GILDA++: Grassmann Incremental Linear Discriminant Analysis

Navya Nagananda
Rochester Institute of Technology
nn3264@rit.edu

Andreas Savakis
Rochester Institute of Technology
andreas.savakis@rit.edu

Abstract

Linear Discriminant Analysis (LDA) is an important supervised dimensionality reduction method. Traditional LDA makes use of the eigenvalue decomposition of the scatter matrices based on the entire dataset. However, in some settings the whole dataset may not be available at once. Our approach considers an incremental LDA framework where the model receives training data in the form of chunks for subsequent analysis. We propose the Grassmann-Incremental Linear Discriminant Analysis (GILDA++) using the proxy matrix optimization method (PMO). The PMO method does not directly optimize a matrix on the manifold, but uses an auxiliary or proxy matrix in ambient space which is retracted to the closest location on the manifold along the loss minimizing geodesic. PMO makes use of an LDA objective by incrementally updating the scatter matrices to handle chunks of data. It makes use of automatic differentiation and stochastic gradient descent to find the lower dimensional LDA projection matrix. GILDA++ is able to handle chunk data, where each chunk has new samples from existing classes or novel classes. Our experiments demonstrate that GILDA++ outperforms the prevailing incremental LDA methods in various datasets.

1. Introduction

Linear Dimensionality Reduction (LDR) is an important tool for the machine learning community, finding applications in face recognition [2, 19], pattern recognition [3, 8], and data visualization [20]. Linear Discriminant Analysis (LDA) is a supervised LDR method, that has been mainly used for classification [3]. Fishers LDA [7, 22] works by minimizing the within class scatter and maximizing the between class scatter of the training data. In order to obtain the lower dimensional projections, a lot of data has to be provided to obtain scatter matrices that are good representations of the data. However, this is not always possible, in cases where the full dataset is not available and training samples are obtained incrementally in chunks of variable size, as illustrated in Figure 1. In many real world scenarios, the complete set of training samples is not available in advance, and data is provided in the form of chunks which may have new classes and/or an uneven distribution of samples from the existing classes [25]. An incremental LDA framework has to be able to handle both training scenarios seamlessly, i.e., training on the full dataset or incrementally. Traditionally, LDA methods use an eigenvalue solution to find the lower dimensional projection matrix, which is not always the most suitable solution to the problem. In [4], LDR is cast as a matrix optimization problem over the Grassmann manifold. This formulation leads to a generic framework for LDR based on stochastic gradient descent (SGD) that can be utilized with various optimization techniques. In the case of LDA, the orthogonality requirement constrains the solution to lie on a Grassmann manifold, making a Grassmann manifold-based formulation well-suited for optimization.

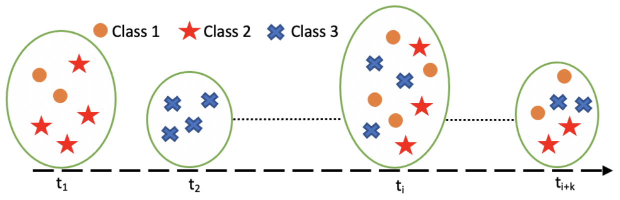

Figure 1. Illustration of data arriving in chunks of variable size for incremental learning.

Incremental LDA methods have been proposed for streaming data (one sample at a time) and chunk incoming data. One of the ways that incremental LDA can be done is by updating the scatter matrices that contribute to the objective function [25]. However, after the updating the scatter matrices, the LDA solution is obtained by the traditional eigenvalue solution which is time consuming and inefficient [4]. Other techniques work on new optimization strategies to improve the computational speed [27]. Methods such as [13, 27, 18] utilize QR decomposition to update the within-class and between-class scatter matrices. More recent methods try to reduce the matrix size for decomposition. LS-ILDA [17] incrementally updates the least square solution by converting LDA into a multivariate linear regression problem. In [14], sufficient-spanning sets are used to update the between-class and within-class scatter matrices by updating the eigenvectors at each step and removing the minor components. This method, however, trades off between accuracy and complexity. The LDA/GSVD [12] approach is used in cases where the data dimension exceeds the number of datapoints, such that the scatter matrices can be singular and hence the inverse is not computable. LDA/GSVD makes use of the pseudo-inverse and the generalized singular value decomposition to arrive at the LDA projection matrix. GSVD-ILDA [28] is the incremental version of [12] which incrementally learns a subspace instead of recomputing it from scratch. It involves updating the eigenvectors of the data centered matrix by removing the minor components. It can however be tricky to determine which minor components are to be removed, thus, there is risk of deteriorating performance.

We propose the Grassmann-Incremental Linear Discriminant Analysis (GILDA++) approach based on proxy matrix optimization (PMO). PMO makes use of a proxy matrix in ambient space instead of directly optimizing on the manifold. The key difference between PMO and the two-step method [4] is that PMO integrates the manifold retraction step into the optimization unlike the two-step process that performs retraction after optimization. Therefore, the optimizer in PMO is less constrained than the two-step process. The Grassmann manifold [1, 9, 26] has been used to arrive at the projection matrix for LDA [4, 21], but incremental approaches have not yet been explored. To the best of our knowledge, our GILDA++ method is the first to integrate the incremental version of the LDA into a manifold optimization problem. As chunks of data are received, incrementally update the scatter matrices using update rules as in [25], and then then perform proxy matrix optimization on the Grassmann manifold to find the lower dimensional projection for LDA. The main contributions of this paper are the following:

1. We propose GILDA++, the first method to utilize the proxy matrix optimization method on the Grassmann manifold to perform incremental LDA. PMO is less constrained than the existing two-step optimization method and allows for larger steps in ambient space and thus faster convergence.

