Feedforward Neural Networks Initialization based on Discriminant Learning

Kateryna Chumachenko[a,*], Alexandros Iosifidis[b], Moncef Gabbouj[a]

[a] *Faculty of Information Technology and Communication Sciences, Tampere University, FI-33720 Tampere, Finland*
[b] *Department of Electrical and Computer Engineering, Aarhus University, DK-8200 Aarhus, Denmark*

Abstract

In this paper, a novel data-driven method for weight initialization of Multilayer Perceptrons and Convolutional Neural Networks based on discriminant learning is proposed. The approach relaxes some of the limitations of competing data-driven methods, including unimodality assumptions, limitations on the architectures related to limited maximal dimensionalities of the corresponding projection spaces, as well as limitations related to high computational requirements due to the need of eigendecomposition on high-dimensional data. We also consider assumptions of the method on the data and propose a way to account for them in a form of a new normalization layer. The experiments on three large-scale image datasets show improved accuracy of the trained models compared to competing random-based and data-driven weight initialization methods, as well as better convergence properties in certain cases.

Keywords: Neural networks initialization, discriminant learning

1. Introduction

In recent years, Deep Learning became the dominant paradigm in the fields of Machine Learning and Computer Vision owing to the availability of large public data and computational resources. Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) are being widely utilized for a variety of tasks, including object detection [1, 2, 3], object tracking [4], semantic image segmentation [5] and action recognition [6, 7].

With the rise of Deep Learning, methods for weight initialization in neural networks received increased attention, and weight initialization strategies that can accelerate the training process while leading to competitive performance remain an open research problem. Multiple approaches have been proposed to

*Corresponding author
Email addresses: kateryna.chumachenko@tuni.fi (Kateryna Chumachenko), ai@ece.au.dk (Alexandros Iosifidis), moncef.gabbouj@tuni.fi (Moncef Gabbouj)

solve this problem to date. Early works in the field of artificial neural networks were relying on weight initialization from random distributions, further evolving to initialization methods with controlled parameters, such as Glorot [8] or He initialization [9]. Others methods proposed data-driven initialization procedures [10, 11, 12, 13, 14, 15], which are described in more detail in Section 2.1. The main motivation behind the latter approach primarily stems from the nature of training processes of neural networks: since gradient-based optimization of non-convex functions leads to local minima solutions, starting the optimization from a favourable point can result in better performance and faster convergence.

Several data-driven initialization methods were proposed based on statistical learning, primarily focusing on utilization of Principal Component Analysis (PCA) [16] or Linear Discriminant Analysis (LDA) [17] to determine the data transformations in successive layers of the network. Nevertheless, these methods have a number of limitations: PCA only satisfies the criteria of high variance in the data while not enforcing discriminative properties, and LDA assumes unimodal class distributions for the data representations in all the layers of the neural network. Here it should be noted that while data representations at the last hidden layer of a trained neural network equipped with softmax/linear output neurons are expected to form unimodal classes, this is not the case for early layers. Therefore, the assumption of class unimodality throughout the layers of the network for weight initialization limits the potential of the model. Another major limitation comes from the limited dimensionality of the projection directions learnt by these methods, thus limiting the number of neurons/weights that can be initialized by adopting them.

As a remedy for the above-mentioned limitations, in this paper, we propose a novel data-driven weight initialization approach based on discriminant learning that allows to relax the above-mentioned limitations. First, we relax the class unimodality assumption for the data representations at all network layers by representing it with several subclasses and formulating the optimization problem for weights initialization accordingly, hence improving the suitability of a model for real-world scenarios. Second, the proposed approach relaxes limitations to the model architecture, as the maximal number of initialized neurons/filters at a certain layer relies on a controlled parameter, i.e. the total number of subclasses forming the classification problem. Third, the proposed approach does not rely on eigendecomposition that becomes computationally intensive for high-dimensional data, hence providing faster initialization especially for wide CNN architectures, i.e., those with a large number of neurons/filters in each layer.

The main contributions of the paper can be summarized as follows:

- A novel weight initialization procedure for MLPs and CNNs is proposed that leads to flexible network architecture design and potentially better generalization due to its multi-modal formulation. It is experimentally shown that the adoption of the proposed initialization procedure leads to faster convergence of the subsequent gradient-based training process compared to existing approaches.

- A new normalization layer that overcomes limitations related to the assumption of mean-centered data, adopted by the proposed method, as well as other data-driven network initialization methods is proposed.
- Experimental results show that utilization of a small number of data samples generally suffices for effective network initialization, hence, further reducing the computational requirements for training the network.

The remainder of the paper is structured as follows. Section 2 describes the related methods utilized for weight initialization in neural networks, Section 3 describes the proposed weight initialization approach along with the motivation behind it, Section 4 presents the experiments performed to assess the proposed approach, along with the experimental results, and Section 5 provides conclusions of the work.

2. Related Work

Generally, methods for weight initialization of neural networks can be divided into two categories: the first is based on initialization from a random distribution and the second follows a data-driven process. For a long time, the most widely-used and straightforward initialization approach was the initialization from a random distribution: a Gaussian distribution with zero mean and small hand-tuned standard deviation, or from a Uniform distribution in the range of $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$, where n is the number of input neurons in the corresponding layer. It has been further observed that such initialization often leads to poor convergence, and saturated activations. In [8], it was shown that the commonly-used activation functions, namely, sigmoid, hyperbolic tangent, and softsign suffer from saturation of activation in the top layers of the network, when initialized from random uniform distribution. As a remedy, a new weight initialization method was proposed, with an objective of preserving the variance of activation vectors between the layers during the forward propagation, and the variance of the gradients between the layers during backward propagation. In practice, the following initialization approach is utilized, approximately satisfying the above-mentioned objectives:

$$\mathbf{W}_j \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right], \tag{1}$$

where $\mathbf{W}_j$ is the weight matrix at layer j, $U[\cdot]$ denotes the Uniform distribution, and n_j and n_{j+1} denote the number of neurons at layers j and $j+1$, respectively. Hereafter, we refer to this approach as Glorot initialization (also commonly referred to as Xavier initialization) [8] . Here we should note that, in its derivation, the method assumes linear activations at the initialization and that the input feature variances are equal.

A further step towards controlling the statistics of the distribution from which the weights are initialized was taken in [9], where a similar motivation to

that of Glorot is utilized for initialization. Unlike the work in [8], the authors consider ReLU activation, and show that the proposed approach outperforms the Glorot initialization especially when used for deep neural networks. The initialization is done as follows:

$$\mathbf{W}_j \sim \mathcal{N}\left[0, \frac{\sqrt{2}}{\sqrt{n_j}}\right], \tag{2}$$

i.e., the weights of layer j are initialized from a Gaussian distribution with zero mean and variance of $\frac{2}{n_j}$. Additionally, fully-randomized methods based on stochastic configuration algorithms have been proposed [18].

As opposed to methods based on random initialization, multiple approaches exploiting certain data properties have recently been proposed. The most notable one is initialization by pre-training on a larger dataset of similar domain[1], such as ImageNet [20] for Computer Vision tasks. Nevertheless, such initialization was questioned in [21], where it was shown that the benefits arising from weights initialization based on pre-training generally lie in faster convergence in earlier iterations, but not necessarily leading to better performance as compared to random initialization. Other notable data-driven approaches include initialization from cluster centroids obtained by applying (spherical) clustering on whitened data, hence capturing statistical properties of the dataset [22, 10, 23]. Another method performs normalization of networks' weights based on the empirical statistics of the network activation obtained from the training data samples, as well as its gradients [10] Notably, the approach presented in [10] applies the normalization to both k-means and PCA initialized networks.

2.1. Weight initialization via subspace learning

A set of data-driven weight initialization methods that were proven beneficial for weight initialization in neural networks relies on utilization of subspace learning techniques. The early works utilizing subspace learning for determining the weights of neural networks include PCANet [13] and LDANet [14]. These methods focus on supervised image classification in a CNN-like manner, where a set of patches are extracted from the training images and flattened to form a data matrix. From this data representation, a weight matrix is obtained by applying Principal Component Analysis [16] or Linear Discriminant Analysis [17]. The resulting weight matrix is subsequently reshaped to obtain a set of convolutional filters, which are convolved with the training images to obtain the data representations at the output of the first layer. This process is applied for several layers[2], followed by a pooling operation and an activation function. In these approaches, no subsequent fine-tuning of the network's parameters via back-propagation is performed, while the pooling operation as well as the uti-

[1] This approach is commonly referred to as transfer learning [19].

