

# Online Multi-Task Learning via Sparse Dictionary Optimization

**Paul Ruvolo**

Franklin W. Olin College of Engineering  
Department of Engineering  
paul.ruvolo@olin.edu

**Eric Eaton**

University of Pennsylvania  
Computer and Information Science Department  
eeaton@cis.upenn.edu

## Abstract

This paper develops an efficient online algorithm for learning multiple consecutive tasks based on the K-SVD algorithm for sparse dictionary optimization. We first derive a batch multi-task learning method that builds upon K-SVD, and then extend the batch algorithm to train models online in a lifelong learning setting. The resulting method has lower computational complexity than other current lifelong learning algorithms while maintaining nearly identical model performance. Additionally, the proposed method offers an alternate formulation for lifelong learning that supports both task and feature similarity matrices.

## Introduction

With the increasing interest in big data and deployed machine learning, it is essential to develop systems that can learn multiple, consecutive tasks over time. The idea of sharing knowledge between tasks to improve collective performance has been studied extensively (Kumar and Daumé 2012; Kang et al. 2011; Zhang et al. 2008; Caruana 1997) from a batch multi-task learning (MTL) perspective, in which task models are trained simultaneously. Recent work in online MTL (Ruvolo and Eaton 2013; Saha et al. 2011) has shown that it is possible to learn tasks consecutively and achieve nearly identical accuracy to batch MTL while dramatically reducing the computational cost. These capabilities are essential to the development of lifelong learning methods that accumulate and continually refine knowledge of multiple tasks over a lifetime of experience.

To facilitate knowledge transfer between task models, one common technique used by MTL algorithms is to learn and maintain a shared repository of latent model components; each task model is then given as a weighted combination of these components. This technique is used by several current MTL methods, including various batch algorithms (Kumar and Daumé 2012; Zhang et al. 2008) and the online Efficient Lifelong Learning Algorithm (ELLA) (Ruvolo and Eaton 2013). In batch MTL algorithms, the latent components are learned simultaneously with the task models in an expensive joint optimization. ELLA employs several simplifications to

eliminate the expensive optimization in support of online learning while minimizing the adverse effect on the performance of the resulting task models. Most notably, ELLA requires that each model, once learned, be based on fixed weights over the latent components. This may adversely affect performance by not permitting task models to adjust the weighting of individual latent components as these components become more refined due to additional training.

In this paper, we investigate an alternate formulation of online MTL based on the K-SVD algorithm (Aharon et al. 2006) that provides a reduction in computational complexity over ELLA, providing improved support for the rapid learning of consecutive tasks. This formulation, which we call ELLA-SVD, also eliminates one of ELLA’s simplifications, enabling task models to flexibly adjust their weights over latent components during learning. We compare this formulation to the original ELLA, a new version of ELLA that incorporates an iterative update step with a similar complexity to our K-SVD formulation, and a hybrid algorithm that blends these two approaches. We show that in some situations ELLA-SVD exhibits similar learning performance to ELLA while allowing continual refinement of all models and latent components. We also show that the hybrid approach yields a robust algorithm that exhibits high performance across a range of domains. We begin by providing an overview of the K-SVD algorithm, and then adapt K-SVD to learn task models in a batch MTL setting, yielding a new algorithm that we call MTL-SVD. We then modify the batch MTL-SVD algorithm to operate online, making it suitable for lifelong learning settings and yielding ELLA-SVD.

## The K-SVD Algorithm

This section reviews the K-SVD algorithm (Aharon et al. 2006) for learning dictionaries for sparse coding, which forms the foundation of our approach. Suppose we are designing a dictionary consisting of  $k$  vectors to sparsely code a set of data  $\{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^d$ . We would like to compute a dictionary  $\mathbf{L} \in \mathbb{R}^{d \times k}$  such that each input point can be coded with a minimal number of dictionary elements. This objective can be realized by solving:

$$\arg \min_{\mathbf{L}} \sum_{i=1}^n \min_{\mathbf{s}^{(i)}} \left\{ \|\mathbf{L}\mathbf{s}^{(i)} - \mathbf{x}_i\|_2^2 + \mu \|\mathbf{s}^{(i)}\|_0 \right\}, \quad (1)$$

---

**Algorithm 1** K-SVD (Aharon et al. 2006)

---

**input** data points  $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ , dictionary size  $k$   
init  $\mathbf{L}$  using random column vectors of unit length  
**loop** until convergence **do**  
  **for**  $i \in \{1, \dots, n\}$ , perform update in Eq. (2)  
  **for**  $j \in \{1, \dots, k\}$ , perform updates in Eqs. (4) – (5)  
**end loop**  
**return**  $\mathbf{L}$

---

where  $\mathbf{s}^{(i)} \in \mathbb{R}^k$  are coefficients over the columns of  $\mathbf{L}$  to encode  $\mathbf{x}_i$ , and  $\mu$  is a positive constant that defines the tradeoff between accurate reconstruction of the data and the sparsity of the coefficient vectors. This objective is computationally hard to optimize due to the cross terms between the dictionary  $\mathbf{L}$  and coefficients  $\mathbf{S} = [\mathbf{s}^{(1)} \dots \mathbf{s}^{(n)}]$  as well as the presence of the  $L_0$  norm  $\|\cdot\|_0$ , which both make the objective non-convex. Some approaches for solving Eq. (1) alternately optimize  $\mathbf{L}$  and  $\mathbf{S}$  until a local minima is reached.<sup>1</sup>

Like other approaches for dictionary learning, K-SVD alternates two optimization steps:

1. Optimize  $\mathbf{S}$  in Eq. (1) given the current  $\mathbf{L}$ .
2. For a particular dictionary element (i.e., the  $j$ th column of  $\mathbf{L}$ ), jointly optimize the element as well as its corresponding coefficient for each  $\mathbf{x}_i$  currently encoded by the element (i.e., the non-zero entries in the  $j$ th row of  $\mathbf{S}$ ).