2. Incremental LDA is cast as an objective function which does not require analytical computations for a closed form solution. A single framework based on stochastic gradient descent can handle the addition of new samples and/or new classes. This makes GILDA++ more flexible, due to the ease of replacing the objective function for a given task within a larger framework.

3. The use of SGD and backpropagation allows easy integration of LDA into a neural network, as the scatter matrices are updated based on the incoming data batches for training.

The rest of this paper is organized as follows. Section 2 presents the equations used for incremental chunk LDA. Section 3 introduces the definitions of some of the concepts used in this work. Section 4 describes the method and the algorithm of GILDA++. Section 5 discusses the experiments performed and the results along with the comparison to existing incremental LDA methods. Section 6 provides a discussion on the computational complexity and convergence of GILDA++. We finally conclude in Section 7.

2. Background

Assume that model has N training samples and let $\mathbf{X} = [x_1, x_2, \dots, x_N] \in \mathbb{R}^{D\times N}$ be the data matrix which is D dimensional and has M classes. LDA finds a linear projection of the data by means of a linear transformation $\mathbf{R}$ over $\mathbf{X}$ that minimizes the ratio the trace of the between-class scatter matrix ($\mathbf{S_b}$) and the within-class scatter matrix ($\mathbf{S_w}$) that are defined as:

$$\mathbf{S_w} = \sum_{c=1}^{M} \mathbf{\Sigma_c} = \sum_{c=1}^{M} \sum_{\mathbf{x}\in\mathbf{x_c}} (\mathbf{x} - \bar{\mathbf{x}}_\mathbf{c})(\mathbf{x} - \bar{\mathbf{x}}_\mathbf{c})^T \tag{1}$$

and

$$\mathbf{S_b} = \sum_{c=1}^{M} n_c(\bar{\mathbf{x}}_\mathbf{c} - \bar{\mathbf{x}})(\bar{\mathbf{x}}_\mathbf{c} - \bar{\mathbf{x}})^T \tag{2}$$

where n_c is the number of samples in each class c; the total number of samples is $N = \sum_{c=1}^{M} n_c$; $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x_i}$ is the mean of the data sample $\mathbf{X}$; and $\bar{\mathbf{x}}_\mathbf{c}$ is the mean vector of the samples that belong to the class c.

In this work, the LDA projection matrix $\mathbf{R} \in \mathbb{O}^{D\times d}$ in a d-dimensional space of lower dimension $d < D$, is obtained by minimizing the following objective,

$$f = -\frac{trace(\mathbf{R}^T\mathbf{S_b}\mathbf{R})}{trace(\mathbf{R}^T\mathbf{S_w}\mathbf{R})}. \tag{3}$$

The projection matrix $\mathbf{R}$ is orthogonal under this objective function and can be modeled with a Grassmann manifold formulation.

Typically LDA is performed by calculating the eigendecomposition of $\mathbf{S_w}^{-1}\mathbf{S_b}$ and taking the eigenvectors corresponding to the d largest eigenvalues as the projection matrix $\mathbf{R} \in \mathbb{R}^{D\times d}$. This discriminant eigenspace can be represented by $\Omega = (\mathbf{S_w}, \mathbf{S_b}, \bar{\mathbf{x}}, N)$. In the case of traditional LDA, this eigenspace is created by taking the entire dataset. However, this does not work in cases when the data is presented incrementally in a chunk or streaming fashion. With the arrival of new samples, the total mean, class mean and the scatter matrices are updated, which in-turn updates the entire discriminant eigenspace, Ω.

2.1. Chunk Incremental LDA

When performing incremental LDA, new samples are provided to the model in chunks, $Y_0, Y_1, ...$ until all the data have been received. At each time step, N samples have already been seen, and a chunk has L_k number of samples such that, $Y_i = \{\mathbf{y_1}, ..., \mathbf{y_{L_k}}\}$ and $L_k \geq 1$. The incremental approach updates the model by incorporating the new set of observations $(\mathbf{S_{w_y}}, \mathbf{S_{b_y}}, \bar{\mathbf{y}}, L_k)$ and finding the new eigenspace, $\phi = (\mathbf{S_w}', \mathbf{S_b}', \bar{\mathbf{x}}', N + L_k)$. There are two cases considered here: (a) the new data belong to known classes, and (b) the new data belong to a new class. In the fist case, l_c of the L_k new samples belong to a class c that now has number of samples $n'_c = n_c + l_c$, and $N + L_k = \sum_{c=1}^{M} n'_c = \sum_{c=1}^{M}(n_c + l_c)$. The updated class mean is $\bar{x}'_c = \frac{1}{n_c+l_c}(n_c\bar{x}_c + l_c\bar{y}_c)$, where $\bar{y}_c$ is the mean of the new samples in class c. The updated total mean is:

$$\bar{x}' = \frac{1}{N + L_k}(N\bar{x} + L_k\bar{y}) \tag{4}$$

where, $\bar{y} = \frac{1}{L_k}\sum_{j=1}^{L_k} y_j$. The updated between-class scatter matrix is [25],

$$\mathbf{S'_b} = \sum_{c=1}^{M} n'_c(\bar{x_c}' - \bar{x}')(\bar{x_c}' - \bar{x}')^T. \tag{5}$$