[2] The original LDANet and PCANet methods apply this process twice to determine the filters of two convolutional layers.

lized activation function are specially designed, i.e. they are not among the commonly-used ones in the field of deep learning. However, these methods can be perceived as the first baselines drawing the connection between the subspace learning and deep learning methods.

Further notable attempts of linking subspace learning with deep learning architectures include LDA-based weight initialization proposed in [15, 24]. By its nature, this work is more similar to our proposed approach in that the weights obtained by a discriminant learning method are used for initialization of the neural network which is further trained with backpropagation, instead of solely considering the forward propagation scenario. LDA is employed to initialize the weights of a layer, and each subsequent layer is initialized from the weight matrix obtained by LDA applied to the outputs of the preceding layer. Similarly to PCANet and LDANet, the weight matrix is learnt from patches extracted from the outputs of the previous layer and is flattened to obtain a rectangular data matrix. The last classification layer is initialized with the discriminant matrix of LDA, and the network is subsequently trained with backpropagation.

The authors in [11, 12] proposed a feedforward design approach for initializing the layers in CNN based on data statistics from the output of their preceding layers. The weights in convolutional layers are obtained from a variant of Principal Component Analysis proposed by the authors, namely, Subspace Approximation with Adjusted Bias (Saab). The dense layers that are added after the convolutional layers are trained by applying a linear regression using subclass labels obtained by clustering the data. The last fully-connected layer is trained by linear regression to true class labels. This method focuses on the forward propagation scenario too.

3. Initialization based on Discriminant Learning

Let us consider a standard dense feedforward neural network. Given a vector $\mathbf{x} \in \mathbb{R}^D$ as input, a neural network with L layers applies a hierarchical transformation

$$\mathbf{y} = f_a^L(\mathbf{W}_L^T f_a^{L-1}(\ \mathbf{W}_{L-1}^T \ ... \ f_a^1(\mathbf{W}_1^T \mathbf{x} + b_1) + b_{L-1}) + b_L), \tag{3}$$

where $f_a^l(\cdot)$ is the (element-wise) activation function at layer l, $\mathbf{W}_l \in \mathbb{R}^{D_l \times D_{l+1}}$ is the corresponding weight matrix, and b_l is the bias term. For the sake of simplicity of notation, here we assume that the bias terms are accounted for by using an augmented version of the data representations of the network layers and, thus, are incorporated in the corresponding weight matrices $\mathbf{W}_l$, $l = 1, \ldots, L$. Similarly, a CNN performs a hierarchical data transformation of the form

$$\mathbf{y} = f_a^L(\hat{\mathbf{W}}_L * f_a^{L-1}(\ \hat{\mathbf{W}}_{L-1} * \ ... \ * f_a^1(\hat{\mathbf{W}}_1 * \mathbf{x} + b_1) + b_{L-1}) + b_L), \tag{4}$$

where $\hat{\mathbf{W}}_l$ is a set of convolutional filters at layer l, b_l is the bias term, and $f_a^l(\cdot)$ is the activation function. For CNNs which combine convolutional

and dense layers, the corresponding data transformation is obtained by simply combining data transformations of the form in (4) and (3) in a hierarchical manner.

3.1. Motivation

Most of the earlier data-driven methods primarily focused on the affine transformation of $\mathbf{y} = \mathbf{W}_l^T \mathbf{x}^{(l)}$, where $\mathbf{x}^{(l)}$ is the representation of the input sample at the feature space defined at layer l. To deal with the convolution operation $\mathbf{y} = \hat{\mathbf{W}}_l * \mathbf{x}^{(l)}$ in (4), the convolution operation is transformed to a vector-based affine transformation by sampling patches from the input $\mathbf{x}^{(l)}$, flattening them to create vectors and determining an affine transformation matrix $\mathbf{W}_l$, which is further reshaped to form $\hat{\mathbf{W}}_l$. Several works [14, 15] utilize LDA for learning the matrix $\mathbf{W}_l$, i.e., the projection is obtained by solving the eigendecomposition problem of $\mathbf{S}_w^{(l)}\mathbf{w} = \lambda \mathbf{S}_b^{(l)}\mathbf{w}$ and selecting eigenvectors corresponding to smallest eigenvalues, where $\mathbf{S}_w^{(l)}$ and $\mathbf{S}_b^{(l)}$ are the within-class and between-class scatter matrices defined on the data representations at the layer l. Others [13, 11, 12] have applied Principal Component Analysis, i.e, the matrix $\mathbf{W}$ is obtained by performing eigendecomposition on the covariance matrix of the data representations at layer l, i.e. $\mathbf{S}_t^{(l)}$.

Both of these approaches have certain limitations. Being an unsupervised method, PCA does not take advantage of the class label information of the data. Therefore, one of its limitations lies in the fact that the learnt subspace is only optimal in terms of preserving the variance of the projected data; however, no discriminative properties are enforced. Besides, PCA can only learn a (sub)space with dimensionality at most equal number of dimensions to that of the original space. This leads to the inability of learning enough meaningful (i.e., those having discriminative properties) filters/neurons, as the number of filters of the first layers is generally significantly higher than that of dimensions in the the input data, especially in the case of CNNs.

Linear Discriminant Analysis provides a remedy to the limitation of PCA related to the disregard of the class label information of data, finding a subspace where the classes are discriminated. However, it relies on an assumption of unimodality of data of each class, which is rarely the case in real-world scenarios, and especially on the earlier layers of the networks. As a result, such an assumption leads to limitations in the learning potential of the model. Besides, the limitation of LDA with regard to the ability to learn a reasonable amount of meaningful neurons or filters is even higher than that of PCA, as the dimensionality of the learnt subspace is bounded by the rank of the between-class scatter matrix, which is, in turn, bounded by the number of classes. Therefore, the use of LDA for initialization only allows to obtain a very limited number of meaningful projection dimensions, and, consequently, a limited number of meaningful neuron weights in the layer, putting limitations on the network architectures that can be initialized using it.

In addition to the above-mentioned limitations, one can notice that both LDA and PCA rely on eigendecomposition of $D \times D$ matrices that becomes

computationally intensive especially for high-dimensional data. At the same time, especially in the case of CNN, the data is likely to reach significantly high dimensionality: given the data matrix is created similarly to [10, 15, 13], the dimensionality of the patch matrix corresponding to layer j reaches $k^2 n_j$, where k is the filter size, and n_j is the number of filters. Considering commonly-used CNN models, where the number of filters of convolutional layers generally ranges from 32 to 512, and a commonly-used filter size of 5 pixels, this leads to dimensionality ranging from 800 up to 12800, which is substantially high in terms of computational requirements of eigendecomposition-based subspace learning methods. For example, in this case the computational complexity of initialization based on LDA or PCA would reach $N(k^2 n_j)^2 + (k^2 n)^3$ [25], while for the proposed approach it is proportional to $Nk^2 n_j d$ or $k^2 n_j N^2$ if $N < k^2 n_j$ and $N(k^2 n_j)^2$ if $N > k^2 n_j$ [26], where N is the number of samples and d is the dimensionality of the learnt space, which is in either case less than the complexity of initialization based on LDA or PCA.

