while achieving consistently lower classification errors compared to those from non-incremental counterparts.

For more extensive evaluation, we tested the incremental feature learning algorithms with merging (IncMDAE) and without merging (IncDAE) on the extended datasets. In addition, we evaluated (non-incremental) 1-layer DAE, linear SVM, and online multiclass LPBoost [30]. All algorithms were tested in an online manner (i.e., evaluating on the next mini-batch of examples that had not previously been seen during training). As Table 1 shows, both the 1-layer incremental feature learning algorithm with merging (IncMDAE^{l1}) and without merging (IncDAE^{l1}) outperformed linear SVM, DAE^{l1} and online multiclass LPBoost. The results suggest that, although it is similar to boosting, our method can effectively use both unlabeled data and labeled data (via pre-training followed by supervised/hybrid fine-tuning). In addition, IncMDAE consistently achieved classification performance similar to or better than IncDAE, suggesting that the merging process is effective at maintaining the balance between specificity and robustness by learning less redundant features.

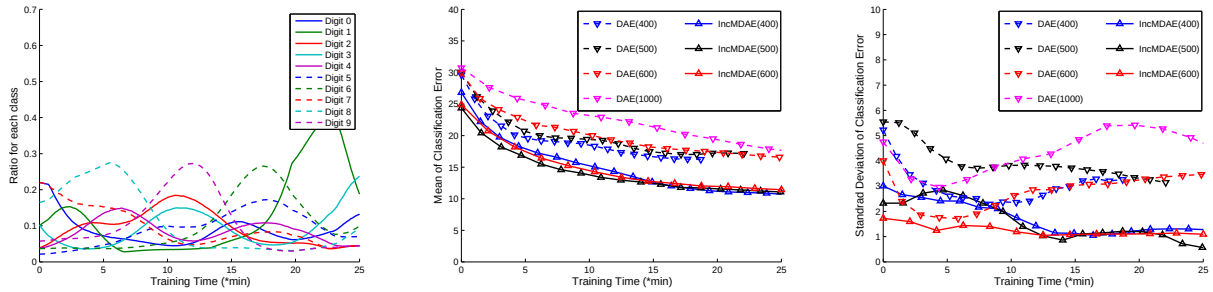
We also evaluated deep network extensions of our algorithm and compared against the deep belief network (referred to as "DBN^{l1-3}") and the stacked denoising autoencoder (referred to as "SDAE^{l1-3}") that both contained 3 layers. For both the DBN and the SDAE, the number of hidden units for each layer was selected out of $\{500, 1000, 2000\}$ via cross-validation. As shown in Table 1, the deep incremental networks (IncMDAE^{l1,2} and IncMDAE^{l1-3}) performed favorably compared to these standard deep networks. The results suggest that our incremental DAE can be a useful building block in learning deep networks.

5.4 Evaluating on non-stationary distributions

One challenge in online learning is to fit the model to the data whose distribution fluctuates over time. In this section, we perform an experiment to see how well the incremental learning algorithm can adapt to the online stream of data as compared to its non-incremental counterpart.

Dataset	rot-bg-img-1M	rot-1M	bg-rand-1M	bg-img-1M	rect-img-1M
linear SVM	55.17 ± 0.45	24.02 ± 0.33	19.17 ± 0.33	29.24 ± 0.46	33.18 ± 0.38
DAE^{l1}	38.51 ± 0.45	10.23 ± 0.35	7.94 ± 0.24	7.65 ± 0.30	24.27 ± 0.31
OMCLPBoost	42.11 ± 0.04	12.02 ± 0.02	7.22 ± 0.03	10.29 ± 0.09	29.51 ± 0.13
DBN^{l1-3}	30.01 ± 0.31	8.00 ± 0.31	$\mathbf{6.27 \pm 0.35}$	9.20 ± 0.19	16.36 ± 0.31
SDAE^{l1-3}	30.13 ± 0.25	7.76 ± 0.35	$\mathbf{6.22 \pm 0.33}$	7.56 ± 0.31	13.20 ± 0.30
IncDAE^{l1}	35.50 ± 0.41	10.28 ± 0.23	7.00 ± 0.10	7.20 ± 0.17	18.50 ± 0.34
IncMDAE^{l1}	35.41 ± 0.49	8.20 ± 0.38	6.90 ± 0.33	6.37 ± 0.28	14.15 ± 0.32
IncMDAE^{l1,2}	29.04 ± 0.45	$\mathbf{6.60 \pm 0.33}$	$\mathbf{6.14 \pm 0.30}$	6.20 ± 0.31	$\mathbf{9.28 \pm 0.44}$
IncMDAE^{l1-3}	$\mathbf{27.60 \pm 0.40}$	$\mathbf{6.80 \pm 0.44}$	$\mathbf{6.30 \pm 0.26}$	$\mathbf{5.24 \pm 0.34}$	9.89 ± 0.54

Table 1: Test (online) error for classification tasks on large-scale datasets. The best performing methods for each dataset (within the standard errors) are shown in bold.