The updated within-class scatter matrix is [25],

$$\mathbf{S'_w} = \sum_{c=1}^{M} \mathbf{\Sigma'_c} \tag{6}$$

$$\mathbf{\Sigma'_c} = \Sigma_c + \frac{n_c l_c^2}{(n_c + l_c)^2}\mathbf{D_c} + \frac{n_c^2}{(n_c + l_c)^2}\mathbf{E_c} + \frac{l_c(l_c + 2n_c)}{(n_c + l_c)^2}\mathbf{F_c} \tag{7}$$

where,

$$\mathbf{D_c} = (\bar{y}_c - \bar{x}_c)(\bar{y}_c - \bar{x}_c)^T, \tag{8}$$

$$\mathbf{E_c} = \sum_{j=1}^{l_c}(\mathbf{y}_{cj} - \bar{x}_c)(\mathbf{y}_{cj} - \bar{x}_c)^T, \tag{9}$$

$$\mathbf{F_c} = \sum_{j=1}^{l_c}(\bar{\mathbf{y}}_{cj} - \bar{y}_c)(\bar{\mathbf{y}}_{cj} - \bar{y}_c))^T. \tag{10}$$

In the second case, we assume that l_{M+1} of the L_k new samples belong to a new, previously unseen class $M + 1$. Here the updated between-class scatter matrix is [25],

$$\mathbf{S'_b} = \sum_{c=1}^{M} n'_c(\bar{x_c}' - \bar{x}')(\bar{x_c}' - \bar{x}')^T + l_{M+1}(\bar{y} - \bar{x}')(\bar{y} - \bar{x}')^T \tag{11}$$

$$= \sum_{c=1}^{M+1} n'_c(\bar{x_c}' - \bar{x}')(\bar{x_c}' - \bar{x}')^T, \tag{12}$$

where n'_c is the number of samples in class c for every chunk of data. If $c = M+1$, then $n'_c = l_{M+1}$ or else, $n'_c = n_c + l_c$. The updated within-class scatter matrix is [25],

$$\mathbf{S'_w} = \sum_{c=1}^{M} \mathbf{\Sigma_c} + \mathbf{\Sigma_{M+1}} = \sum_{c=1}^{M+1} \mathbf{\Sigma'_c}, \tag{13}$$

where, $\mathbf{\Sigma_{M+1}} = \Sigma_{y\in\{y_c\}}(\mathbf{y} - \bar{\mathbf{y}}_\mathbf{c})(\mathbf{y} - \bar{\mathbf{y}}_\mathbf{c})^T$.

The objective function for LDA in Eqn. (3) remains the same, except the scatter matrices are now represented by the expressions in Eqns. (12) and (13).

3. Mathematical Foundation

Definition 1: Stiefel Manifold

The Stiefel manifold (SM) is a compact embedded submanifold of the Euclidean space. A point on the SM, St(k, n), is the set of $n \times k$ dimensional matrices, $k \leq n$, with orthonormal columns equipped with the Frobenius inner product [6].

$$St(k, n) \triangleq \{\mathbf{W} \in \mathbb{R}^{n\times k} : \mathbf{W}^T\mathbf{W} = \mathbf{I}_k\}$$

Definition 2: Grassmann Manifold

The Grassmann manifold represents the collection of subspaces spanned by the columns $n \times k$ dimensional matrices with orthonormal columns [6]

$$G(n, k) \triangleq \{Span(\mathbf{X}) : \mathbf{X} \in \mathbb{R}^{n\times k}, \mathbf{X}^T\mathbf{X} = \mathbf{I}_k\}$$

The points on the GM can also be defined as being equivalence classes of $n \times k$ orthogonal matrices, where two matrices are equivalent if their columns span the same k-dimensional subspace [6].

A point on the Stiefel manifold represents a basis of the subspace whereas a point on the GM represents an entire subspace [24]. Thus, the set of orthogonal matrices that define the same subspace will exist as a single point on the GM. Cunningham et al. [4] argue that for dimensionality reduction, the GM is more suitable for optimization.

Tangent Space

The tangent space at a point $\mathbf{Y}$ on the manifold M is the set of all points $\mathbf{X}$ that satisfy the following equation:

$$\mathbf{Y}^T\mathbf{X} + \mathbf{X}^T\mathbf{Y} = \mathbf{0}_k, \tag{14}$$

where $\mathbf{0}_k$ is a matrix containing all zero entries. The tangent space $T_Y M$ at a point $\mathbf{Y}$ on manifold M, is the linear approximation to the manifold at a particular point and contains all the tangent vectors to M at $\mathbf{Y}$. While optimizing over a manifold, the gradient of an objective function at a point $\mathbf{Y} \in M$ is defined to be along the tangent space at that point. The tangent space is important for optimization on the manifold because the direction of update to the point being optimized must exist in the point's tangent space. Each tangent space defines an exponential mapping to the manifold and thus defines a geodesic curve. The gradients can then be used to travel along the geodesic in order to minimize the objective function.

