Journal of Machine Learning Research 22 (2021) 1-28 Submitted 2/21; Revised 8/21; Published 10/21

Non-linear, Sparse Dimensionality Reduction via Path Lasso Penalized Autoencoders

Oskar Allerbo ALLERBO@CHALMERS.SE
Mathematical Sciences
University of Gothenburg and Chalmers University of Technology
SE-412 96 Gothenburg, Sweden

Rebecka Jörnsten JORNSTEN@CHALMERS.SE
Mathematical Sciences
University of Gothenburg and Chalmers University of Technology
SE-412 96 Gothenburg, Sweden

Editor: Pradeep Ravikumar

Abstract

High-dimensional data sets are often analyzed and explored via the construction of a latent low-dimensional space which enables convenient visualization and efficient predictive modeling or clustering. For complex data structures, linear dimensionality reduction techniques like PCA may not be sufficiently flexible to enable low-dimensional representation. Non-linear dimension reduction techniques, like kernel PCA and autoencoders, suffer from loss of interpretability since each latent variable is dependent of all input dimensions. To address this limitation, we here present path lasso penalized autoencoders. This structured regularization enhances interpretability by penalizing each *path* through the encoder from an input to a latent variable, thus restricting how many input variables are represented in each latent dimension. Our algorithm uses a group lasso penalty and non-negative matrix factorization to construct a sparse, non-linear latent representation. We compare the path lasso regularized autoencoder to PCA, sparse PCA, autoencoders and sparse autoencoders on real and simulated data sets. We show that the algorithm exhibits much lower reconstruction errors than sparse PCA and parameter-wise lasso regularized autoencoders for low-dimensional representations. Moreover, path lasso representations provide a more accurate reconstruction match, i.e. preserved relative distance between objects in the original and reconstructed spaces.

Keywords: Sparse Dimensionality Reduction, Non-linear Dimensionality Reduction, Regularized Neural Networks, Group Lasso, Autoencoders

1. Introduction

Dimensionality reduction is a key component in data compression, data visualization and feature extraction. One of the most widely used techniques is principal component analysis (PCA), that uses the eigendecomposition of the sample covariance matrix to construct latent dimensions as linear combinations of the original dimensions. The interpretability of the latent representation is increased if each latent dimension consists only of a subset of the original dimensions, as in sparse PCA (Zou et al., 2006). Since the introduction of sparse PCA, several variants have been presented, such as non-negative sparse PCA (Zass

and Shashua, 2007), where the loadings are all non-negative; multilinear sparse PCA (Lai et al., 2014), that operates on tensors instead of vectors; and robust PCA (Meng et al., 2012; Croux et al., 2013), that is less affected by outliers.

While more interpretable than standard PCA, sparse PCA and its variants are still linear and therefore cannot capture more complex relations in the data. Furthermore, they have limitations on how efficiently they make use of the latent dimensions, something that is especially important when the number of latent dimensions is small. Several non-linear generalizations of PCA exist, the most important ones being kernel PCA (Schölkopf et al., 1998) and autoencoders (Kramer, 1991). In kernel PCA, a kernel function is used to implicitly and non-linearly map the data to a space with higher dimensionality than d_x (the original dimensionality) in which linear PCA is performed. Autoencoders use an hourglass shaped neural network with d_x input and output nodes and d_z (the latent dimensionality) nodes in the middle layer. The same data is used both as input and output, so the goal of the autoencoder is to reconstruct the original data from a lower dimensional representation, that is found in the middle layer.