3.2. Proposed approach

In this section we consider the limitations of already existing methods and propose steps for their relaxation. More specifically, we consider assumptions on unimodality of data representations in the layers of a network, limitations in the number of neurons/filters that can be initialized, and the high computational requirements in high-dimensional settings. A first step towards overcoming these limitations can be taken by employing Subclass Discriminant Analysis [27], that relaxes the assumptions on unimodality of classes. To recall, this is achieved by expressing each class with a set of subclasses determined by applying some clustering algorithm on the data of each class. Similarly to LDA, SDA optimizes the Fisher-Rao's criterion. Considering the optimization problem to be solved for initializing the weights of the l-th layer, the generalized eigenanalysis problem $\mathbf{S}_t^{(l)}\mathbf{w} = \lambda \mathbf{S}_b^{(l)}\mathbf{w}$ is solved, where

$$\mathbf{S}_t^{(l)} = \sum_{i=1}^{N} (\mathbf{x}_i^{(l)} - \boldsymbol{\mu}^{(l)})(\mathbf{x}_i^{(l)} - \boldsymbol{\mu}^{(l)})^T, \tag{5}$$

$$\mathbf{S}_b^{(l)} = \sum_{i=1}^{C-1} \sum_{n=i+1}^{C} \sum_{j=1}^{K_i} \sum_{h=1}^{K_n} p_{ij}^{(l)} p_{nh}^{(l)} (\boldsymbol{\mu}_{ij}^{(l)} - \boldsymbol{\mu}_{nh}^{(l)})(\boldsymbol{\mu}_{ij}^{(l)} - \boldsymbol{\mu}_{nh}^{(l)})^T, \tag{6}$$

where C is the number of classes, K_i is the number of subclasses in class i, $\boldsymbol{\mu}^{(l)}$ is the mean of the data representations in layer l, i and n are class labels, and j and h are subclass labels. $p_{ij}^{(l)}$ and $p_{lh}^{(l)}$ are the subclass priors, i.e. $p_{ij}^{(l)} = \frac{N_{ij}}{N}$, where N_{ij} is the number of samples in subclass j of class i and N is the total number of samples in $\mathbf{X}^{(l)} = [\mathbf{x}_1^{(l)}, \ldots, \mathbf{x}_N^{(l)}] \in \mathbb{R}^{D_l \times N}$. The matrix $\mathbf{W}_l \in \mathbb{R}^{D_l \times D_{l+1}}$ can be then formed by the eigenvectors corresponding to the D_{l+1} smallest eigenvalues.

Such representation is particularly beneficial in the CNN case, where each data sample constitutes a representation of a patch from an image. Assuming

that patches within the same class corresponding to non-essential background and those representing the object of interest or certain useful features are clustered into different subclasses, there is no penalization for them being matched far from each other in the learnt feature space. In contrast, LDA forces all data samples belonging to the same class to lie close to each other in the projection space, enforcing unnecessary similarity requirements for essential features and background patches. Moreover, by utilizing SDA the potential dimensionality of the subspace is bounded by the total number of subclasses forming the problem at hand. That is, the maximum number of discriminant directions that can be determined is increased to $\sum_{i=1}^{C} K_i$. The potential set of architectures is, therefore, significantly expanded compared to LDA. However, it is still bounded by the dimensionality of input data. We propose to overcome this limitation by following a process inspired by Graph Embedding [28] and Spectral Regression [29] in the following.

The criterion function of SDA can be reformulated utilizing Graph Embedding framework [28]. For data centered at $\boldsymbol{\mu}^{(l)}$, it can be seen that

$$\mathbf{S}_t^{(l)} = \mathbf{X}^{(l)}\mathbf{X}^{(l)T}, \tag{7}$$

$$\mathbf{S}_b^{(l)} = \mathbf{X}^{(l)}\mathbf{L}_b^{(l)}\mathbf{X}^{(l)T}, \tag{8}$$

where $\mathbf{L}_b^{(l)}$ is the Laplacian matrix defined on the data representations at the l-th layer of the network for the between-class matrix:

$$\mathbf{L}_b^{(l)}(i,j) = \begin{cases} \frac{N-N_{c_i}}{N^2 N_{ch}}, & \text{if } z_i^{(l)} = z_j^{(l)} = h \\ 0, & \text{if } z_i^{(l)} \neq z_j^{(l)},\ c_i = c_j \\ -\frac{1}{N^2}, & \text{if } c_i \neq c_j \end{cases}, \tag{9}$$

where c_i is the class label of $\mathbf{x}_i^{(l)}$, and $z_i^{(l)}$ is the subclass label of $\mathbf{x}_i^{(l)}$, N_c is the number of samples in class c and $N_{ch}^{(l)}$ is the number of samples in subclass h of class c at layer l.

Exploiting the new formulations of $\mathbf{S}_b^{(l)}$ and $\mathbf{S}_t^{(l)}$, and Spectral Regression [29], the solution can be obtained by following several steps:

1. The between-class Laplacian matrix $\mathbf{L}_b^{(l)}$ is created following Eq. 9.
2. Assuming there exists such $\mathbf{t}$ that $\mathbf{t} = \mathbf{X}^{(l)T}\mathbf{w}$, the eigenanalysis problem $\mathbf{L}_b^{(l)}\mathbf{t} = \lambda\mathbf{t}$ is solved and the matrix $\mathbf{T}^{(l)}$ is created out of the obtained vectors.
3. The regression of $\mathbf{T}^{(l)}$ to $\mathbf{W}^{(l)}$ is performed as

$$\mathbf{W}^{(l)} = \left(\mathbf{X}^{(l)}\mathbf{X}^{(l)T} + \alpha\mathbf{I}\right)^{-1}\mathbf{X}^{(l)}\mathbf{T}^{(l)T}. \tag{10}$$

The matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{D_l \times D_{l+1}}$ can be further orthogonalized or l_2-normalized. In practice, we observed that l_2-normalization results in better performance. More-

over, when applying l_2-normalization instead of orthogonalization, the number of projection directions D_{l+1} can be expanded beyond the dimensionality D_l of the data representations at layer l. This is achieved by performing a class-wise clustering process to determining $\sum_{i=1}^{C} K_i > D_{l+1}$ subclasses, and using the eigenvectors of $\mathbf{L}_b^{(l)}$ corresponding to the largest D_{l+1} eigenvalues to form $\mathbf{T}^{(l)}$. Such an approach allows us to define the number of neurons in layer $l+1$ by controlling the total number of subclasses in layer l, leading to the initialization of as many meaningful neurons as is required by the architecture of the network. Note that, due to the block structure of $\mathbf{L}_b^{(l)}$, the first $C-1$ dimensions are guaranteed to encode the class discriminant information, similarly to LDA. In this sense, the layer initialized using the proposed approach is guaranteed to have at least the same discriminative power as using LDA.

Here we should note that the use of clustering and subsequent cluster label information has been previously performed in [22, 11]. In [22], clustering is applied to the whole dataset and the cluster centroids are used for initialization. In [11], clustering is applied to the whole dataset and one-hot encoded vectors are created using the obtained cluster labels, followed by a least-squares regression to obtain the projection matrix used for initialization. In both of these settings, however, the class label information is not considered. Therefore, the use of such methods in a supervised setting is rather limited. Besides, the proposed approach determines the projection directions in which the data achieves optimal subclass separability, rather than regressing directly to the cluster labels.

The proposed approach can further be extended to improve the computational efficiency on large datasets, where eigendecomposition of $\mathbf{L}_b^{(l)}$ becomes infeasible. The speed-up is achieved by observing that $\mathbf{L}_b^{(l)}$ has a certain block structure, therefore its eigenvectors have a similar block structure as well. Given that a vector of ones is an eigenvector of $\mathbf{L}_b^{(l)}$, we can create the $\sum_{i=1}^{C} K_i - 1$ target vectors of random values with desired structure and orthogonalize them starting from a vector of ones. The detailed procedure for creation of target vectors is shown in Algorithm 1. The approach has recently been shown beneficial in a conventional subspace learning setting for speeding up eigendecopmposition-based SDA [26, 30], and an incremental solution was proposed [31]. While the methods in [26, 30, 31] were proposed for purely shallow statistical learning, here we investigate the utilization of similar ideas for data-driven neural network initialization. The suitability of the proposed ideas for network initialization is dictated by a range of advantages provided by the method in terms of accounting for potential multi-modality present in intermediate layers of the network, faster initialization compared to conventional data-driven methods, as well as absence of restrictions in terms of width of neural network layers. At the same time we investigate ways of addressing the limitations of the methodology in terms of assumptions on data properties. For the sake of clarity, hereafter we refer to the proposed initialization approach as fastSDA as defined in [26, 30, 31] in contrary to eigendecomposition-based SDA.