Gradient on the Manifold

When optimizing over the Grassmann manifold, the direction of each update step must be found in the tangent space of the current location on the manifold. However, it is not possible to restrict the calculated gradients of the loss to the manifold tangent space. To overcome this limitation, the gradients are projected to the tangent space. For an objective function F, the gradients with respect to a point on the manifold $\mathbf{Y} \in M$ is defined as:

$$\nabla F = \frac{\partial F}{\partial \mathbf{Y}} - \mathbf{Y}(\frac{\partial F}{\partial \mathbf{Y}})^T \mathbf{Y}^T \tag{15}$$

Projection

The motion in ambient Euclidean space does not imply that the point will move on the manifold. To move on the manifold during optimization, a small step is taken along the direction that exists in the tangent space of a point on the manifold followed by a retraction to the manifold. However, sometimes a desired direction of motion may not exist in the tangent space of the point. In these instances, the direction must first be projected onto the tangent space. The equation for the projection of a point $\mathbf{Z}$ that exists in ambient Euclidean space to the tangent space of the manifold at point $\mathbf{Y}$ is given as:

$$\pi_{T,\mathbf{Y}}(\mathbf{Z}) = \mathbf{Y}\frac{1}{2}(\mathbf{Y}^T\mathbf{Z} - \mathbf{Z}^T\mathbf{Y}) + (\mathbf{I}_k - \mathbf{Y}\mathbf{Y}^T)\mathbf{Z}. \tag{16}$$

The projection operation eliminates the components that are normal to the tangent space and only keeps the components that are on the manifold tangent space at the current point, $\mathbf{Y}$.

Retraction

The retraction r_k from ambient space to the manifold is a mapping from a point in ambient space to the closest point in the manifold. Updating the solution along the loss minimizing direction will likely introduce a small component that moves the point out of the manifold into ambient space. Therefore, it is important to retract the updated point back to the manifold using a retraction operation. In this work, we use the retraction operation in [4]:

$$r_{\mathbf{k}}(\mathbf{Z}) = \mathbf{U}\mathbf{V}^T \tag{17}$$

using the SVD of $\mathbf{Z} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.

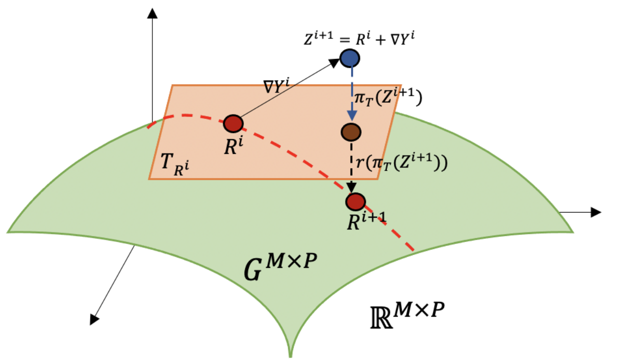

Figure 2. Illustration of the 2-step matrix optimization method. The tangent space is represented by the orange plane and the Grassmann manifold is in green. The dotted red line is the loss minimizing geodesic.

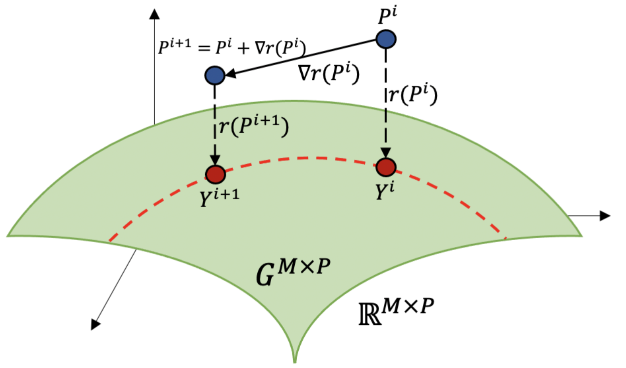

Figure 3. Illustration of the proxy matrix optimization method. The Grassmann manifold is represented in green and the dotted red line is the loss minimizing geodesic.

```
Data: $\mathbf{X} \in \mathbb{R}^{D\times N}$
Result: Locally Optimal $\mathbf{P}$ such that $r(\mathbf{P^i})$
        minimizes the loss $f_\mathbf{X}$ from Eq. 3
initialize $\mathbf{P} \in \mathcal{R}^{M\times P}$ and $\mathbf{P} \notin \mathcal{G}^{M\times P}$;
for $i > iter$ do
    $\mathbf{USV}^T = \mathbf{P^i}$ ; /* Retract $\mathbf{P^i}$ to $\mathcal{G}^{M\times P}$ */
    $r(\mathbf{P^i}) = \mathbf{UV}^T$;
    $\nabla r(\mathbf{P^i}) = \frac{\partial}{\partial \mathbf{P^i}} f_X(r(\mathbf{P^i}))$ ; /* Calculate
     gradients for $\mathbf{P^i}$ using Eq.15 */
    $\mathbf{P^{i+1}} = \mathbf{P^i} - \beta\nabla\mathbf{r}(\mathbf{P^i})$ ; /* Update $\mathbf{P}$ */
end
```
Algorithm 1: Proxy Matrix Optimization.