Although there do exist several algorithms for sparse kernel PCA, see work by Tipping (2001), Smola et al. (2002), Wang and Tanaka (2016) and Guo et al. (2019), as well as for sparse autoencoders (e.g. Ng et al., 2011), the terminology might be a little confusing, since the algorithms are sparse in a different sense than sparse PCA. Instead of being sparse in the sense original to latent dimensions, they are sparse in the sense observations to latent dimensions, which means that each observation is active only in a subset of the latent dimensions (and vice versa, each latent dimension depends only on a subset of the data). To further illustrate this, we look at the linear case, with d_x and d_z as before and with n being the number of observations. For $\boldsymbol{Z} \in \mathbb{R}^{n\times d_z}$, $\boldsymbol{X} \in \mathbb{R}^{n\times d_x}$ and $\boldsymbol{W} \in \mathbb{R}^{d_x\times d_z}$, the latent representation $\boldsymbol{Z}$ is obtained from the original representation $\boldsymbol{X}$ as

$$\boldsymbol{Z} = \boldsymbol{X} \cdot \boldsymbol{W}. \tag{1}$$

Then sparsity in the sense original to latent dimensions means that $\boldsymbol{W}$ is sparse, while sparsity in the sense observations to latent dimensions means that $\boldsymbol{Z}$ is sparse. A third notion of sparsity is feature selection, where only a subset of the original dimension are included in the model. This corresponds to entire rows of $\boldsymbol{W}$ being zero, allowing the remaining rows to be dense.

The sparse autoencoder generalizes Equation (1) to $\boldsymbol{Z} = \text{Enc}(\boldsymbol{X})$, where Enc is the non-linear encoder and, to produce a sparse $\boldsymbol{Z}$, it includes a constraint on how frequently a latent unit is allowed to be non-zero, where frequency is measured among observations. This means that the architecture of the encoder might still be dense, in contrast to sparsity corresponding to a sparse $\boldsymbol{W}$, which requires a sparse architecture.

Neural networks with sparse architectures are usually obtained using different versions of lasso, or l_1-, regularization (Tibshirani, 1996), and there are numerous examples of these. Scardapane et al. (2017) used group lasso (Yuan and Lin, 2006) to remove all the links to or from a node, while Yoon and Hwang (2017) combined group lasso with exclusive lasso (Zhou et al., 2010) on filters in convolutional neural networks to, on one hand, totally eliminate some filters (using group lasso) and, on the other hand, make filters as different as possible (using exclusive lasso). Lasso regularized autoencoders include work by Wu et al. (2019), with a linear encoder and a lasso regularized decoder; Dabin et al. (2020), with a lasso

regularized encoder and a linear decoder; and Ainsworth et al. (2018), which uses group lasso and variational autoencoders to split the original dimensions into pre-defined groups and then uses one decoder per group, with a shared latent space.

In this paper we propose path lasso, that uses group lasso regularization to eliminate all connections between two nodes in two non-adjacent layers of a fully connected feedforward neural network. We apply path lasso, in combination with exclusive lasso, to an autoencoder to introduce sparsity between original and latent variables, obtaining non-linear, sparse dimensionality reduction in the same sense as in sparse PCA. Path lasso forces each latent dimension to be a function only of a subset of the original dimensions, while exclusive lasso encourages these subsets to differ. To the best of our knowledge this is the first non-linear dimensionality reduction algorithm that is sparse in this sense.

The rest of this paper is organized as follows: In Section 2 we introduce the path lasso penalty and the path lasso penalized autoencoder. In Section 3 we run experiments on real and simulated data sets, comparing path lasso to PCA, autoencoders, sparse autoencoders, sparse PCA and an autoencoder with parameter-wise l_1-regularization. We show that, for a given sparsity, path lasso results in a lower reconstruction error and is better at reconstruction match, i.e. retaining relative positioning of objects in the reconstructed space as in the original space. We conclude with a discussion in Section 4.

2. Method

This section is structured in the following way: Sections 2.1 and 2.2 present short reviews of different flavours of the lasso algorithm and of proximal gradient descent, while Sections 2.3 to 2.6 describe different aspects of path lasso. Section 2.3 and 2.4 describe how paths between nodes in two non-adjacent layers are defined and penalized, and how the path penalties are transformed to individual link penalties. Section 2.5 discusses when and how to apply the path penalty and Section 2.6 describes how we adapt path lasso when using it in an autoencoder. In Appendix A we discuss different methods to accelerate training.

2.1 Review of Lasso Penalties

The lasso algorithm sets some model parameters exactly equal to zero, thus eliminating them from the model. There are different versions of the lasso, four of which are used in this paper. Here follows a short summary to these.