Algorithm 1: Discriminant target vectors calculation

Function `getTargets(`y,y_{cl},C,Z,N,D`):`

Input: $y : N \times 1$ vector with class labels; $y_{cl} : N \times 1$ vector with the cluster labels; Z : number of clusters in each class; C : number of classes; N : number of elements; D : dimensionality of data;

`%class-level vectors`;

for $i \leftarrow$ *iterate through 1:C* **do**
  $RVals$ = random(1,C-1)
  $T^{(l)}[y == i, :]$ = tile($RVals$, len(y==i),1)
end

$S \leftarrow$ unique numbers of elements in each class sorted in ascending order;

`%cluster level vectors`;

for $s \leftarrow$ iterate through S **do**
  $k \leftarrow$ classes with s elements; $m \leftarrow$ length(k);
  $RVals$ = random($m * Z$, m*(Z - 1))
  for $i \leftarrow$ *iterate through* k **do**
    for $j \leftarrow$ *iterate through 1:Z* **do**
      ixs = where($y == i$ & $y_{cl} == j$)
      $Tclust^{(l)}[ixs,:] \leftarrow$ tile($RVals$, (length(ixs),1))
    end
  end
  $T^{(l)} \leftarrow$ append $Tclust^{(l)}$ as columns on the right;
end

$T^{(l)} \leftarrow$ append $N \times 1$ vector of ones as a column on the left;
Orthogonalize $T^{(l)}$; remove first column of $T^{(l)}$;
return $T^{(l)T}$

Algorithm 2: Initialization of l^{th} Dense layer.

Function `dense_init(`$X^{(l)}$,y,N_neur,C,D`):`

Input: $X^{(l)} : D \times N$ data representation at l^{th} layer; $y : N \times 1$ vector with class labels;N_neur :number of neurons;

$Z \leftarrow$ceil(N_neur/C)
$X^{(l)} \leftarrow \frac{X^{(l)} - mean(X^{(l)})}{\sqrt{var(X^{(l)}) + \epsilon}}$
$y_{cl} \leftarrow$ Cluster($X^{(l)}, Z$)
$T^{(l)} \leftarrow$ getTargets(y,y_{cl},C,Z,N,D)
if $D < N$ **then**
  $R \leftarrow (\text{chol}(X^{(l)} X^{(l)T}))^{-1}$
  $W^{(l)} \leftarrow R R^T X^{(l)} T^{(l)T}$
else
  $R \leftarrow (\text{chol}(X^{(l)T} X^{(l)}))^{-1}$
  $W^{(l)} \leftarrow X^{(l)} R^T R T^{(l)T}$
end
$W^{(l)} \leftarrow$Select first N_neur dimensions of $W^{(l)}$ and normalize with $l2$ norm
return $W^{(l)}$

3.3. Initialization procedures

The proposed approach can be used for initializing Dense and Convolutional layers following equations (3) and (4). The initialization procedure starts from

learning the weight matrix of the first layer on the input data. The data is further transformed with the learnt matrix and transformations defined by the architecture, e.g., Activation and Pooling. The transformed data is subsequently used for initializing the next Dense or Convolutional layer, and the process continues until the Output layer, which is initialized randomly[3]. After the initialization of the whole network, it is trained with backpropagation in the conventional manner. The procedures for initializing weight matrix $\mathbf{W}^{(l)}$ and filters $\hat{\mathbf{W}}^{(l)}$ for the l^{th} Dense or Convolutional layer are shown in Algorithms 2 and 3, respectively.

In order to account for the mean-centering assumption on the data, the input data at each layer is standardized during the weight initialization step. Therefore, to take this into account during the backpropagation step, we add Batch Normalization layers before each of the Dense layers in the architecture.

3.4. Vector Batch Normalization

Initializing the parameters of a neural network with the proposed method requires the training data to be mean-centered, such that Eqs. (7) and (8) express the total and the between-class scatter of the training data. For initializing Dense layers, this is accounted by means of Batch Normalization. For Convolutional layers, the standard Batch Normalization does not satisfy our needs, since the normalization is done using the per-channel mean and variance. Instead, we would like to normalize the feature maps in a way that would produce the mean-centered rectangular patch matrix. In other words, we seek to standardize each non-overlapping $k \times k \times d_{l-1}$ patch with the mean and standard deviation of all such patches (or alternatively, all patches in a mini-batch). Therefore, to account for mean-centering in Convolutional layers, we introduce a new normalization layer that we further refer to as Vector Batch Normalization. We extract all non-overlapping $k \times k \times d_{l-1}$ dimensional patches from the input appropriately padded with zeros. Further, each patch is flattened to a $1 \times k^2 d_{l-1}$ vector and the mean and variance are calculated from the resulting $NN_p \times k^2 d_{l-1}$ data matrix. The feature maps are then normalized as follows:

$$\hat{\mathbf{x}}_i^k = \frac{\mathbf{x}_i^k - \boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\boldsymbol{\sigma}_{\mathcal{B}}^2 + \epsilon}}, \tag{11}$$

$$\mathbf{y}_i^k = \gamma \hat{\mathbf{x}}_i^k + \beta, \tag{12}$$

where $\mathbf{x}_i^k$ is the i^{th} flattened patch, $\boldsymbol{\mu}_{\mathcal{B}}$ and $\boldsymbol{\sigma}_{\mathcal{B}}^2$ are the $1 \times k^2 d_{j-1}$-dimensional mean and variance vectors of the vectorized patches in the minibatch, and γ and β are the learnt parameters controlling the scale and offset, initialized as

[3]Least-squares regression to class labels can also be applied to initialize the last layer. However, we observed that in most cases random initialization of the output layer results in better generalization performance of the models during the subsequent training using backpropagation.

Algorithm 3: Initialization of l^{th} Convolutional layer

Function `VectorBNorm`($X^{(l)}$,f_size)**:**

  Input: $X^{(l)} : N \times S1 \times S2 \times D$ data representation at l^{th} layer; f_size : filter size

  $X^{(l)} \leftarrow$ Zero-pad $X^{(l)}$ to shape divisible by f_size

  $X_{fl}^{(l)} \leftarrow$ Extract and vectorize all $f\_size \times f\_size$ non-overlapping patches from $X^{(l)}$

  $\mu \leftarrow \text{mean}(X_{fl}^{(l)})$; $\sigma \leftarrow \text{var}(X_{fl}^{(l)})$;

  for *patch* $\leftarrow$ iterate through *all non-overlapping* $f\_size \times f\_size$ *patches in* $X^{(l)}$ **do**

    $patch \leftarrow \frac{flatten(patch)-\mu}{\sqrt{\sigma+\epsilon}}$

    $patch \leftarrow$ Reshape patch to $(f\_size \times f\_size \times D)$

  end

  return $X^{(l)}$

Function `conv_init`($X^{(l)}$,y,N_filt, f_size)**:**

  Input: $X^{(l)} : N \times S1 \times S2 \times D$ data representation at l^{th} layer; $y : N \times 1$ vector with class labels; N_filt :number of filters; f_size :filter size;

  $Z \leftarrow \text{ceil}(N\_filt/C)$

  $X^{(l)} \leftarrow \text{VectorBNorm}(X^{(l)}, f\_size)$

  $X_{fl}^{(l)} \leftarrow$ Extract and vectorize all $f\_size \times f\_size$ non-overlapping patches from $X^{(l)}$

  $y_{cl} \leftarrow \text{Cluster}(X_{fl}^{(l)}, Z)$

  $T^{(l)} \leftarrow \text{getTargets}(y, y_{cl}, C, Z, N, D)$

  if $D < N$ **then**

    $R \leftarrow (\text{chol}(X_{fl}^{(l)} X_{fl}^{(l)T}))^{-1}$

    $W^{(l)} \leftarrow R R^T X_{fl}^{(l)} T^{(l)T}$

  else

    $R \leftarrow (\text{chol}(X_{fl}^{(l)T} X_{fl}^{(l)}))^{-1}$

    $W^{(l)} \leftarrow X_{fl}^{(l)} R^T R T^{(l)T}$

  end

  $W^{(l)} \leftarrow$Select first N_filt dimensions of $W^{(l)}$ and normalize with $l2$ norm

  $\hat{W}^{(l)} \leftarrow$Reshape $W^{(l)}$ to $(N\_filt, f\_size, f\_size, D)$

  return $\hat{W}^{(l)}$

1 and 0, respectively. Similarly to conventional Batch Normalization, moving mean and moving variance are estimated for normalization during inference.