4. Methodology

In [4], a two-step approach is used to retract a matrix from the ambient Euclidean space onto the manifold to find the optimal projection matrix. The two-step process is illustrated in Fig. 2. For each iteration, i, the projection matrix $\mathbf{R^i}$ is updated by calculating its gradient. This takes the ma-

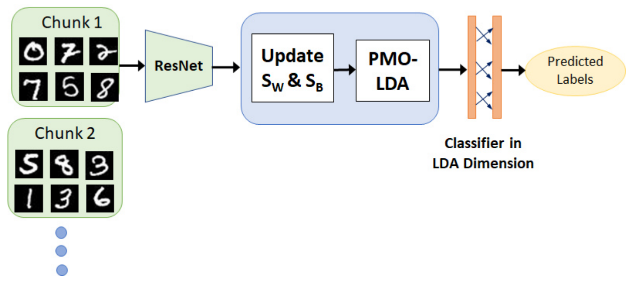

Figure 4. Network architecture of the GILDA++ method for incremental learning from chunks of MNIST images.

trix out of the manifold into the ambient Euclidean space to produce $\mathbf{Z^{i+1}}$. This point is then projected onto the tangent space using Eq. (16). From the tangent space, the point is retracted back onto the manifold using Eq. (17).

The PMO method does not directly optimize a matrix on the manifold, but uses an auxiliary or proxy matrix in ambient space which is retracted to the closest location on the manifold using Eq. (17). The PMO process is illustrated in Fig. 3 and the corresponding algorithm is outlined in Algorithm 1. PMO performs optimization in ambient space by moving each update in the direction that minimizes the loss. The first step in the PMO process is to retract the proxy matrix $\mathbf{P^i}$ to $\mathbf{Y^i}$, its closest location on the manifold $\mathbf{G^{MXP}}$. Once the proxy matrix is retracted to the manifold, the loss is calculated based on the loss function at $\mathbf{Y^i}$. For LDA, this is the loss described in Eqn. (3). This loss is then back-propagated through the singular value decomposition of proxy matrix using a method developed by [11] to a new point $\mathbf{P^{i+1}}$. This point is then retracted back onto the manifold using Eq. 17 to point $\mathbf{Y^{i+1}}$. PMO leverages the autograd routine in Pytorch to back-propagate through the SVD and hence removing the need for any analytical gradient calculation.

The general outline of GILDA++ is provided in Fig. 4. The input to the model can either be the raw images or feature vectors, depending on the experiment performed. The training data chunks are input into the incremental module which makes use of Eqns. (12) and (13) to update the scatter matrices. These updated scatter matrices are fed into the objective function described in Eqn. (3) which is used in the PMO method. Algorithm 1 provides the optimal LDA projection matrix, $\mathbf{X_s}$. The train and test data is then projected onto the lower dimensional space using $\mathbf{X_s}$ and a linear layer is trained to obtain the final classification accuracy.

The integration of Algorithm 1 into a neural network is done by converting the PMO-LDA module into a fully connected layer, which is subsequently connected to the FC layers of the classifier. The loss for LDA is based on the objective function of Eqn. (3) and is backpropagated to obtain the optimal projection matrix.

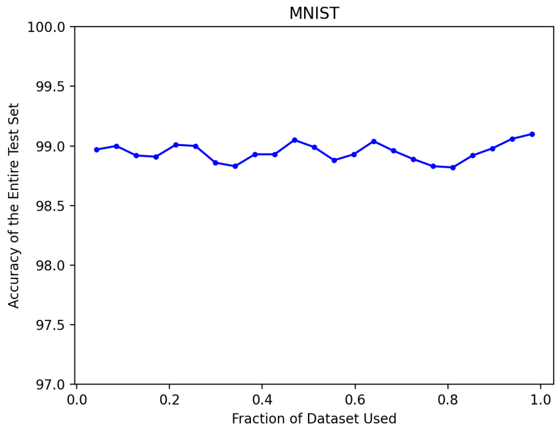

Figure 5. Plot of the accuracy of GILDA++ for MNIST with the fraction of training data provided.

5. Experiments and Results

In this section, we show the effectiveness of GILDA++ for incremental LDA using PMO on three different types of experiments and compare our results with some of the existing methods. Each experiment initializes the scatter matrices in two stages, the base initialization stage, which is the training data provided in the first batch and the subsequent update steps. All the experiments were performed on an NVIDIA Titan V GPU.

5.1. Deep Features

A ResNet-18 [10] network was trained from scratch on the CIFAR-10 [15] and MNIST [16] datasets. These networks were used to extract image features that were fed into the GILDA++ model in a chunk incremental manner. The base initialization and each chunk had a total of 256 samples from all the classes. The dimension of the features from the final convolutional layer of ResNet-18 is 512. Thus, each chunk adds new samples to the existing classes. The scatter matrices are calculated for every chunk increment and the corresponding LDA projection matrix is computed. The dimension of the projected space is one less than the total number of classes. The final accuracy is obtained by training a linear layer on the lower dimensional training projections and testing it on the lower dimensional test data. The linear layer is trained with a cross entropy loss. The batch size in each of these experiments is 256. The optimization used is SGD [23].