(Standard) Lasso applies an l_1-penalty to all the parameters in the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$, and is defined as

$$\lambda \|\boldsymbol{\theta}\|_1 = \lambda \sum_{i=1}^{d} |\theta_i|,$$

where $\lambda > 0$ is the regularization strength. Due to the non-differentiability of the absolute value at zero, some of the θ_i's are set to exactly zero and thus eliminated from the model.

Adaptive Lasso applies an individual l_1-penalty to all the parameters in the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$, and is defined as

$$\sum_{i=1}^{d} \lambda_i |\theta_i|,$$

where $\lambda_i := \frac{\lambda}{|\hat{\theta}_i^{\mathrm{R}}|^\gamma}$ for some $\gamma > 0$ (common practice is $\gamma = 2$), and $\hat{\theta}_i^{\mathrm{R}}$ is the ridge regression estimate of θ_i. The idea is that important parameters will have larger values of $\hat{\theta}_i^{\mathrm{R}}$ and thus be penalized less than unimportant parameters.

Group Lasso penalizes pre-defined groups of parameters together, which means that either all, or none, of the parameters in the group are set to zero. The group lasso penalty for a group $g \in \mathcal{G}$ is defined as

$$\lambda \|\boldsymbol{\theta_g}\|_2 = \lambda \sqrt{\sum_{i \in g} \theta_i^2},$$

where $\mathcal{G} = \{g_1, \dots g_G\}$ is a disjoint partition of the index set $\{1, \dots, d\}$, i.e. each g is a set of indices defining a group, and, for a given $\boldsymbol{\theta} \in \mathbb{R}^d$, $\boldsymbol{\theta_g}$ is a d-dimensional vector with components equal to $\boldsymbol{\theta}$ for indices within g and zero otherwise. The total group lasso penalty is then taken as the sum of the penalties over the different groups. As seen, for a given group g,

$$\sqrt{\sum_{i \in g} \theta_i^2} = 0 \iff \theta_i = 0 \ \forall i \in g.$$

The *Exclusive Lasso* can be seen as the opposite of the group lasso. Again, the parameters are split into pre-defined groups, but now the goal is instead to impose a similar number of non-zero parameters in every group. With g as before, its exclusive lasso penalty defined as

$$\lambda \|\boldsymbol{\theta_g}\|_1^2 = \lambda \left(\sum_{i \in g} |\theta_i| \right)^2,$$

and, again, the total penalty is defined as the sum over the groups. Since the number of mixed terms in the squared sum grows with the number of elements in the sum, the total number of mixed terms over all the groups is minimized when the non-zero elements are evenly distributed among the groups.

2.2 Review of Proximal Gradient Descent

The reason that some parameters in lasso penalized models are set exactly to zero, is that the derivative of the absolute value is not unique at zero: $\left.\frac{\partial |\theta|}{\partial \theta}\right|_{\theta=0} \in [-1, 1]$. Gradient descent methods are unable to use this non-uniqueness and as a consequence no parameters are set exactly to zero. Proximal gradient descent (Rockafellar, 1976) on the other hand takes the non-uniqueness into account, resulting in exact zeros.

If the objective function, $f(\boldsymbol{\theta})$, can be decomposed into $f(\boldsymbol{\theta}) = g(\boldsymbol{\theta}) + h(\boldsymbol{\theta})$, where g is differentiable (typically a reconstruction error) and h is not (typically a lasso regularization term), then a standard gradient descent step, followed by a proximal gradient descent step, is defined as

$$\boldsymbol{\theta}^{t+1} = \mathrm{prox}_{\alpha h}(\boldsymbol{\theta}^t - \alpha \nabla g(\boldsymbol{\theta}^t)),$$

where $\alpha > 0$ is the learning rate and prox is the proximal operator that depends on h.

For lasso, with $h(\boldsymbol{\theta}) = \lambda ||\boldsymbol{\theta}||_1 = \sum_i \lambda |\theta_i|$, the proximal operator decomposes componentwise and is, with $(x)^+ := \max(x, 0)$,

$$\mathrm{prox}_{\alpha h}(\theta_i) = \mathrm{sign}(\theta_i) \cdot (|\theta_i| - \alpha\lambda)^+. \tag{2}$$

I.e., each θ_i is additively shrunk towards zero, and once it changes sign it becomes exactly zero.