(a) (b) (c)

Figure 4: Effects of non-stationary data distribution on bg-img-1M dataset. (a) An example non-stationary distribution (of class labels) generated from a Gaussian Process. (b) Average test (online) classification error. (c) Standard deviation of the test (online) classification error.

We generated training data using bg-img-1M dataset whose label distribution changes over time; the ratio of labels at time t was randomly assigned for each class via $\text{ratio}_k(t) = \frac{\exp\{a_k(t)\}}{\sum_{j=1}^{K}\exp\{a_j(t)\}}, k = 1, ..., K$, where $a_k(t)$ is a random curve generated by Gaussian process (an example class distribution is shown in Figure 4(a)). We report results averaged over 5 random trials.

Figure 4 shows the mean and the standard deviation of test classification error on the fluctuating bg-img-1M dataset. Our incremental learning model shows a significantly lower mean classification error. Furthermore, in Figure 4(c), the low standard deviations of errors achieved by the IncMDAE models imply that our algorithm is more robust and can rapidly adapt to the fluctuating distribution. This result suggests that the incremental feature learning could be an effective training method for online learning with challenging non-stationary data distributions.

6 Conclusion

In this paper, we proposed an incremental feature learning algorithm based on denoising autoencoders. The proposed algorithm can learn good feature representations from large datasets by incrementing and merging features. Further, this algorithm can be used to avoid expensive cross-validation in selecting the number of features in large datasets. Our experimental results suggest that (1) incremental feature learning provides an efficient way to learn from large datasets by starting with a small set of initial features, and automatically adjusting the number of features as the training proceeds; (2) the proposed algorithm leads to comparable or lower reconstruction and classification errors than its non-incremental counterparts; (3) the incremental learning algorithm outperforms other state-of-the-art methods on large online datasets; and (4) incremental feature learning is more adaptive and robust to highly non-stationary input distributions. We believe our approach holds promise in developing scalable feature learning algorithms for large-scale datasets.

Acknowledgments

This work was supported in part by a Google Faculty Research Award.

References

[1] R. P. Adams, H. M. Wallach, and Z. Ghahramani. Learning the stucture of deep sparse graphical models. In *AISTATS*, 2010.

[2] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. *NIPS*, 2007.

[3] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, 2007.

[4] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, 2005.

[5] B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, 1992.

[6] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[7] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *NIPS*, 2001.

[8] N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

[9] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. An online algorithm for large scale image similarity learning. In *NIPS*, 2009.

[10] B. Chen, G. Polatkan, G. Sapiro, D. Dunson, and L. Carin. The hierarchical beta process for convolutional factor analysis and deep learning. In *ICML*, 2011.

[11] B. Chen, K. Swersky, B. Marlin, and N. de Freitas. Sparsity priors and boosting for learning localized distributed feature representations. Technical report, University of British Columbia, 2010.

[12] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. *NIPS*, 1990.

[13] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. *ICML*, 1996.

[14] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[15] A. Grubb and J. A. Bagnell. Boosted backpropagation learning for training deep modular networks. *ICML*, 2010.

[16] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[17] T. Jebara. Multi-task feature and kernel selection for SVMs. In *ICML*, 2004.

[18] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

[19] A. Krizhevsky. Learning multiple layers of features from Tiny Images. Master's thesis, University of Toronto, 2009.

[20] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML*, 2007.

[21] Q. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng. On optimization methods for deep learning. In *ICML*, 2011.

[22] S.-I. Lee, V. Chatalbashev, D. Vickrey, and D. Koller. Learning a meta-level prior for feature relevance from multiple related tasks. In *ICML*, 2007.

[23] X. Liu, G. Zhang, Y. Zhan, and E. Zhu. An Incremental Feature Learning Algorithm Based on Least Square Support Vector Machine. *Frontiers in Algorithmics*, pages 330–338, 2008.

[24] G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. In *Large Scale Kernel Machines*, pages 301–320. MIT Press, 2007.

[25] N. C. Oza. Online ensemble learning. *Ph.D. Thesis, University of California, Berkeley*, 2001.

[26] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: Transfer learning from unlabeled data. In *ICML*, 2007.

[27] M. Ranzato, Y. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. *NIPS*, 2007.

[28] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *NIPS*, 2006.

[29] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich. Subgradient methods for maximum margin structured learning. In *ICML*, 2006.

[30] A. Saffari, M. Godec, T. Pock, C. Leistner, and H. Bischo. Online multi-class lpboost. *CVPR*, 2010.

[31] S. Thrun. Is learning the n-th thing any easier than learning the first? *NIPS*, 1996.

[32] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

[33] R. S. Zemel and T. Pitassi. A gradient-based boosting algorithm for regression problems. *NIPS*, 2001.