From the plots in Figs. 6 and 5, it is clear that even with a small fraction of the data, it is possible to obtain a high accuracy. After a significant portion of the data have been received, performance plateaus and adding more data does not increase in discrimination capabilities.

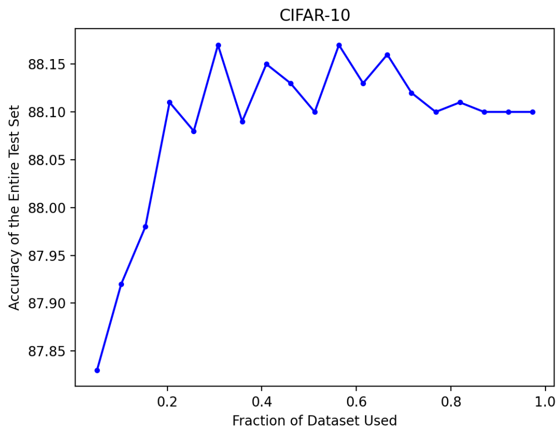

Figure 6. Plot of the accuracy of GILDA++ for CIFAR-10 with the fraction of training data provided.

Table 1. Properties of the UCI datasets.

Dataset	# classes	# instances	# attributes
Wine	3	178	13
Breast Cancer	2	569	30
Heart Disease	2	303	13
Iris	3	150	4
Sonar	2	208	60
Segmentation	7	210	19
Vehicle	4	846	18

5.2. Adding New Samples

The datasets used for these experiments are from the UCI Machine Learning Repository [5]. The specifics of the datasets and their properties are described in Table 1. The train-test split is done randomly and in a 80:20 ratio for each of the datasets. The base initialization is done with a few samples from all the classes and each new chunk of data adds samples to the existing classes. In each of the cases, the lower dimensional projection is one less that the total number of classes. The results are compared with the accuracies obtained from the ILDA method described in [25] and are reported in Table 2.

The plot for the accuracies versus the fraction of training data for the Iris dataset is shown in Fig 7 for GILDA++ and ILDA [25]. From the graph, it is evident that GILDA++ does better in terms of final accuracy on the test dataset. It also shows that just 40% of the training data is sufficient to obtain a discriminative eigenspace.

In Fig. 8 we show the projections in the lower dimensional space for when the projection matrix is obtained from 20%, 50% and 100% of the training data for the UCI wine dataset. The lower dimensional projections appear to become more linearly separable as the amount of training data increases. The accuracy reported is the accuracy on the test data.

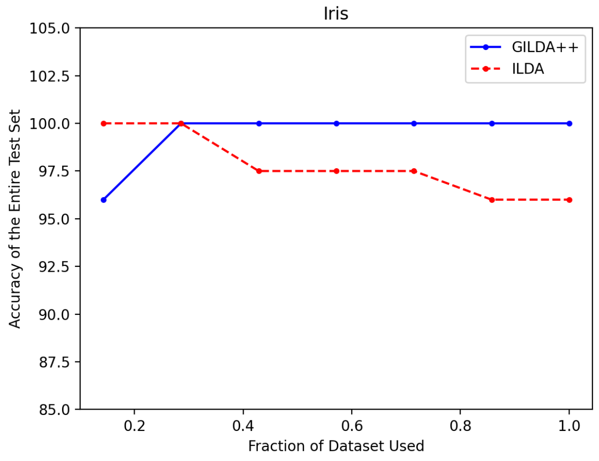

Figure 7. Accuracy comparison of GILDA++ and ILDA [25] for the Iris dataset for fractions of the training data.

Table 2. Comparison between GILDA++ and ILDA [25] for the UCI datasets.

Dataset	Dim	ILDA	GILDA++
Wine	2	96.6	100
Breast Cancer	1	93.8	93.8
Heart Disease	1	59.1	69.2
Iris	2	98	100
Sonar	1	81.2	82.5
Segmentation	6	83.9	84.2
Vehicle	3	75.4	78.3

5.3. Adding New Classes

In this set of experiments, the base initialization is done with data in two classes and each incremental chunk of data provides a random number of new samples which may or may not contain new classes. The chunk size used here is 20. This experiment is done on the ORL database of faces and the final results are compared with two existing methods: Incremental Fast Batch LDA (IFLDA/QR) [27] and Orthogonal LDA (OLDA) [18].

IFLDA/QR [27] is the incremental version of the fast batch LDA (FLDA/QR) that leverages the QR decomposition of the lower triangular matrix. This algorithm takes the centroid of each class cluster and uses it as the matrix for decomposition. The QR decomposition is optimized by making use of the Cholesky-factorization. FLDA/QR has an intrinsic incremental mechanism that is updated using the Gram-Schmidt re-orthogonalization process in a method called IFLDA/QR. While this method is fast and can handle tasks such as insertion of samples to an existing class, insertion of a novel cluster, and insertion of a chunk of data, they