For group lasso, with $h(\boldsymbol{\theta}) = \lambda \sum_{g\in\mathcal{G}} ||\boldsymbol{\theta_g}||_2 = \lambda \sum_{g\in\mathcal{G}} \sqrt{\sum_{i\in g} \theta_i^2}$, where $\boldsymbol{\theta_g}$ are the θ_i's that belong to group g and $\mathcal{G}$ is the set of all groups, the proximal operator for θ_i is

$$\text{prox}_{\alpha h}(\theta_i) = \theta_i \cdot \left(1 - \frac{\alpha\lambda}{\sqrt{\sum_{j\in g_i} \theta_j^2}}\right)^+, \tag{3}$$

where g_i is the group that θ_i belongs to. Thus all members of the group are penalized equally and set to zero at the exact same time.

2.3 Path Penalties

Let $\{\boldsymbol{o_l}\}_{l=0}^L$, $\boldsymbol{o_l} \in \mathbb{R}^{d_l}$, denote the outputs of $L+1$ consecutive layers in a fully connected feedforward neural network, where the first and last layers are also denoted by $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively, i.e. $\boldsymbol{x} := \boldsymbol{o_0}$ and $\boldsymbol{y} := \boldsymbol{o_L}$. Then $\boldsymbol{y}$ depends on $\boldsymbol{x}$ as

$$\boldsymbol{y} = \Phi_L(\boldsymbol{W_L}\Phi_{L-1}(\boldsymbol{W_{L-1}}\Phi_{L-2}(\dots\Phi_1(\boldsymbol{W_1}\boldsymbol{x} + \boldsymbol{b_1})\dots) + \boldsymbol{b_{L-1}}) + \boldsymbol{b_L}), \tag{4}$$

where $\{\boldsymbol{W_l}\}_{l=1}^L$, $\boldsymbol{W_l} \in \mathbb{R}^{d_l \times d_{l-1}}$, are the weight matrices, $\{\boldsymbol{b_l}\}_{l=1}^L$, $\boldsymbol{b_l} \in \mathbb{R}^{d_l}$, the bias vectors, and $\{\Phi_l\}_{l=1}^L$ the (not necessarily identical) element-wise activation functions. Thinking of Equation (4) as a graph, each weight matrix element, $w_{i_1 i_2}^l := (\boldsymbol{W_l})_{i_1 i_2}$, corresponds to a link between two nodes in two adjacent layers

By combining links from multiple weight matrices, paths between nodes in non-adjacent layers can be constructed. Between a given node in layer $\boldsymbol{x}$, x_{i_0}, and a node in layer $\boldsymbol{y}$, y_{i_L}, there are in total $\prod_{l=1}^{L-1} d_l$ paths, each consisting of L links, see Figure 1 for an illustration. A path is broken if at least one of its links has value zero and to disconnect the two nodes, all paths between them need to be broken. Just as Neyshabur et al. (2015), we define the value of a path as the product of its absolute valued links; see Definition 1.

Definition 1 (Path Value) *For $k = 1, 2, \dots, \prod_{l=1}^{L-1} d_k$, where each k corresponds to a unique combination of the indices i_1 to i_{L-1}*

$$p_{i_L i_0}^k := |w_{i_L i_{L-1}^k}^L| \cdot |w_{i_{L-1}^k i_{L-2}^k}^{L-1}| \cdot \dots |w_{i_1^k i_0}^1|.$$

With this definition a broken path, where at least one link is zero, has the value zero. We further define a group of paths so that all paths connecting x_{i_0} to y_{i_L} form one group; then if all paths in the group are zero, the nodes are disconnected. Applying the group lasso proximal operator to path $p_{i_L i_0}^k$, belonging to group $g_{i_L i_0}$ then amounts to

$$p_{i_L i_0}^k \cdot \left(1 - \frac{\alpha\lambda}{\sqrt{\sum_{l\in g_{i_L i_0}} (p_{i_L i_0}^l)^2}}\right)^+, \tag{5}$$

which according to Proposition 3 can be written as

$$p_{i_L i_0}^k \cdot \left(1 - \frac{\alpha\lambda}{(\boldsymbol{W}_{\text{PL}})_{i_L i_0}}\right)^+$$

with $\boldsymbol{W}_{\text{PL}}$ according to Definition 2.