We next describe each of these steps of K-SVD; the complete K-SVD algorithm is given as Algorithm 1.

**Step 1: Optimizing  $\mathbf{S}$**  Given a fixed  $\mathbf{L}$ , Eq. (1) decomposes into  $n$  independent optimization problems of the form:

$$\mathbf{s}^{(i)} \leftarrow \arg \min_{\mathbf{s}} \{ \|\mathbf{L}\mathbf{s} - \mathbf{x}_i\|_2^2 + \mu \|\mathbf{s}\|_0 \} . \quad (2)$$

Eq. (2) is known as the *sparse coding problem*, and can be solved (approximately) using numerous techniques (e.g., Matching Pursuit (MP), Orthogonal MP, or the Lasso).

**Step 2: Optimizing a Dictionary Element** This step updates a particular column  $\mathbf{l}_j$  of  $\mathbf{L}$  as well as the corresponding coefficients for data points that are encoded using  $\mathbf{l}_j$  (i.e., have a non-zero coefficient value). First, we form the matrix  $\mathbf{E}$  representing the residual for each data point given that  $\mathbf{l}_j$  is zeroed out. The  $i$ th column of  $\mathbf{E}$  is given by:

$$\mathbf{e}_i = \mathbf{x}_i - \sum_{r \neq j} s_r^{(i)} \mathbf{l}_r , \quad (3)$$

where  $s_r^{(i)}$  is the  $r$ th entry of  $\mathbf{s}^{(i)}$ . Next, we perform a singular value decomposition (SVD) on  $\mathbf{E}$ . The first left singular vector gives the updated value for  $\mathbf{l}_j$  and the corresponding right singular vector scaled by the corresponding singular value yields the updated coefficients (i.e., the  $j$ th row of  $\mathbf{S}$ ).

We would like both steps (1) and (2) to either maintain or improve the quality of our solution to Eq. (1). Unfortunately, using the SVD of  $\mathbf{E}$  will cause some coefficients in  $\mathbf{S}$

---

<sup>1</sup>Optimizing  $\mathbf{L}$  given a fixed  $\mathbf{S}$  is a convex optimization problem, whereas optimizing the columns of  $\mathbf{S}$  with fixed  $\mathbf{L}$ , while not convex, can be relaxed into a convex optimization problem by replacing the  $L_0$  norm with the  $L_1$  norm.

that were previously zero to become non-zero, eliminating the guarantee that the quality of our solution cannot become worse. To eliminate this possibility, we take the SVD of the subset  $\mathcal{A}$  of the columns of  $\mathbf{E}$  such  $m \in \mathcal{A} \Leftrightarrow s_j^{(m)} \neq 0$ :

$$(\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}) = \text{svd}(\mathbf{E}_{\mathcal{A}}) \quad (4)$$

$$\mathbf{l}_j \leftarrow \mathbf{u}_1 \quad \mathbf{s}_j^{(\mathcal{A})} \leftarrow \sigma_{1,1} \mathbf{v}_1 , \quad (5)$$

where  $\mathbf{E}_{\mathcal{A}}$  denotes the matrix formed from the subset of columns in  $\mathcal{A}$ , the singular values are assumed to all be positive (this is possible for any real matrix) and sorted in descending order, and  $\mathbf{s}_j^{(\mathcal{A})}$  denotes the vector formed from the columns in  $\mathcal{A}$  of the  $j$ th row of  $\mathbf{S}$ . It is well-known that this optimization procedure minimizes  $\|\mathbf{E}_{\mathcal{A}} - \mathbf{B}\|_2^2$  for all rank-1 matrices  $\mathbf{B} = \mathbf{l}_j \mathbf{s}_j^{(\mathcal{A})\top}$ . This implies that the resulting  $\mathbf{l}_j$  and  $\mathbf{s}_j^{(\mathcal{A})}$  also minimize  $\sum_{i=1}^n \|\mathbf{L}\mathbf{s}^{(i)} - \mathbf{x}_i\|_2^2$ , which shows that quality of the solution to Eq. (1) cannot have worsened.

## Multi-Task Learning Using K-SVD

This section extends K-SVD to batch MTL. The key step of our approach is to adapt K-SVD from the objective of learning a dictionary for sparse coding a set of input data points to learning a dictionary for sparse coding a set of parameter vectors for individual task models. We begin by describing the MTL setting, and then describe how to adapt K-SVD to this setting. We call our resulting algorithm *MTL-SVD*.