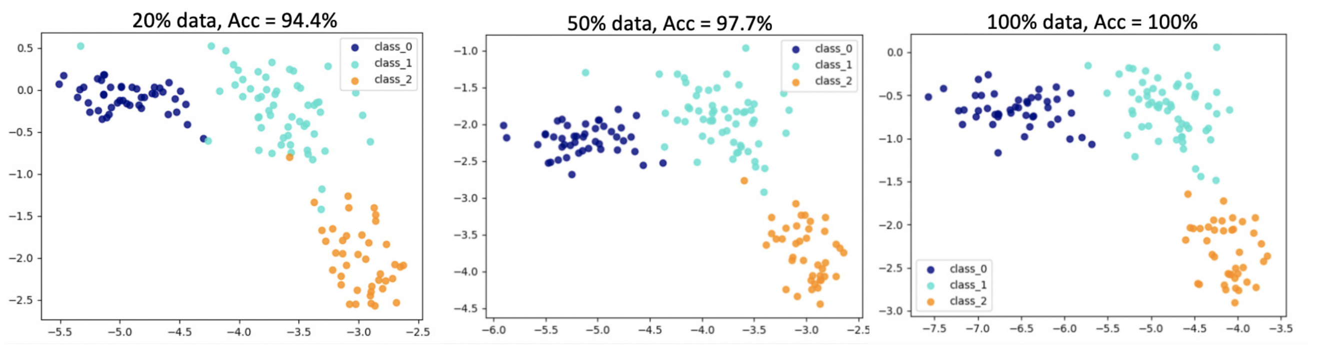

Figure 8. Projection vectors for the UCI wine dataset for 20%, 50% and 100% of the training data and their respective test accuracy.

all require different algorithms for updating the projection matrix.

Orthogonal LDA (OLDA) [18] solves for the objective function described in equation Eqn. (3). It also makes use of the Cholesky decomposition and the QR decomposition to arrive at the optimal projection matrix. Incremental OLDA is the incremental version of OLDA.

Both of the above methods arrive at the exact solution by means of analytically solving for the objective function. The GILDA++ approach is more flexible by making use of the LDA objective function. Table 3 provides the final accuracies on the ORL dataset for GILDA++ and each of these methods.

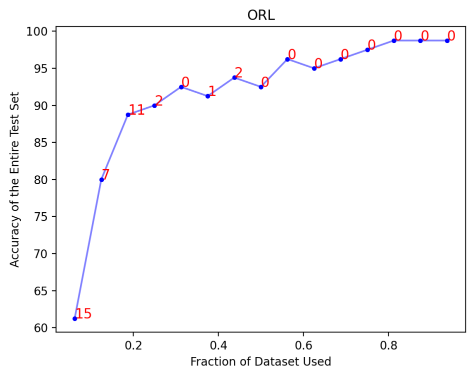

Figure 9. Graph of GILDA++ accuracy versus the fraction of the training data from the ORL dataset. The numbers in red indicate the number of new classes (new face ID's) added in that particular chunk of incoming training data.

From Fig. 9, the maximum accuracy obtained at the final stage of incremental LDA is 98.75%. Even when the scatter matrices are initialized with a few classes, the GILDA++ method provides good accuracy. Each chunk of data randomly adds add new samples from existing classes or new classes denoted by the numbers in red in Fig.9.

Table 3. Accuracy and complexity comparison of GILDA++ with other incremental LDA methods for the ORL dataset.

Method	Accuracy	Complexity
GILDA++	98.75%	$\mathcal{O}(16 \times 10^5)$
IFLA/QR	92.25%	$\mathcal{O}(48 \times 10^5)$
OLDA	98.1%	$\mathcal{O}(64 \times 10^5)$

6. Discussion

In this section we discuss the computational complexity of GILDA++ and its convergence properties.

6.1. Computational Complexity

GILDA++ consists of three steps that are done at every epoch. The first one involves calculating the scatter matrices from each batch of incremental data. The updated scatter matrices are then used for the LDA objective function computation. The second step is the retraction of the projection matrix from ambient Euclidean space to the closest point on the manifold. The third step is to calculate the loss function and backpropagate it. Consider that the total number of classes in each batch is c, D is the input dimension and d is the lower dimensional projection, which in the case of LDA is one lower than the total number of classes, C. The number of samples in each incremental chunk is the batch size, L. N is the total number of samples. In the first step, the computational complexity of calculating the between-class scatter matrix is $\mathcal{O}(cD^2)$ and the within-class scatter matrix is $\mathcal{O}(LD^2)$. The objective function given by Eqn. (3) has a computational complexity of $\mathcal{O}((L+c)D^2)$. In the second step, the retraction to the manifold is essentially an SVD which has a computational complexity of $\mathcal{O}(D^2d + d^3)$. The third step is the backpropagation of the loss function which involves taking the partials and has a computational complexity of $\mathcal{O}(LDd)$. This leads to an overall complexity of $\mathcal{O}((L+d+c)D^2 + d^3 + LDd)$ per iteration. Since d is one less than the number of classes, it is very small in comparison to D. The complexity or IFLDA/QR and Orthogonal LDA for the ORL faces dataset are provided in Table. 3. The complexity of the ILDA method is obtained by calculating the scatter matrices and the eigenvalue solution for each iteration, which is then $\mathcal{O}((L + c)D^2 + NDmin(N, D))$, which is the least efficient of all the methods.