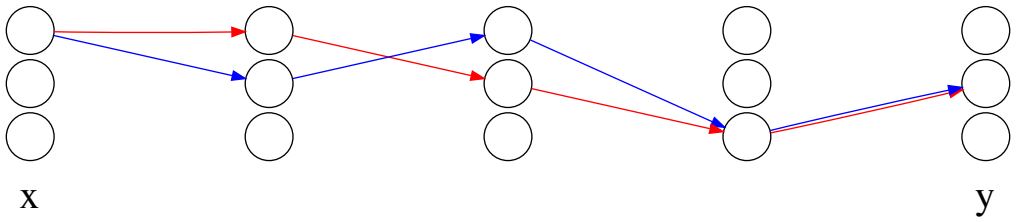


Figure 1: Illustration of two of the 3^3 (where the exponent is the number of inner layers and the base is their width) possible paths between node one in layer one and node two in layer five. Each arrow denotes a link, which corresponds to an element in a weight matrix.

Definition 2 (Path Lasso Matrix) *With the square and the square root taken element-wise,*

$$\boldsymbol{W}_{PL} := \sqrt{\prod_{l=L,\dots,1} (\boldsymbol{W_l})^2} = \sqrt{(\boldsymbol{W_L})^2 \cdot (\boldsymbol{W_{L-1}})^2 \cdot \ldots (\boldsymbol{W_1})^2}. \tag{6}$$

Proposition 3 *The element (i_L, i_0) in $\boldsymbol{W}_{PL}$ is the group lasso penalty on the group consisting of all paths between nodes x_{i_0} and y_{i_L}, i.e.*

$$(\boldsymbol{W}_{PL})_{i_L i_0} = \sqrt{\sum_{k \in g_{i_L i_0}} (p^k_{i_L i_0})^2}$$

For a proof, see Appendix B.

We call each element in $\boldsymbol{W}_{\text{PL}}$ a connection, i.e. $(\boldsymbol{W}_{\text{PL}})_{i_L i_0}$ is the strength of the connection between nodes x_{i_0} and y_{i_L} and if it is zero, meaning that all paths between the nodes are broken, then the nodes are disconnected. Proposition 4 gives a theoretical justification for this definition of connections, stating that if there is no connection between x_{i_0} and y_{i_L}, then the derivative of y_{i_L} with respect to x_{i_0} is zero, regardless of the value of $\boldsymbol{x}$.

Proposition 4 *Let the vector $\boldsymbol{y}$ depend on the vector $\boldsymbol{x}$ as stated in Equation* (4) *and let $\boldsymbol{W}_{PL}$ be the path lasso matrix defined in Equation* (6)*. Then, if all weights and activations are bounded,*

$$(\boldsymbol{W}_{PL})_{i_L i_0} = 0 \implies \frac{\partial y_{i_L}}{\partial x_{i_0}} = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)_{i_L i_0} = 0 \qquad \forall \boldsymbol{x} \in \mathbb{R}^{d_0}.$$

For a proof, see Appendix B.

2.4 From Paths to Links

Equation (5) describes how all paths between two nodes in the network are penalized towards zero, but it does not tell how this penalization affects the individual links. However, since the neural network is expressed in terms of the individual links, rather than in paths, the penalized paths must be translated into penalized links, i.e. the penalty must be expressed on link level, rather than on path level. Proposition 5 describes how this translation can be done, i.e. how applying the proximal group lasso operator to the paths can be translated into applying a standard lasso proximal operator to each individual link.