### Problem Setting

In the batch MTL setting, the agent simultaneously learns models for a set of supervised learning tasks  $\{\mathcal{Z}^{(1)}, \mathcal{Z}^{(2)}, \dots, \mathcal{Z}^{(T)}\}$ . Each task  $\mathcal{Z}^{(t)} = (\hat{f}^{(t)}, \mathbf{X}^{(t)}, \mathbf{y}^{(t)})$  is defined by a (hidden) mapping  $\hat{f}^{(t)} : \mathcal{X}^{(t)} \mapsto \mathcal{Y}^{(t)}$  from an instance space  $\mathcal{X}^{(t)} \subseteq \mathbb{R}^d$  to a set of labels  $\mathcal{Y}^{(t)}$  (typically  $\mathcal{Y}^{(t)} = \{-1, +1\}$  for classification tasks and  $\mathcal{Y}^{(t)} = \mathbb{R}$  for regression tasks). Task  $t$  has  $n_t$  training instances  $\mathbf{X}^{(t)} \in \mathbb{R}^{d \times n_t}$  with corresponding labels  $\mathbf{y}^{(t)} \in \mathcal{Y}^{(t)n_t}$  given by  $\hat{f}^{(t)}$ . We assume that the learner is given all labeled training data for all learning tasks in a single batch. The agent's goal is to construct task models  $f^{(1)}, \dots, f^{(T)}$  where each  $f^{(t)} : \mathbb{R}^d \mapsto \mathcal{Y}^{(t)}$  such that each  $f^{(t)}$  will approximate  $\hat{f}^{(t)}$  to enable the accurate prediction of labels for new instances.

### Model of Task Structure

Our model of latent task structure is based on the GO-MTL model proposed by Kumar & Daumé (2012), which allows for learning of overlap and grouping between tasks, and is closely related to the model used by Maurer et al. (2013). We assume a parametric framework in which each task model  $f^{(t)}(\mathbf{x}) = f(\mathbf{x}, \boldsymbol{\theta}^{(t)})$  is specified by a task-specific parameter vector  $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^d$ . To facilitate transfer between tasks, we maintain a library of  $k$  latent model components  $\mathbf{L} \in \mathbb{R}^{d \times k}$  that are shared between task models. Each  $\boldsymbol{\theta}^{(t)}$  can be represented as a linear combination of the columns of  $\mathbf{L}$  according to the coefficients  $\mathbf{s}^{(t)} \in \mathbb{R}^k$  (i.e.,  $\boldsymbol{\theta}^{(t)} = \mathbf{L}\mathbf{s}^{(t)}$ ). We encourage the  $\mathbf{s}^{(t)}$ 's to be sparse (i.e., use few latent compo-

nents) to ensure that each learned latent component captures a maximal reusable chunk of knowledge.

Given the labeled training data for each task, we optimize the task models to minimize the predictive loss over all tasks while encouraging the models to share structure through  $\mathbf{L}$ . This optimization problem is realized by:

$$e_T(\mathbf{L}) = \sum_{t=1}^T \min_{\mathbf{s}^{(t)}} \left\{ \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L} \left( f \left( \mathbf{x}_i^{(t)}; \mathbf{L}\mathbf{s}^{(t)} \right), y_i^{(t)} \right) + \lambda \|\mathbf{L}\mathbf{s}^{(t)}\|_2^2 + \mu \|\mathbf{s}^{(t)}\|_0 \right\}, \quad (6)$$

where  $(\mathbf{x}_i^{(t)}, y_i^{(t)})$  is the  $i$ th labeled training instance for task  $t$ ,  $\mathcal{L}$  is a known loss function for fitting the task models, and  $\lambda$  and  $\mu$  are non-negative constants that define the amount of ridge-regression on the models and the penalty for non-sparse solutions, respectively.

In earlier work (Ruvolo and Eaton 2013), we proposed a method for optimizing Eq. (6) efficiently by recasting it as a problem of learning a dictionary for sparse-coding task models that were trained using single-task learning. This approach takes the second-order Taylor expansion of Eq. (6) about  $\mathbf{L}\mathbf{s}^{(t)} = \boldsymbol{\theta}^{(t)}$ , where  $\boldsymbol{\theta}^{(t)}$  is an optimal predictor for task  $t$  learned on only that task’s training data. We apply this technique to yield a simplified form of our objective:

$$g_T(\mathbf{L}) = \sum_{t=1}^T \min_{\mathbf{s}^{(t)}} \left\{ \frac{1}{n_t} \|\boldsymbol{\theta}^{(t)} - \mathbf{L}\mathbf{s}^{(t)}\|_{\mathbf{D}^{(t)}}^2 + \mu \|\mathbf{s}^{(t)}\|_0 \right\} \quad (7)$$

where

$$\mathbf{D}^{(t)} = \lambda \mathbf{I} + \nabla_{\boldsymbol{\theta}^{(t)}}^2 \frac{1}{2n_t} \sum_{i=1}^{n_t} \mathcal{L} \left( f \left( \mathbf{x}_i^{(t)}; \boldsymbol{\theta} \right), y_i^{(t)} \right) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$$

$$\boldsymbol{\theta}^{(t)} = \arg \min_{\boldsymbol{\theta}} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L} \left( f \left( \mathbf{x}_i^{(t)}; \boldsymbol{\theta} \right), y_i^{(t)} \right) + \lambda \|\boldsymbol{\theta}\|_2^2,$$

and  $\|\mathbf{v}\|_{\mathbf{A}}^2 = \mathbf{v}^\top \mathbf{A} \mathbf{v}$ . Note that Eq. (7) is identical to Eq. (1) except that we use the norm  $\|\cdot\|_{\mathbf{D}^{(t)}}$  as opposed to  $\|\cdot\|_2^2$  in order to incorporate the 2nd-order information in  $\mathbf{D}^{(t)}$  about how sensitive the loss function is to deviations in various directions of the reconstructed model  $\mathbf{L}\mathbf{s}^{(t)}$  from the single-task model. Therefore, we could ignore the difference in the objective functions and apply K-SVD unmodified to the  $\boldsymbol{\theta}^{(t)}$ ’s to arrive at a sensible MTL algorithm. However, our goal will be to improve upon this naïve approach.