4. Experimental setup

In order to evaluate the proposed network initialization approach, we ran experiments on three image classification datasets: CIFAR-10 [32], MNIST [32], and Linnaeus-5 [33]. CIFAR-10 dataset contains images of 32 × 32 pixels with 3 channels and 10 object categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. MNIST dataset contains grayscale images of size 28×28 posing a handwritten digit recognition problem. Linnaeus-5 dataset contains RGB images of 32×32 dimensionality of 5 object categories: berry, bird,

dog, flower, and other. We use the provided train-test splits for evaluation. In CIFAR-10 dataset, training set is split into 48,000 images used for training, and 12,000 for validation. In MNIST dataset, training set is split into 40,000 and 10,000 images for training and validation, respectively. In both datasets, 10,000 images are used for testing. In Linneaus-5 datasets, 4,800 images are used for training, 1,200 for validation, and 2,000 for testing. Sample images from each dataset are shown in Fig. 1. Additionally, we employ two non-image datasets: CovType [34] and KDD [35]. CovType poses the task of classification of forest cover type from cartographic variables, having 7 classes with 54 attributes. We utilize 348,612 samples for training, and 116,200 for testing and validation. The KDD dataset poses the task of classification of network traffic into different types of attacks. The dataset has 23 classes and 41 attributes. We utilize a subset of 296,431 samples for training, and 98,795 for validation and testing.

Figure 1: Examples of dataset images from Linnaeus-5 (top), MNIST (middle) and CIFAR-10 (bottom) datasets.

In this work, we focus on settings that require small models, as well as on settings where large datasets might not be available, and hence data-dependant initialization is especially required for model performance. Experiments with initialization of deeper models are therefore left for future work. We evaluate our approach on two CNN architectures with 5 and 6 hidden layers, and MLPs with 4 and 5 hidden layers. Recall that following the proposed methodology, the maximum number of neurons in MLPs or filters in CNNs at a certain layer is equal to $\sum_{i=1}^{C} K_i$, where K_i is the number of subclasses for class i. We set $K_i = Z$ subclasses for all classes, leading to $CZ-1$ neurons in MLPs or filters in CNNs at a certain layer. In our experimental setup we construct the networks starting from 16 or 32 subclasses and reducing the number of subclasses by a factor of 2 with each subsequent layer. This results in two architectures with the layers having width of$\{319, 159, 79, 39, 19\}$ or $\{159, 79, 39, 19\}$ neurons/filters for MNIST and CIFAR datasets, and $\{159, 79, 39, 19, 9\}$ or $\{79, 39, 19, 9\}$ neurons/filters for Linnaeus-5 dataset. In CNN case, another fully-connected layer of 128 neurons is added after the last convolutional layer, initialized following Algorithm 2. The output layer consists of 5 or 10 neurons depending on the

dataset, and a softmax activation function.

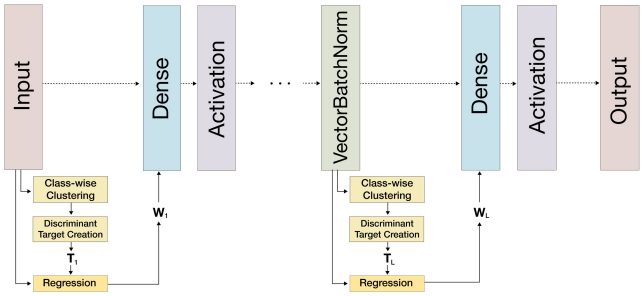

Figure 2: Structure of fastSDA-initialized Dense network

The overall architecture structure for MLPs is outlined in Fig. 2. We apply an activation function after each Dense layer, and a Batch Normalization layer before each Dense layer except the output layer (assuming the input data is standardized). The overall structure of the CNN architectures is shown in Fig. 3. We apply a Vector Batch Normalization layer before every convolution layer, followed by Max Pooling and Activation. After the last convolutional block, data is flattened and Batch Normalization is applied, followed by a Dense layer with 128 neurons, an activation function, and an output layer with softmax activation. For all the networks we perform experiments with three commonly-used activation functions: ReLU, LeakyReLU with α=0.3, and Tanh (hyperbolic tangent). The output layer is initialized randomly from a Gaussian distribution with zero mean and standard deviation equal to 0.05. In CNN, the bias terms are omitted in all models, and in MLPs they are initialized from zeros. To obtain the cluster labels during fastSDA initialization, mini-batch k-means clustering is performed [36].

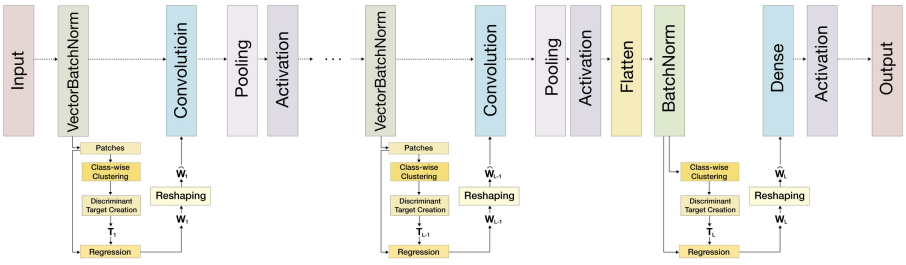

Figure 3: Structure of fastSDA-initialized CNN

In MLPs we compare the proposed initialization approach with random initialization from Gaussian distribution with $\mu = 0$ and $\sigma = 0.05$ (RNorm), random initialization from uniform distribution in the range $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ (RUni),

where n is the number of input neurons in the corresponding layers. We also provide comparisons with Glorot initialization [8] and He initialization [9]. We also compare the results with data-driven approaches by substituting the fastSDA step with either K-Means initialization (KM), LDA, or PCA. For K-Means initialization, we whiten the data and apply spherical clustering into n clusters, subsequently initializing each neuron with one of the cluster centroids following [37]. In LDA and PCA initialization, we initialize the neurons to the eigenvectors of the corresponding weight matrices, similarly to [15, 24]. Since LDA and PCA can return at maximum $C-1$ and D eigenvectors, respectively, in the case that the number of eigenvectors corresponding to non-zero eigenvalues are lower than the number of neurons required by the architecture, we initialize them randomly from a Gaussian distribution with zero mean and standard deviation of 0.05. In LDA and PCA, eigenvector matrices are normalized such that the $\updownarrow-2$ norm of each column is equal to 1, similarly to the proposed approach, to ensure that any difference in performance arises from the utilized statistical learning method rather than from normalization. The output layers are initialized randomly, similarly to our proposed approach. All the initialization methods are evaluated on the same architectures as the proposed approach.

Similarly, in CNN, we compare the proposed initialization approach with Glorot initialization [8], He initialization [9], random Gaussian and random uniform distributions with the parameters similar to the ones utilized in MLPs, K-Means initialization, and PCA initialization. We use the same architecture as shown in Fig. 3 for all initialization methods. Besides, for random initializations, He, and Glorot methods we provide the results for the architectures where Vector Batch Normalization is replaced with conventional Batch Normalization, to ensure that the accuracy gain obtained with the proposed approach does not result solely from the new normalization layer.

It can be noted that the patch extraction in the initialization of CNN results in a significant increase in the number of data samples used to learn the projection space, which might lead to undesired overhead during the clustering step of the proposed approach. As a remedy for this, we show that a small number of samples is generally sufficient to learn a good projection space that leads to competitive performance. To showcase this, we provide the results in which only a limited number of training samples is used during the initialization step. Specifically, we test the proposed approach with 200 and 500 samples per class (i.e., the total of 2000 or 5000 samples in CIFAR-10 and MNIST, and 1000 or 2500 samples in Linnaeus-5 dataset). Besides, we also evaluate the methods without utilization of any type of Batch Normalization. In this case, normalization of data is also not performed during learning of the projection space initialization of the weights, and the solution is, therefore, approximate.