Proposition 5 *The group lasso proximal step in path space can be transformed to standard lasso proximal steps in link space by first solving the matrix equation*

$$\prod_{l=L,\dots,1} |(\boldsymbol{W_l})^t| \odot \left(1 - \frac{\alpha\lambda}{(\boldsymbol{W}_{PL})^t}\right)^+ = \prod_{l=L,\dots,1} \tilde{\boldsymbol{W}}_{\boldsymbol{l}} \tag{7}$$

for $\{\tilde{\boldsymbol{W}}_{\boldsymbol{l}}\}_{l=1}^{L}$*, and then set* $(\boldsymbol{W_l})^{t+1} := \text{sign}((\boldsymbol{W_l})^t) \odot \tilde{\boldsymbol{W}}_{\boldsymbol{l}}$*, where* $\tilde{w}_{ij}^l =: (|(w_{ij}^l)^t| - \alpha\lambda_{ij}^{l,t})^+$ *for some* $\lambda_{ij}^{l,t} > 0$*.*
The absolute value, sign and division are taken element-wise and $\odot$ *denotes element-wise multiplication.*

The proof, which is presented in Appendix B, contains a step where the paths in each group are summed over. This step transforms the system of non-linear equations from one equation per path to one equation per connection, i.e. from $\prod_{l=0}^{L} d_l$ to $d_0 \cdot d_L$ equations, and to a form can be solved efficiently using non-negative matrix factorization, NMF, with the extra requirement that $\tilde{w}_{i_l i_{l-1}}^l \leq |(w_{i_l i_{l-1}}^l)^t|$, or equivalently $\lambda_{i_l i_{l-1}}^{l,t} \geq 0$. We describe this algorithm in Appendix C.

The reduced number of equations leads to an undetermined system (unless the hidden layers are very narrow). Therefore, all equations can hold, although there might be more than one solution. Since we are interested in a solution that lies close to the unpenalized weight matrices, we use $|\boldsymbol{W_l}|$ as the seed for each $\tilde{\boldsymbol{W}}_{\boldsymbol{l}}$ in the matrix factorization.

An optimization step using proximal gradient descent on paths is summarized in Algorithm 1. The bottleneck of this algorithm is the matrix factorization step from Equation (7). In Appendix A, different methods to alleviate this bottleneck are discussed.

2.5 Applying the Path Lasso Penalty

Since training a neural network is a non-convex optimization problem, with multiple local optima, how and when the regularization is added might affect which optimum is found. If a too high penalty is added too early during training, the risk of getting stuck in a bad local optimum is larger than if the regularization is added later. To mitigate this, training was split into three stages:

- Following the adaptive lasso approach of Allerbo and Jörnsten (2020), we first trained the network without path regularization to obtain individual penalties for each connection during the path lasso stage.

Algorithm 1 Proximal Path Lasso Optimization Step

Input: Parameters at time t, $\{(\boldsymbol{W_l})^t, (\boldsymbol{b_l})^t\}_{l=l}^L$; data, $\boldsymbol{x}$; learning rate, α; regularization strength, λ.
Output: Parameters at time $t+1$: $\{(\boldsymbol{W_l})^{t+1}, (\boldsymbol{b_l})^{t+1}\}_{l=l}^L$.

1: Update all weights and biases using one step of standard (stochastic) gradient descent:

$$\{(\boldsymbol{W_l})^{t+\frac{1}{2}}, (\boldsymbol{b_l})^{t+\frac{1}{2}}\}_{l=1}^L \leftarrow \{(\boldsymbol{W_l})^t, (\boldsymbol{b_l})^t\}_{l=1}^L - \alpha \cdot \nabla \left(\mathrm{NN}(\{(\boldsymbol{W_l})^t, (\boldsymbol{b_l})^t\}_{l=1}^L; \boldsymbol{x})\right),$$

where NN denotes the neural network.