Our principal modification to the original K-SVD is to replace the SVD in Eq. (4) with a generalized SVD:<sup>2</sup>

$$(\mathbf{U}, \boldsymbol{\Sigma}, \mathbf{V}) = \text{gsvd}(\mathbf{E}_{\mathcal{A}}, \mathbf{M}, \mathbf{W}), \quad (8)$$

where  $\mathbf{M}$  is a symmetric positive semidefinite (PSD) matrix in  $\mathbb{R}^{d \times d}$  and  $\mathbf{W}$  is a symmetric PSD matrix in  $\mathbb{R}^{T \times T}$ . Later we will discuss how to choose  $\mathbf{M}$  and  $\mathbf{W}$ ; for now we assume they are given. Using the GSVD instead of the SVD properly takes 2nd order information into account during the optimization. The GSVD step in Eq. (8) along with the up-

<sup>2</sup>Note that computing the GSVD is not more computationally expensive than computing the SVD, with the exception that we must first compute the matrix square roots of  $\mathbf{M}$  and  $\mathbf{W}$ .

---

## Algorithm 2 MTL-SVD

---

**input** training data  $(\mathbf{X}^{(1)}, \mathbf{y}^{(1)}), \dots, (\mathbf{X}^{(T)}, \mathbf{y}^{(T)})$ ;  
dictionary size  $k$ ; single-task learner  $\text{STL}()$   
init  $\mathbf{L}$  using random column vectors of unit length  
**for**  $t \in \{1, \dots, T\}$ ,  $(\boldsymbol{\theta}^{(t)}, \mathbf{D}^{(t)}) \leftarrow \text{STL}(\mathbf{X}^{(t)}, \mathbf{y}^{(t)})$   
**loop** until convergence **do**  
**for**  $t \in \{1, \dots, T\}$ , perform update in Eq. (11)  
**for**  $j \in \{1, \dots, k\}$ , perform updates in Eqs. (8) and (5)  
**end loop**  
**return**  $\mathbf{L}$

---

dates in (5) yield the values of  $\mathbf{l}_j$  and  $\mathbf{s}_j^{(\mathcal{A})}$  that minimize:

$$\sum_{t_1=1}^{|\mathcal{A}|} \sum_{t_2=1}^{|\mathcal{A}|} w_{t_1, t_2} \left( \mathbf{e}_{\mathcal{A}_{t_1}} - \mathbf{l}_j \mathbf{s}_j^{(\mathcal{A}_{t_1})} \right)^\top \mathbf{M} \left( \mathbf{e}_{\mathcal{A}_{t_2}} - \mathbf{l}_j \mathbf{s}_j^{(\mathcal{A}_{t_2})} \right). \quad (9)$$

This formulation adjusts a particular column of  $\mathbf{L}$  and all corresponding non-zero entries of the  $\mathbf{s}^{(t)}$ ’s. Additionally, instead of using squared-loss, we are free to choose  $\mathbf{M}$  and  $\mathbf{W}$  in a manner to weight certain errors, either between different features or between different tasks, more heavily. We can view  $\mathbf{M}$  as analogous to  $\mathbf{D}^{(t)}$  in Eq. (7), except that  $\mathbf{M}$  must be shared among all tasks that use a particular latent model component and cannot be set independently for each task. Additionally, the off-diagonal terms of the matrix  $\mathbf{W}$  correspond to extra terms that do not appear in Eq. (7); these additional (optional) terms can be used to enforce task relationship biases. Essentially, we can view  $\mathbf{M}$  as a feature relationship matrix and  $\mathbf{W}$  as a task relationship matrix.

While there are many ways to choose  $\mathbf{M}$  and  $\mathbf{W}$  (we leave an exploration of these other options as future work), here we begin by setting all off-diagonal elements of  $\mathbf{W}$  to 0 in order to ignore task relationship biases. We set  $\mathbf{M}$  as a consensus of each of the  $\mathbf{D}^{(t)}$ ’s by computing their mean:  $\mathbf{M} = \frac{1}{|\mathcal{A}|} \sum_{t \in \mathcal{A}} \mathbf{D}^{(t)}$ . Next, we set the value of the diagonal entries of  $\mathbf{W}$  such that for each task we rescale  $\mathbf{M}$  to closely approximate the corresponding  $\mathbf{D}^{(t)}$ :

$$w_{t,t} = \frac{\mathbf{1}^\top \mathbf{D}^{(\mathcal{A}_t)} \mathbf{1}}{\sum_{t' \in \mathcal{A}} \mathbf{1}^\top \mathbf{D}^{(t')} \mathbf{1}}, \quad (10)$$

where  $\mathbf{1}$  is the vector of all 1’s of the appropriate dimensionality (i.e.,  $\mathbf{1}^\top \mathbf{D}^{(t)} \mathbf{1}$  sums all entries of  $\mathbf{D}^{(t)}$ ).