We train the models with Stochastic Gradient Descent with a learning rate of 0.001, a batch size of 32, and categorical cross-entropy as the loss function until the accuracy on the validation set stops improving for 10 epochs. The model that resulted in the best validation accuracy is then used for reporting the results on the test set. Data in MLP experiments is standardized and images in CNN experiments are mean-centered and rescaled to match the range of 0 to

Table 1: Accuracies on MLP architectures

Linnaeus-5

	LReLU_{16}	ReLU_{16}	Tanh_{16}	LReLU_{32}	ReLU_{32}	Tanh_{32}
RNorm	37.25	31.30	30.40	29.55	32.50	33.60
RUni	35.05	35.25	31.35	30.95	30.95	32.65
He	36.70	32.45	32.95	38.55	29.65	32.00
Glorot	**39.55**	31.35	33.90	37.00	30.60	36.25
KM	38.65	34.70	35.60	**40.75**	31.00	32.40
LDA	32.65	32.70	33.00	37.25	31.40	33.00
PCA	32.95	34.60	33.90	33.30	31.90	35.65
fSDA	39.10	38.10	36.80	38.60	**40.35**	36.50
fSDA_{500}	37.05	**38.45**	**37.00**	37.80	37.80	**37.05**
fSDA_{200}	36.30	35.30	34.40	35.20	30.20	34.05

CIFAR-10

	LReLU_{16}	ReLU_{16}	Tanh_{16}	LReLU_{32}	ReLU_{32}	Tanh_{32}
RNorm	47.46	45.69	39.78	47.10	43.28	38.36
RUni	45.68	42.09	36.77	45.42	42.49	38.12
He	47.26	43.89	40.30	46.72	41.65	39.00
Glorot	47.16	44.61	42.21	47.14	41.66	40.86
KM	47.15	45.81	42.35	48.47	45.44	41.81
LDA	46.09	44.59	40.44	46.44	43.46	39.69
PCA	**48.85**	45.04	40.66	47.29	42.91	40.24
fSDA	48.20	46.32	**44.51**	47.97	48.18	**43.49**
fSDA_{500}	48.56	**47.32**	43.65	**48.77**	**48.48**	42.66
fSDA_{200}	47.55	46.19	43.55	46.93	46.40	42.62

MNIST

	LReLU_{16}	ReLU_{16}	Tanh_{16}	LReLU_{32}	ReLU_{32}	Tanh_{32}
RNorm	96.54	96.44	95.78	96.42	96.26	96.06
RUni	95.85	95.11	93.59	96.33	95.70	94.46
He	96.33	96.06	95.44	96.22	95.18	95.29
Glorot	96.85	96.38	96.15	96.55	96.13	95.80
KM	96.52	96.38	96.17	96.76	96.25	96.29
LDA	96.12	95.75	95.98	96.44	95.79	95.71
PCA	96.72	96.44	96.11	96.59	96.10	95.67
fSDA	96.85	96.56	**96.34**	96.72	96.70	96.35
fSDA_{500}	96.81	**96.89**	**96.34**	96.95	**97.29**	96.53
fSDA_{200}	**97.22**	96.72	95.92	**97.12**	96.68	**96.83**

1.

4.1. Results

The accuracy for MLP models with different initialization methods is shown in Table 1 and Table 2, where we report results on three activation functions and two architectures, i.e., LReLU_{16} stands for architecture corresponding to 16 subclasses as described earlier and Leaky ReLU activation function. Similarly, Tables 3 and 4 show the accuracies for CNNs without and with normalization layers, respectively. The best accuracy is highlighted in bold.

Table 2: Classification results of linear methods in COVTYPE and KDD datasets

COVTYPE

	ReLU16	LReLU16	Tanh16	LReLU32	ReLU32	Tanh32
RNorm	58.81	60.11	63.65	57.35	52.08	52.14
RUni	59.19	59.25	60.37	54.08	59.90	56.33
He	59.61	61.48	59.15	54.84	52.22	54.37
Glorot	63.42	59.96	63.88	53.68	51.29	53.71
KM	59.80	63.01	61.46	55.91	59.35	55.05
LDA	60.68	60.20	**65.07**	53.42	56.01	54.27
PCA	59.89	59.24	64.13	52.51	56.10	56.91
fSDA	**63.97**	57.42	63.14	**57.65**	**60.83**	54.60
fSDA500	61.39	**63.84**	62.58	51.54	57.06	55.73
fSDA200	63.66	62.65	63.13	56.84	53.29	**58.04**

KDD

	ReLU16	LReLU16	Tanh16	LReLU32	ReLU32	Tanh32
RNorm	97.10	95.88	96.79	98.96	97.88	99.13
RUni	98.48	96.75	98.34	97.73	97.52	99.05
He	90.29	94.28	97.92	97.48	97.70	**99.20**
Glorot	96.98	96.78	98.31	97.62	97.95	9823
KM	96.88	95.99	**98.68**	98.43	97.70	98.00
LDA	97.12	96.05	98.36	97.92	98.12	98.94
PCA	97.19	96.63	97.73	98.85	96.64	99.17
fSDA	96.62	96.86	97.56	98.18	97.31	98.87
fSDA500	**98.73**	**98.08**	96.96	**99.17**	**98.42**	98.05
fSDA200	97.30	96.68	98.50	99.03	97.67	98.46

Table 3: CNN results without normalization

Linnaeus-5

	LReLU_{16}	ReLU_{16}	Tanh_{16}	LReLU_{32}	ReLU_{32}	Tanh_{32}
RNorm	63.05	55.70	60.85	57.40	60.00	59.10
RUni	48.15	50.95	54.65	52.55	50.40	56.30
He	55.70	55.95	61.85	58.85	56.15	58.60
Glorot	61.85	**61.75**	60.40	61.50	60.90	62.10
KM	61.90	52.00	46.50	63.25	63.85	45.45
PCA	64.50	54.05	59.35	61.30	**64.70**	62.15
fSDA	**64.75**	48.85	**62.10**	**64.55**	61.90	**63.55**

CIFAR-10

	LReLU_{16}	ReLU_{16}	Tanh_{16}	LReLU_{32}	ReLU_{32}	Tanh_{32}
RNorm	68.54	68.29	67.08	66.35	65.40	65.67
RUni	64.02	62.02	66.53	69.01	68.65	70.92
He	68.33	67.75	70.68	68.77	69.25	**72.61**
Glorot	70.27	**70.49**	71.00	71.01	70.93	71.28
KM	70.46	69.75	70.72	72.25	71.41	70.70
PCA	70.31	70.20	70.70	71.87	70.62	71.10
fSDA	**71.10**	69.83	**71.01**	**72.66**	**71.52**	70.46

MNIST

	LReLU_{16}	ReLU_{16}	Tanh_{16}	LReLU_{32}	ReLU_{32}	Tanh_{32}
RNorm	98.99	98.96	99.23	98.81	98.98	99.18
RUni	98.78	99.01	99.13	98.93	99.05	**99.35**
He	98.85	98.92	99.24	98.91	98.86	99.30
Glorot	98.79	98.79	99.09	99.00	98.68	99.24
KM	99.03	99.02	99.21	99.01	99.14	99.10
PCA	99.01	**99.13**	99.27	98.97	99.08	99.18
fSDA	**99.05**	99.03	**99.28**	**99.08**	**99.17**	99.26

As can be seen from Table 1, in the majority of architectures and datasets, the proposed initialization outperforms other competing methods in terms of accuracy. In the CNN scenario, the proposed approach often outperforms competing methods already without considering mean-centering and the use of any type of normalization layers. We can see that mean-centering of the data during the initialization and the subsequent use of VectorBatchNormalization layers result in improved accuracy even further in the vast majority of the scenarios. Note that such normalization also leads to improved accuracy of PCA and K-Means initialization in most of the cases.