2: Penalize paths, by applying the group lasso proximal operator, and update the path values accordingly:
a: Construct the path lasso penalty matrix for the weight outputs from step 1, according to Equation (6):

$$(\boldsymbol{W}_{\mathrm{PL}})^{t+\frac{1}{2}} \leftarrow \sqrt{\prod_{l=L,\dots,1} \left((\boldsymbol{W_l})^{t+\frac{1}{2}}\right)^2}.$$

b: Penalize paths according to Equation (7):

$$\boldsymbol{P}^{t+1} \leftarrow \prod_{l=L,\dots,1} |(\boldsymbol{W_l})^{t+\frac{1}{2}}| \odot \left(1 - \frac{\alpha\lambda}{(\boldsymbol{W}_{\mathrm{PL}})^{t+\frac{1}{2}}}\right)^+.$$

3: Translate penalized paths to penalized links:
a: Translate penalized paths to absolute valued penalized links, using modified non-negative matrix factorization:

$$\left\{(\tilde{\boldsymbol{W}}_{\boldsymbol{l}})^{t+1}\right\}_{l=1}^L \leftarrow \mathrm{NMF}\left(\boldsymbol{P}^{t+1}\right).$$

b: Restore signs:

$$(\boldsymbol{W_l})^{t+1} \leftarrow \mathrm{sign}((\boldsymbol{W_l})^t) \odot (\tilde{\boldsymbol{W}}_{\boldsymbol{l}})^{t+1}.$$

- In a second stage, we added path regularization with an individual penalty to each connection, depending on the magnitude of the connection after the first stage:

$$\lambda_{i_L,i_0} := \frac{\lambda}{((\hat{\boldsymbol{W}}_{\mathrm{PL}})_{i_L i_0})^\gamma},$$

where $\hat{\boldsymbol{W}}_{\mathrm{PL}}$ is the value of $\boldsymbol{W}_{\mathrm{PL}}$ after the first optimization stage and $\gamma > 0$. Throughout this paper, $\gamma = 2$ was used.

- To reduce bias and thus improve performance, we finally added a stage of unregularized training after the path lasso stage, with the links set to zero in the previous stage kept to zero.

All stages were trained until convergence and stages two and three were warm started with the solutions from the previous stage.

2.6 Path Lasso for Dimensionality Reduction

Path lasso as described so far is applicable to any two non-adjacent layers in any feedforward neural network. To use it for sparse non-linear dimensionality reduction, it was applied to an autoencoder, with the following adaptations:

- Path penalties were applied between the input ($\boldsymbol{x} \in \mathbb{R}^{d_x}$) and latent ($\boldsymbol{z} \in \mathbb{R}^{d_z}$) variables, and between the latent and output ($\hat{\boldsymbol{x}} \in \mathbb{R}^{d_x}$) variables.
- To enforce the encoder and the decoder to be symmetric, the group lasso groups were defined as all paths connecting x_i and z_j, together with all paths connecting z_j and $\hat{x}_i$, i.e.

$$\boldsymbol{W}_{\text{PL}} := \sqrt{\prod_{l=L,\dots,1} (\boldsymbol{W}_{\boldsymbol{l}}^{\boldsymbol{E}})^2 + \left(\prod_{l=L,\dots,1} (\boldsymbol{W}_{\boldsymbol{l}}^{\boldsymbol{D}})^2 \right)^\top},$$

  where $\{\boldsymbol{W}_{\boldsymbol{l}}^{\boldsymbol{E}}\}_{l=1}^{L}$ and $\{\boldsymbol{W}_{\boldsymbol{l}}^{\boldsymbol{D}}\}_{l=1}^{L}$ are the weight matrices of the encoder and the decoder, respectively. This means that if an input is disconnected to a latent variable, so is the corresponding output, by construction.
- To encourage the algorithm to make equal use of the latent dimensions, we added an exclusive lasso penalty to the elements in $\boldsymbol{W}_{\text{PL}}$, with as many groups as there are latent dimensions, each group being defined as the connections to a given latent dimension.

3. Experiments

In order to evaluate path lasso for dimensionality reduction we applied it to three different data sets, one with synthetic data, consisting of Gaussian clusters on a hypercube, one with text documents from newsgroup posts, and one with images of faces. In each experiment, 20 % of the data was set aside for testing and the remaining 80 % was split 90-10 into training and validation data; all visualizations were made using the testing data. All autoencoders used one hidden layer with tanh activations in the encoder and decoder respectively, and were trained with l_2-loss. For optimization stages not using proximal gradient descent, the Adam optimizer (Kingma and Ba, 2014) was used.