To be suitable for use with GSVD, both  $\mathbf{M}$  and  $\mathbf{W}$  must be PSD.  $\mathbf{M}$  is guaranteed to be PSD since it is an average of PSD matrices (the individual  $\mathbf{D}^{(t)}$ ’s).  $\mathbf{W}$  is PSD since it is a diagonal matrix with non-negative diagonal entries (the diagonal entries are non-negative since they are each a sum of the elements of a PSD matrix, which must be positive).

Now that we have presented the modifications related to the dictionary update step of K-SVD, we write the updated computation of  $\mathbf{s}^{(t)}$  (originally defined in Eq. (2)) as:

$$\mathbf{s}^{(t)} \leftarrow \arg \min_{\mathbf{s}} \left\{ \|\mathbf{L}\mathbf{s} - \mathbf{x}_i\|_{\mathbf{D}^{(t)}}^2 + \mu \|\mathbf{s}\|_0 \right\}, \quad (11)$$

which we solve using the Lasso (Tibshirani 1996). The complete MTL-SVD algorithm is given in Algorithm 2.

## Computational Complexity

First, MTL-SVD uses a single-task learner to compute the tuples  $(\theta^{(t)}, \mathbf{D}^{(t)})$  for all tasks. We use the function  $\xi(d, n_t)$  to represent the complexity of this base learner on a problem of dimensionality  $d$  with  $n_t$  training instances. Next, the update of  $\mathbf{s}^{(t)}$  given in Eq. (11) can be solved using the Lasso in time  $O(d^3 + d^2k + k^2d)$ . Each update of a particular basis component and the corresponding entries in  $\mathbf{S}$  (Eqs. (8) and (5)) involves computing the square root of a  $d \times d$  matrix and taking the SVD of a  $d \times r$  matrix (where  $r$  is the maximum number of tasks that use a single basis vector). These two steps yield a complexity of  $O(d^3 + r^2d)$ . We must repeat the updates in Eqs. (8) and (5) for each entry of  $\mathbf{s}^{(t)}$ . After an initial startup cost of  $O(\sum_{t=1}^T \xi(d, n_t))$  to compute the single-task models, the per-iteration complexity of MTL-SVD is  $O(d^2k + k^2d + kd^3 + kr^2d)$ . In comparison, GO-MTL (Kumar and Daumé 2012), a current MTL algorithm, has a per-iteration complexity of  $O(d^3k^3)$ .

## Lifelong Learning Using K-SVD

Next, we adapt MTL-SVD to a lifelong learning setting in which tasks arrive consecutively. We call our new algorithm *ELLA-SVD*, since it is an alternate formulation of ELLA.

### The Lifelong Learning Problem

We now consider a variant of the MTL setting in which the agent faces a *series* of supervised learning tasks  $\mathcal{Z}^{(1)}, \mathcal{Z}^{(2)}, \dots, \mathcal{Z}^{(T_{\max})}$  (Figure 1). In contrast to the MTL setting, we assume that the learner receives the tasks consecutively and *a priori* does not know the total number of tasks  $T_{\max}$ , the distribution of these tasks, or their order.

Each time step, the agent receives a batch of labeled training data for some task  $t$ , either a new task or as additional training data for a previously learned task. In the lifelong learning setting, we use  $T$  to denote the number of tasks the agent has encountered so far, with  $0 \leq T \leq T_{\max}$ . After receiving each batch, the agent may be asked to make predictions on data instances of any previous task. Its goal is to construct task models  $f^{(1)}, \dots, f^{(T)}$  where each  $f^{(t)} : \mathbb{R}^d \mapsto \mathcal{Y}^{(t)}$  such that: *a)* each  $f^{(t)}$  will approximate  $\hat{f}^{(t)}$  to enable the accurate prediction of labels for new instances, *b)* each  $f^{(t)}$  can be rapidly updated as the agent encounters additional training data for known tasks, and *c)* new  $f^{(t)}$ 's can be added efficiently as the agent encounters new tasks. We assume that the total number of tasks  $T_{\max}$  and the total amount of data instances  $\sum_{t=1}^{T_{\max}} n_t$  will be large, and so a lifelong learner must have a computational complexity to update the task models that scales favorably with both quantities. Figure 1 illustrates the lifelong learning process.

### Adapting MTL-SVD for Lifelong Learning

MTL-SVD is inapplicable to the lifelong learning setting, since as each new task is presented to the learner, MTL-SVD would need to be rerun until convergence—a process could take arbitrarily long. As in ELLA, we employ an update strategy that, whereupon receiving data for task  $t$ , only the corresponding entry  $\mathbf{s}^{(t)}$  is updated. Following this

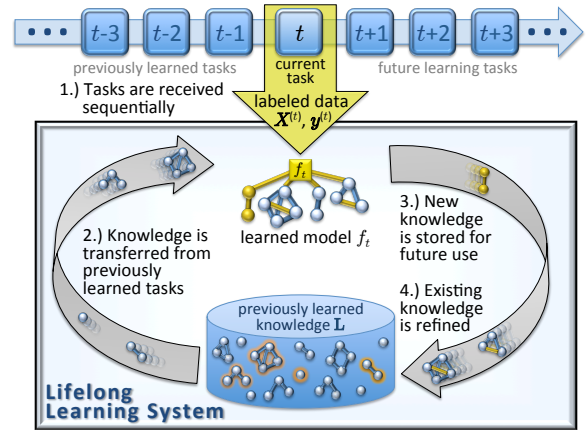


Figure 1: An illustration of the lifelong learning process.