Considering the initialization using smaller number of samples, we observe that in the CNNs, both 200 and 500 samples are often sufficient for outperforming the competing methods (in the case fSDA_{200} or fSDA_{500} outperforms competing methods except fSDA, it is underlined in the tables). Considering the MLP initialization, the results with regard to initialization with a smaller number of samples are rather similar to that of CNN and the use of a small number of samples generally leads to a fair performance. Another fact worth

Table 4: CNN results with normalization layers

Linnaeus-5

		LReLU_{16}	ReLU_{16}	Tanh_{16}	LReLU_{32}	ReLU_{32}	Tanh_{32}
BNorm	RNorm	52.55	51.45	48.80	51.75	50.10	46.15
	RUni	59.00	55.25	49.65	58.50	58.30	47.70
	He	55.00	52.20	50.55	57.30	53.95	50.10
	Glorot	54.50	54.95	53.00	57.20	56.35	52.60
VecBNorm	RNorm	49.55	47.05	44.85	50.80	47.70	41.85
	RUni	53.90	51.55	44.30	51.85	52.90	43.80
	He	55.15	53.40	50.75	57.20	54.10	51.15
	Glorot	56.40	52.90	53.00	58.85	57.05	52.90
	KM	60.70	62.40	60.85	61.55	58.70	57.65
	PCA	60.90	62.90	60.00	**63.90**	60.35	59.75
	fSDA	**64.25**	62.30	61.75	59.75	62.70	**61.75**
	fSDA_{500}	62.00	**64.35**	**62.10**	61.45	**64.20**	61.70
	fSDA_{200}	60.65	62.90	59.70	60.95	64.05	59.20

CIFAR-10

		LReLU_{16}	ReLU_{16}	Tanh_{16}	LReLU_{32}	ReLU_{32}	Tanh_{32}
BNorm	RNorm	66.17	63.67	59.35	64.89	62.05	63.78
	RUni	69.50	69.38	62.32	71.66	70.23	64.88
	He	68.49	68.74	65.89	70.18	71.34	64.57
	Glorot	71.71	72.04	68.41	73.77	73.51	66.94
VecBNorm	RNorm	63.65	61.89	59.96	64.50	60.68	62.71
	RUni	64.99	65.71	54.70	67.31	66.65	56.32
	He	69.17	68.17	64.24	71.54	70.11	64.88
	Glorot	71.45	71.75	65.79	73.73	72.88	68.56
	KM	72.03	75.01	68.52	**77.18**	76.20	67.03
	PCA	72.67	74.40	68.06	72.71	77.17	71.65
	fSDA	**75.02**	**75.36**	**70.59**	76.66	**77.79**	71.87
	fSDA_{500}	74.13	74.35	69.80	71.29	76.22	**72.60**
	fSDA_{200}	70.32	74.57	69.35	75.71	77.33	71.39

MNIST

		LReLU_{16}	ReLU_{16}	Tanh_{16}	LReLU_{32}	ReLU_{32}	Tanh_{32}
BNorm	RNorm	98.82	98.78	98.45	98.94	98.70	98.41
	RUni	99.16	99.20	99.03	99.11	99.19	99.04
	He	99.00	99.17	99.03	99.19	99.30	99.10
	Glorot	99.10	99.30	99.23	99.22	**99.35**	99.24
VecBNorm	RNorm	98.76	98.35	98.30	98.76	98.63	98.19
	RUni	98.99	98.81	98.49	99.10	98.92	98.58
	He	99.03	99.14	98.95	99.13	99.14	98.87
	Glorot	**99.18**	99.21	99.08	99.15	99.16	99.22
	KM	99.10	99.07	99.12	99.21	99.20	99.18
	PCA	98.82	99.18	99.20	**99.25**	99.24	**99.34**
	fSDA	98.67	**99.26**	**99.24**	**99.25**	99.24	99.28
	fSDA_{500}	98.98	99.14	99.17	97.92	99.13	99.10
	fSDA_{200}	98.65	99.04	95.16	99.17	99.18	99.05

noticing is that in a few cases, the use of a smaller number of samples leads to performance improved compared to using the full dataset. A possible interpretation of this is that the model trained on a smaller number of samples overfits less to the training data, thus providing better generalization properties.

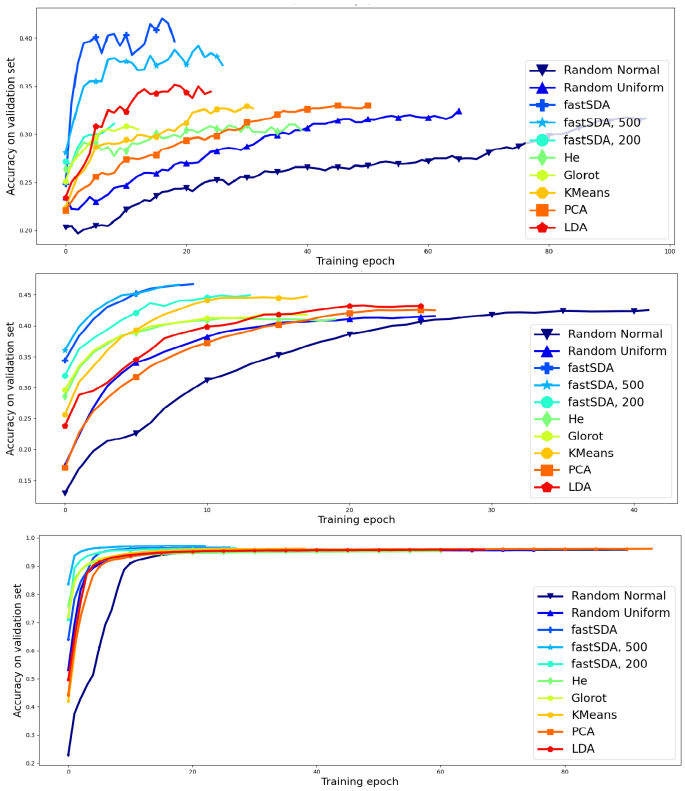

Figure 4: Convergence plots on MLPs. Datasets top to bottom: Linnaeus-5, CIFAR-10, MNIST

Figures 4 and 5 show the convergence speed of different methods, where we plot the accuracy on the validation set versus the number of training epochs. For the sake of variety, we provide the results on architectures corresponding to 32 subclasses and ReLU activation function for MLP architectures, and 16 subclasses and LeakyReLU activation function for CNN architecture. The plots outline several essential points: we observe that fastSDA-initialized models generally start from a higher accuracy compared to other methods, and generally they also take less epochs to converge. This is clearly seen especially on the MLP architectures. In addition, we can see that utilization of a larger number of samples for initialization results in a higher initial accuracy and a faster convergence compared to using a smaller number of samples. In CNN, we observe that the convergence properties are not as good as in the MLP case, and our proposed methods are mostly doing on-par with competing ones. However, this

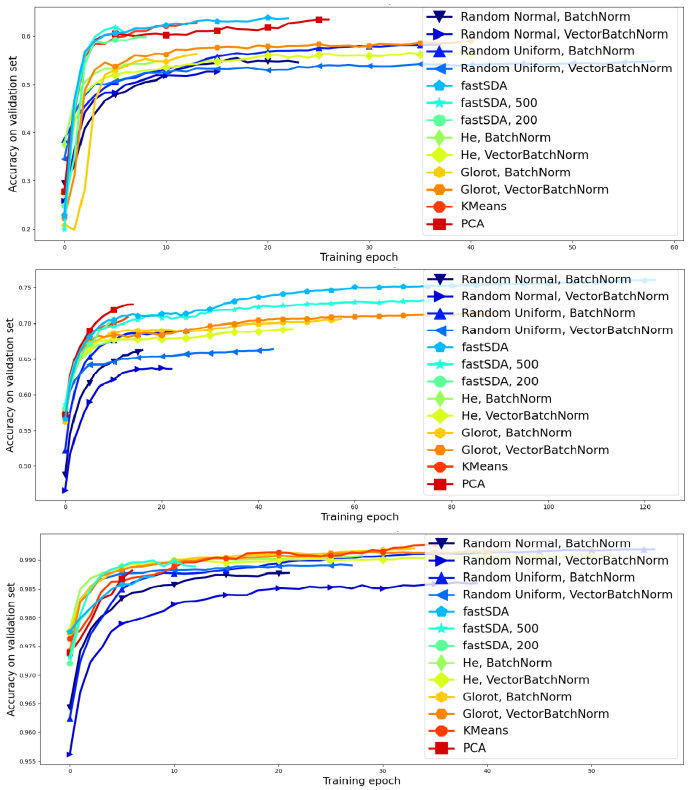

Figure 5: Convergence plots on CNNs. Datasets top to bottom: Linnaeus-5, CIFAR-10, MNIST

is compensated by the fact that our methods are able to achieve a better overall accuracy, and a more detailed investigation on the convergence properties of CNNs is left as a future work. Overall, these observations support our intuition that fastSDA initialization allows to start the optimization process from a more favourable point in the feature space.