6.2. Convergence

We provide a heuristic proof of convergence of GILDA++ and show its advantage over the two-step method in [4] for manifold optimization. The two-step method is shown to have a guaranteed convergence [4]. This follows from the fact that the gradients in the ambient Euclidean space can be decomposed into their normal and tangential components. The tangential components lie on the tangent space to the manifold. Since both of these are orthogonal to one another, setting the normal component to zero does not affect the tangential component. The gradients are defined to be in the loss minimizing direction and as a result, the tangential components of the gradients also lie in the loss minimizing direction. The points on the tangent space are then retracted to the manifold, which by definition of retraction is the closest point on the manifold and thus lies on the loss minimizing geodesic of the manifold.

The GILDA++ method integrates the manifold retraction step into the loss function. This means that the gradient of the loss function will automatically move the matrix along the loss minimizing geodesic on the manifold, thus showing convergence to the minimum value. If the GILDA++ method is initialized with the eigenvalue solution, the number of iterations taken to reach the minimum is lower than the two-step method due to this unconstrained optimization and the removal of the projection to the tangent space step.

7. Conclusions

We introduce GILDA++ that combines incremental LDA with matrix optimization on the Grassmann manifold to arrive at the lower dimensional projections which are used for classification. We show that GILDA++ performs better than the existing methods for experiments that add new samples and new classes for every chunk of incoming data. Since GILDA++ is an incremental method and uses automatic differentiation and SGD to arrive at the lower dimensional LDA projection, it is ideally suited to be used as a layer in a neural network for end-to-end training.

Acknowledgements

This research was partly supported by the Air Force Office of Scientific Research (AFOSR) under Dynamic Data Driven Applications Systems (DDDAS) grant FA9550-18-1-0121 and the National Science Foundation award number 1808582.

References

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. Riemannian geometry of Grassmann manifolds with a view on algorithmic computation. *Acta Appl. Math.*, 80(2):199–220, 2004.

[2] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman. Eigenfaces vs. fisherfaces: recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, 1997.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[4] John P. Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research*, 16(89):2859–2900, 2015.

[5] Dheeru Dua and Casey Graff. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2017.

[6] Alan Edelman, Tomás A Arias, and Steven T Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[7] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936.

[8] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition (2nd Ed.)*. Academic Press Professional, Inc., USA, 1990.

[9] Jihun Hamm and Daniel D. Lee. Grassmann discriminant analysis: A unifying view on subspace-based learning. *ICML '08*, page 376–383, 2008.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[11] C. Ionescu, O. Vantzos, and C. Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2965–2973, 2015.

[12] Jieping Ye, R. Janardan, C. H. Park, and H. Park. An optimization criterion for generalized discriminant analysis on undersampled problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):982–994, 2004.

[13] Jieping Ye, Qi Li, Hui Xiong, H. Park, R. Janardan, and V. Kumar. IDR/QR: an incremental dimension reduction algorithm via QR decomposition. *IEEE Transactions on Knowledge and Data Engineering*, 17(9):1208–1222, 2005.

[14] Tae-Kyun Kim, B. Stenger, J. Kittler, and R. Cipolla. Incremental linear discriminant analysis using sufficient spanning sets and its applications. *International Journal of Computer Vision*, 91:216–232, 2010.

[15] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[16] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[17] L. Liu, Y. Jiang, and Z. Zhou. Least square incremental linear discriminant analysis. In *2009 Ninth IEEE International Conference on Data Mining*, pages 298–306, 2009.

[18] Gui-Fu Lu, Jian Zou, and Yong Wang. A new and fast implementation of orthogonal lda algorithm and its incremental extension. *Neural Process. Lett.*, 43(3):687–707, June 2016.

[19] Juwei Lu, Konstantinos Plataniotis, and Anastasios Venetsanopoulos. Face recognition using LDA-based algorithms. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 14:195–200, 02 2003.

[20] Kevin R. Moon, David van Dijk, Zheng Wang, William Chen, Matthew J. Hirn, Ronald R. Coifman, Natalia B. Ivanova, Guy Wolf, and Smita Krishnaswamy. PHATE: A dimensionality reduction method for visualizing trajectory structures in high-dimensional biological data. *bioRxiv*, 2017.

[21] Navya Nagananda, Breton Minnehan, and Andreas Savakis. Grassmann Iterative Linear Discriminant Analysis with Proxy Matrix Optimization . *NeurIPS workshop on Differential Geometry meets Deep Learning*, 2020.

[22] C. Radhakrishna Rao. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(2):159–203, 1948.

[23] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[24] S. K. Roy, Z. Mhammedi, and M. Harandi. Geometry aware constrained optimization techniques for deep learning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4469, 2018.

[25] Shaoning Pang, S. Ozawa, and N. Kasabov. Incremental linear discriminant analysis for classification of data streams. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(5):905–914, 2005.

[26] Pavan Turaga, Ashok Veeraraghavan, and Rama Chellappa. Statistical analysis on stiefel and grassmann manifolds with applications in computer vision. *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[27] Yi Wang, Xin Fan, Zhongxuan Luo, Tianzhu Wang, Maomao Min, and Jiebo Luo. Fast online incremental learning on mixture streaming data. *AAAI Conference on Artificial Intelligence*, 2017.

[28] H. Zhao and P. C. Yuen. Incremental linear discriminant analysis for face recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(1):210–221, 2008.