update,  $\mathbf{L}$  is then adapted to optimize performance across *all* known tasks. However, in contrast to MTL-SVD these two steps are not repeated until convergence. Instead, they are each performed once for each new batch of training data. These simplifications incur only a small performance penalty w.r.t. Eq. (7) as shown in (Ruvolo and Eaton 2013). This allows us to modify MTL-SVD for lifelong learning:

1. When receiving training data for task  $t$ , only  $\mathbf{s}^{(t)}$  is updated via Eq. (11).
2. The updates in Eqs. (8) and (5) are performed only on the subset of columns of  $\mathbf{L}$  corresponding to the non-zero entries of  $\mathbf{s}^{(t)}$  (i.e., we only update the basis vectors that are used to construct the model for the current task).
3. The two steps of the K-SVD algorithm are not repeated until convergence, but instead each performed only once per new batch of training data.

These modifications provide large gains in efficiency for three reasons. First, we do not need to update the  $\mathbf{s}^{(t)}$ 's for all previous tasks (which becomes expensive as  $T$  grows large). Second, the GSVD update step only has to be performed a number of times equal to the number of non-zero entries in  $\mathbf{s}^{(t)}$  (which is a large savings when  $\mathbf{s}^{(t)}$  is sparse). Third, we do not iterate updates of  $\mathbf{s}^{(t)}$  and  $\mathbf{L}$  until convergence; instead we only need to run each step once.

## Computational Complexity

As before, we use  $\xi(d, n_t)$  to represent the complexity of the base learner on a problem of dimensionality  $d$  with  $n_t$  training instances. The update of  $\mathbf{s}^{(t)}$  via Eq. (11) can be solved using the Lasso in time  $O(d^3 + d^2k + k^2d)$ . Each update of a particular latent basis component and the corresponding entries in  $\mathbf{S}$  (Eqs. (8) and (5)) involves computing the square root of a  $d \times d$  matrix and taking the SVD of a  $d \times r$  matrix (where  $r$  is the number of tasks that utilize the basis component being updated). These two steps yield a complexity of  $O(d^3 + r^2d)$ . We must repeat Eqs. (8) and (5) for each non-zero entry of  $\mathbf{s}^{(t)}$ . Let  $q$  indicate the number of such non-zero entries. Then, the total complexity for incorporating a new batch of training data is  $O(\xi(d, n_t) + d^2k + k^2d + qd^3 + qr^2d)$ . In comparison, ELLA has a complexity for incorporating a new batch of training data of  $O(\xi(d, n_t) + d^3k^2)$ , which is significantly less efficient than ELLA-SVD.

## Extensions

In this section, we modify the original ELLA to include an incremental update, which reduces its complexity to more closely match ELLA-SVD. We also show that this incremental update can be integrated into ELLA-SVD to yield a hybrid approach that has strong empirical performance.

### ELLA with an Incremental Update

ELLA-SVD has a much lower computational complexity than the original ELLA. To facilitate better empirical comparison between the algorithms, we explore a modified version of ELLA, which we call *ELLA Incremental*, that closely matches the complexity of ELLA-SVD. We incorporate an incremental update into ELLA that updates each of the columns of  $\mathbf{L}$  independently when data for a new task is received. Further, only the columns of  $\mathbf{L}$  that are non-zero in the current  $\mathbf{s}^{(t)}$  are updated and each column is updated at most once per batch of training data. The cost incurred for this more efficient update to  $\mathbf{L}$  is that ELLA Incremental is not guaranteed to achieve the globally optimal updated value of  $\mathbf{L}$  (given a fixed  $\mathbf{S}$ ) as can be guaranteed in ELLA.

### Hybrid Approach

Our proposed ELLA-SVD can be combined with ELLA Incremental into a hybrid approach. In this hybrid, we first perform one update step of ELLA-SVD and then one update step of ELLA Incremental. To guard against the two updates interfering with each other, if the ELLA-SVD step degrades the quality of our solution to Eq. (7), then we only use the ELLA Incremental update for that iteration. We call this hybrid approach *ELLA Dual Update*. This approach maintains the strong advantage in computational complexity over ELLA of both ELLA-SVD and ELLA Incremental. Additionally, as we show in the next section, the hybrid approach performs much better than ELLA-SVD in situations where the assumptions of the ELLA-SVD algorithm are inaccurate.

## Experiments

We evaluated four different algorithms for lifelong learning: *a*) ELLA, as defined in (Ruvolo and Eaton 2013), *b*) ELLA-SVD, *c*) ELLA Incremental, and *d*) ELLA Dual Update. We are primarily interested in developing a technique that closely approximates the accuracy of the ELLA algorithm, but with better computational complexity. As the base learner  $f(\cdot)$ , we used linear or logistic regression. Each algorithm was tested on four multi-task data sets:

**Synthetic Regression Tasks** We created a set of  $T_{\max} = 100$  random tasks with  $d = 13$  features and  $n_t = 100$  instances per task. The task parameter vectors  $\theta^{(t)}$  were generated as a linear combination of  $k = 6$  random latent components in  $\mathbb{R}^{12}$ . The vectors  $\mathbf{s}^{(t)}$  had a sparsity level of 0.5 (i.e., half the latent components were used to construct each  $\theta^{(t)}$ ). The training data  $\mathbf{X}^{(t)}$  was generated from a standard normal distribution. The training labels for each task were given as  $\mathbf{y}^{(t)} = \mathbf{X}^{(t)\top} \theta^{(t)} + \epsilon$ , where each element of  $\epsilon$  is independent noise generated from a standard normal distribution. A bias term was added as the 13th feature.