For reference, we provide the initialization times in seconds for larger architecture corresponding to 32 subclasses for MLPs and CNNs in Table 5. As can be seen, the speed of initialization depends both on dimensionality and dataset size (recall that MNIST has 1 channel unlike CIFAR-10 and Linnaeus-5 that have 3 channels, and Linnaeus-5 is the smallest dataset). In MLPs, the overhead created by clustering plays a bigger role compared to dimensionality of data, leading to fastSDA with full training data being slower than PCA. However, in CNN and when using a smaller number of images for initialization, our approach is generally faster.

Table 5: Times for initialization in 32-subclass architecture (seconds)

MLP

	KM	LDA	PCA	fSDA	fSDA_{500}	fSDA_{200}
CIFAR	108	1251	25	108	42	26
MNIST	73	68	3	41	11	8
LIN	23	176	23	25	16	8

CNN

	KM	PCA	fSDA	fSDA_{500}	fSDA_{200}
CIFAR	22020	12364	6315	807	532
MNIST	16301	6158	4516	521	294
LIN	958	1208	369	216	152

5. Conclusion

In this paper we proposed a novel data-driven approach for weight initialization based on discriminant learning. The proposed initialization was formulated for dense and convolutional layers appearing in Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). In addition, we considered some of the limitations of the method caused by assumptions on the data and proposed ways to remedy them. Experimental results show that the proposed approach provides several benefits compared to competing ones, including improved training accuracy and initial accuracy, while achieving equal or faster convergence and initialization time. In addition, we showed that the initialization time can be improved even further by applying the initialization based on a small number of samples with no degrading effect on accuracy.

Acknowledgment

This work is supported by Business Finland under project 5G Vertical Integrated Industry for Massive Automation (5G-VIIMA). A. Iosifidis acknowledges funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337 (MARVEL).

References

[1] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, Q. Tian, Centernet: Keypoint triplets for object detection, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 6569–6578.

[2] Z.-Q. Zhao, P. Zheng, S.-t. Xu, X. Wu, Object detection with deep learning: A review, IEEE transactions on neural networks and learning systems 30 (11) (2019) 3212–3232.

[3] K. Chumachenko, A. Männistö, A. Iosifidis, J. Raitoharju, Machine learning based analysis of finnish world war ii photographers, IEEE Access 8 (2020) 144184–144196.

[4] S. Yun, J. Choi, Y. Yoo, K. Yun, J. Y. Choi, Action-driven visual object tracking with deep reinforcement learning, IEEE Transactions on Neural Networks and Learning Systems 29 (6) (2018) 2239–2252.

[5] C. Li, W. Xia, Y. Yan, B. Luo, J. Tang, Segmenting objects in day and night: Edge-conditioned cnn for thermal image semantic segmentation, IEEE Transactions on Neural Networks and Learning Systems (2020).

[6] X. Chen, J. Weng, W. Lu, J. Xu, J. Weng, Deep manifold learning combined with convolutional neural networks for action recognition, IEEE transactions on neural networks and learning systems 29 (9) (2017) 3938–3952.

[7] A. Iosifidis, A. Tefas, I. Pitas, "view-invariant action recognition based on artificial neural networks, IEEE Transactions on Neural Networks and Learning Systems 23 (3) (2012) 412–424.

[8] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, 2010, pp. 249–256.

[9] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.

[10] P. Krähenbühl, C. Doersch, J. Donahue, T. Darrell, Data-dependent initializations of convolutional neural networks, arXiv preprint arXiv:1511.06856 (2015).

[11] C.-C. J. Kuo, M. Zhang, S. Li, J. Duan, Y. Chen, Interpretable convolutional neural networks via feedforward design, Journal of Visual Communication and Image Representation 60 (2019) 346–359.

[12] Y. Chen, Y. Yang, M. Zhang, C.-C. J. Kuo, Semi-supervised learning via feedforward-designed convolutional neural networks, in: 2019 IEEE International Conference on Image Processing (ICIP), IEEE, 2019, pp. 365–369.

[13] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, Y. Ma, Pcanet: A simple deep learning baseline for image classification?, IEEE transactions on image processing 24 (12) (2015) 5017–5032.

[14] Y. Ge, J. Hu, W. Deng, Pca-ldanet: A simple feature learning method for image classification, in: 2017 4th IAPR Asian Conference on Pattern Recognition (ACPR), IEEE, 2017, pp. 370–375.

[15] M. Alberti, M. Seuret, V. Pondenkandath, R. Ingold, M. Liwicki, Historical document image segmentation with lda-initialized deep neural networks, in: Proceedings of the 4th International Workshop on Historical Document Imaging and Processing, 2017, pp. 95–100.

[16] R. Duda, P. Hart, D. Stork, Pattern Classification, 2nd Edition, Wiley, New York, NY, USA, 2000.

[17] R. O. Duda, P. E. Hart, D. G. Stork, Pattern classification, John Wiley & Sons, 2012.

[18] D. Wang, M. Li, Stochastic configuration networks: Fundamentals and algorithms, IEEE transactions on cybernetics 47 (10) (2017) 3466–3479.

[19] M. Rosenstein, Z. Marx, L. Kaelbling, T. Dietterich, To transfer or not to transfer, in: Neural Information Processing Workshop on Transfer Learning, 2005, pp. 1–4.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database, in: CVPR09, 2009.

[21] K. He, R. Girshick, P. Dollár, Rethinking imagenet pre-training, in: Proceedings of the IEEE international conference on computer vision, 2019, pp. 4918–4927.

[22] A. Coates, A. Y. Ng, Learning feature representations with k-means, in: Neural networks: Tricks of the trade, Springer, 2012, pp. 561–580.

[23] J. Mairal, P. Koniusz, Z. Harchaoui, C. Schmid, Convolutional kernel networks, in: Advances in neural information processing systems, 2014, pp. 2627–2635.

[24] M. Seuret, M. Alberti, M. Liwicki, R. Ingold, Pca-initialized deep neural networks applied to document image analysis, in: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Vol. 01, 2017, pp. 877–882.

[25] D. Cai, X. He, J. Han, Training linear discriminant analysis in linear time, in: 2008 IEEE 24th International Conference on Data Engineering, IEEE, 2008, pp. 209–217.

[26] K. Chumachenko, J. Raitoharju, A. Iosifidis, M. Gabbouj, Speed-up and multi-view extensions to subclass discriminant analysis, Pattern Recognition 111 (107660) (2020) 1–15.

[27] M. Zhu, A. Martinez, Subclass discriminant analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (2006).

[28] Y. Shuicheng, X. Dong, B. Zhang, H. Zhang, Q. Yang, S. Lin, Graph embedding and extensions: a general framework for dimensionality reduction, IEEE Transactions on Pattern Analysis and Machine Intelligence 29 (2007) 40–51.

[29] D. Cai, X. He, J. Han, Srda: an efficient algorithm for large scale discriminant analysis, IEEE Transactions on Knowledge and Data Engineering 20 (2007) 1–12.

[30] K. Chumachenko, M. Gabbouj, A. Iosifidis, Robust fast subclass discriminant analysis, in: European Signal Processing Conference, 2020.

[31] K. Chumachenko, J. Raitoharju, M. Gabbouj, A. Iosifidis, Incremental fast subclass discriminant analysis, in: International Conference on Image Processing, 2020.

[32] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images (2009).

[33] G. Chaladze, L. Kalatozishvili, Linnaeus 5 dataset for machine learning, Tech. rep., Tech. Rep (2017).

[34] J. A. Blackard, D. J. Dean, Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables, Computers and electronics in agriculture 24 (3) (1999) 131–151.

[35] J. Stolfo, W. Fan, W. Lee, A. Prodromidis, P. K. Chan, Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection, Results from the JAM Project by Salvatore (2000) 1–15.

[36] D. Sculley, Web-scale k-means clustering, in: Proceedings of the 19th international conference on World wide web, 2010, pp. 1177–1178.

[37] A. Coates, A. Y. Ng, Learning feature representations with k-means, in: Neural networks: Tricks of the trade, Springer, 2012, pp. 561–580.