**Student Exam Score Prediction** The London Schools data set consists of exam scores from 15,362 students in 139 schools. We treat the data from each school as a separate task. The goal is to predict the exam score of each student. We use the same feature encoding as used by Kumar & Daumé (2012), where four school-specific and three student-specific categorical variables are encoded as a collection of binary features. We use the exam year and a bias term as additional features, giving each data instance  $d = 27$  features.

**Land Mine Detection** In the land mine data set (Xue et al. 2007), the goal is to detect whether or not a land mine is present in an area based on radar images. The input consist of four-moment based features, three correlation-based features, one energy-ratio feature, one spatial variance feature, and a bias term. The data set contains a total of 14,820 data instances divided into 29 different geographical regions. We treat each geographical region as a different task.

**Facial Expression Recognition** This data set is from a recent facial expression recognition challenge (Valstar et al. 2011). Each task involves recognizing one of three facial action units (#5: upper lid raiser, #10: upper lip raiser, and #12: lip corner pull) from an image of one of seven subjects' faces. There are a total of 21 tasks, each with 450–999 images. We use the same feature encoding as Ruvolo & Eaton (2013), using a multi-scale Gabor pyramid to extract 2,880 Gabor features for each image, then reducing them to 100 dimensions using PCA, and adding a bias term.

### Evaluation Procedure

Following the lifelong learning framework, each task was presented consecutively to each algorithm as a single batch of data that contained all training instances for that task. For each task, the training data was divided into both a training and a held-out test set (with 50% of the data designated for each). The task order was also randomized.

We generated learning curves for each method on each data set by averaging the performance over all tasks. We used the area under the ROC (AROC) as our metric for accuracy on classification tasks and negative root mean-squared (-rMSE) as our metric for regression tasks. We chose AROC rather than accuracy due to the skewed class distribution for the land mine detection and facial expression recognition tasks. To evaluate the progress of learning performance as new tasks were presented, we measured the average task performance across all learned tasks. For tasks that were already learned this was straightforward: we simply evaluated how well the currently learned model for that task generalized to the corresponding test set. For tasks that had yet to be learned, we created a model for the task by fitting the currently learned basis  $\mathbf{L}$  to the training data for the task (but did not modify  $\mathbf{L}$ ). Therefore, the learning curves evaluate both how well the current basis models previously learned tasks as well as how well it generalizes to *unseen* tasks.

The  $\lambda$  and  $k$  parameters were independently selected for each method via a gridsearch over all combinations of  $\lambda \in \{e^{-5}, \dots, e^5\}$  and  $k \in \{1, \dots, 10\}$ ;  $\mu$  was set to  $e^{-5}$  for all algorithms. In order to ablate performance variation due to online adjustment of the hyperparameters (an open problem in online MTL), the values of these parameters were chosen

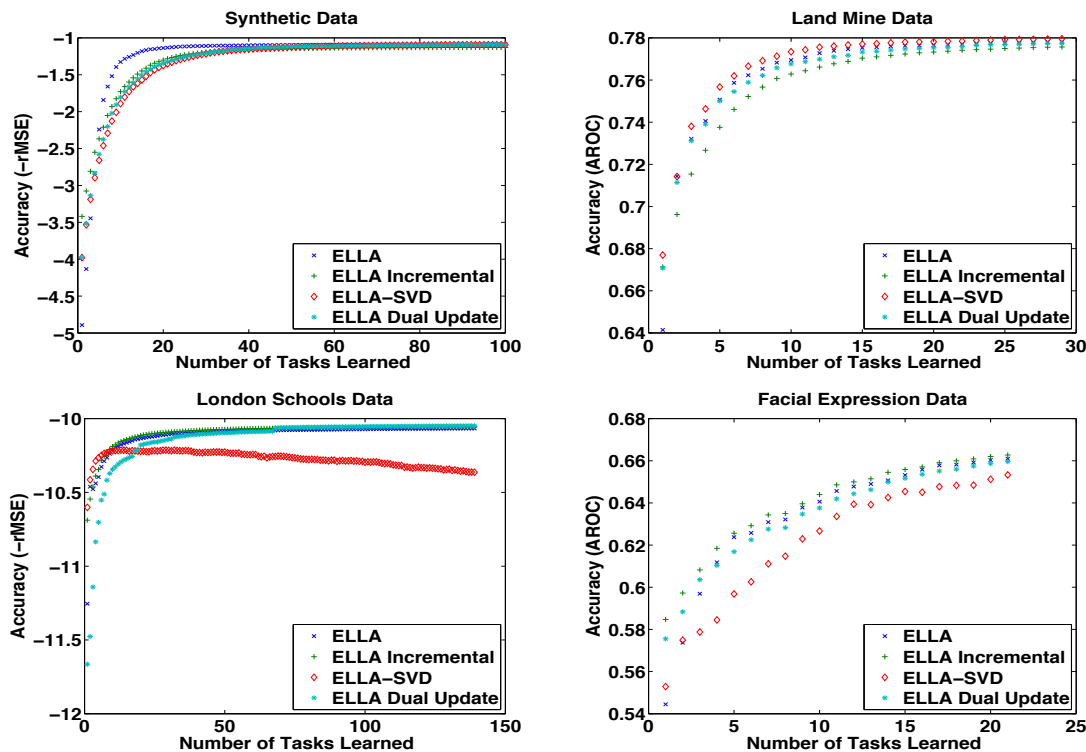


Figure 2: Lifelong learning results, averaged over 100 trials. Performance was evaluated on *all* tasks after learning each new task.

to maximize the average performance when learning using all tasks. Although this procedure inflates performance relative to fitting the parameters online using a validation set, it allows us to better compare the relative performance levels of the different algorithms (which is our principal focus).

## Results

The results of our evaluation are given in Figure 2. The proposed ELLA-SVD approach is better than all other methods on the land mine task. Specifically, the approach performs much better than the other efficient update approach, ELLA Incremental. On the synthetic regression tasks, the original ELLA method is clearly the best, with the ELLA-SVD and ELLA Incremental approaches lagging behind.

In contrast to the strong performance of ELLA-SVD on land mine and the synthetic tasks, ELLA-SVD does not perform well on either facial expression recognition or student exam score prediction. In particular, the performance of ELLA-SVD on student exam score prediction actually *declines* as it learns more tasks. Further investigation revealed that the cause of this problem was that the matrix  $\mathbf{M}$  formed as a consensus of the  $\mathbf{D}^{(t)}$ 's (which is required for Eq. (8)) is a poor approximation to the true objective function we would like to minimize (Eq. (7)). The primary reason for this poor approximation is that the input distributions for each task (i.e., each school) are quite different due to the school-specific features of each instance. In this case, the ELLA-SVD updates turn out to be counter-productive.

We proposed the ELLA Dual Update approach in order to get the best of both worlds. That is, we seek to achieve the

high performance of ELLA-SVD on tasks where it is appropriate for application (e.g., for land mine detection), and to fall back to ELLA Incremental when ELLA-SVD performs poorly (e.g., for the London schools data). The results for the Dual Update version shown in Figure 2 suggest that this hybrid approach is successful. The performance of ELLA Dual Update clusters tightly with the best performing algorithm for each learning problem (with the exception of the synthetic regression tasks, for which none of the more-efficient approaches does as well as the original ELLA).

## Conclusion

We explored the use of the K-SVD algorithm (Aharon et al. 2006) in the lifelong machine learning setting. Adapting K-SVD to the lifelong learning setting required several key innovations including: *a*) replacing the SVD step in the original algorithm with a generalized SVD, and *b*) selectively updating components of the model as new task data is presented. We showed that ELLA-SVD performs well on problems where the input distributions of the data are similar.

For domains where the input distributions are *not* similar, we showed that a hybrid approach (in which we interleave the ELLA-SVD update with another efficient update step called ELLA Incremental) performs robustly. In future work, we will conduct experiments to better understand the tradeoffs between ELLA-SVD and ELLA Incremental. Additionally, we plan to test our more-efficient versions of ELLA in settings where applying the original ELLA is computationally intractable (e.g., when  $k$  and  $d$  are large).

## Acknowledgements

This research was supported by ONR grant #N00014-11-1-0139 and AFOSR grant #FA8750-14-1-0069. We would like to thank Terran Lane, Diane Oyen, and the anonymous reviewers for their helpful feedback.

## References

- Aharon, M.; Elad, M.; and Bruckstein, A. 2006. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing* 54(11):4311–4322.
- Caruana, R. 1997. Multitask learning. *Machine Learning* 28(1):41–75.
- Kang, Z.; Grauman, K.; and Sha, F. 2011. Learning with whom to share in multi-task feature learning. In *Proceedings of the 28th International Conference on Machine Learning*.
- Kumar, A., and Daumé III, H. 2012. Learning task grouping and overlap in multi-task learning. In *Proceedings of the 29th International Conference on Machine Learning*.
- Maurer, A.; Pontil, M.; and Romera-Paredes, B. 2013. Sparse coding for multitask and transfer learning. In *Proceedings of the 30th International Conference on Machine Learning*.
- Ruvolo, P., and Eaton, E. 2013. ELLA: An efficient lifelong learning algorithm. In *Proceedings of the 30th International Conference on Machine Learning*.
- Saha, A.; Rai, P.; Daumé III, H.; and Venkatasubramanian, S. 2011. Online learning of multiple tasks and their relationships. In *Proceedings of the 14th Conference on Artificial Intelligence and Statistics (AISTATS)*.
- Tibshirani, R. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* 58(1):267–288.
- Valstar, M.; Jiang, B.; Mehu, M.; Pantic, M.; and Scherer, K. 2011. The first facial expression recognition and analysis challenge. In *Proceedings of the IEEE International Conference on Automatic Face & Gesture Recognition*, 921–926.
- Xue, Y.; Liao, X.; Carin, L.; and Krishnapuram, B. 2007. Multi-task learning for classification with Dirichlet process priors. *Journal of Machine Learning Research* 8:35–63.
- Zhang, J.; Ghahramani, Z.; and Yang, Y. 2008. Flexible latent variable models for multi-task learning. *Machine Learning* 73(3):221–242.