In addition to path lasso, we used a standard autoencoder, a sparse autoencoder, an autoencoder with parameter-wise l_1-regularization and thresholding (hereafter referred to as standard lasso), PCA and sparse PCA. The reason for adding thresholding in standard lasso is because unless proximal methods are used, no parameters are set exactly to zero, but instead to very small values, as discussed in Section 2.2. It should also be noted that the sparse autoencoder is sparse in terms of observations to latent dimensions, and not in terms of original to latent dimensions, as we are interested in; see the text associated to Equation (1) for details. The standard autoencoder and PCA are of course not sparse in any sense. Since "observation sparse" algorithms are not as relevant to us as the "truly sparse" algorithms, we omit them in some of the comparisons.

The following measures were calculated on the testing data:

- Reconstruction error as explained variance (R^2).

- Fraction of correctly identified reconstructions. A reconstructed observation is considered correctly identified if it is closer to its own original observation than any of the other ones, measured in l_2-distance, i.e. $\|\hat{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{i}}\|_2 < \|\hat{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{j}}\|_2$, $i \neq j$. This is hereafter referred to as observation reconstruction match.

- Fraction of correctly reconstructed labels. The reconstructed label is defined as the label of the original observation that is closest to the reconstructed observation, where distance is measured as above, i.e. the label of $\boldsymbol{x}_{\boldsymbol{j}}$, where $\|\hat{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{j}}\|_2 < \|\hat{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{k}}\|_2$, $j \neq k$. This is hereafter referred to as label reconstruction match.

3.1 Synthetic Data Set

Sixteen clusters were generated in $\mathbb{R}^4$, centered at each of the sixteen vertices in the hypercube $\{0,1\}^4$. For cluster i, 100 data points were sampled according to $\boldsymbol{x}_{\boldsymbol{i}} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{i}}, 0.01 \cdot \boldsymbol{I}_{\boldsymbol{4}})$, where $\boldsymbol{I}_{\boldsymbol{4}}$ is the identity matrix and $\boldsymbol{\mu}_{\boldsymbol{i}}$ is one of the sixteen vertices in $\{0,1\}^4$. The four dimensional data set was reduced down to two dimensions using the six different algorithms. For the four autoencoder based algorithms, the number of nodes in the five layers of the autoencoder were 4, 50, 2, 50 and 4, respectively. The three sparse algorithms (in the sense original to latent dimensions) were penalized so that four of the original eight connections remained. The experiment was performed twice, with and without added noise, distributed according to $\mathcal{N}(0, 0.3^2)$.

Ten different splits of the data into training and validation sets were done. For each split, three different optimization seeds were used and the seed resulting in the best R^2 value on the validation data was chosen. The resulting mean and standard deviations are presented in Table 1. The p-values come from the one-sided paired rank test, testing whether path lasso performs better than the competing algorithm. P-values smaller than 1 % are marked in bold. With noise added, path lasso performs significantly better than the other algorithms both in terms of R^2 and reconstruction match. Without noise path lasso still performs better than the dense algorithms in terms of observation reconstruction match, while all the four non-linear methods perform very well in terms of R^2 and label reconstruction.

The results with the best R^2 values are plotted in Figure 2, where clusters that are diagonal to each other in $\mathbb{R}^4$ (e.g. $(0,1,0,0)$ and $(1,0,1,1)$) are plotted using the same color, but with different markers - circles or crosses. For the three sparse algorithms, path lasso, standard lasso and sparse PCA, each of the two latent dimensions becomes a combination of two of the original four dimensions, which can be seen in the axis aligned data in the plots. Even with no added noise the linear algorithms, PCA and sparse PCA, are not able to fully separate the sixteen clusters, while all four non-linear algorithms, based an autoencoders, are.

3.2 Text - 20 Newsgroup Data Set

To test the algorithm on text data, the 20 newsgroups data set[1] was used. Out of the original 20 categories, the following 4 were selected: `soc.religion.christian`, `sci.space`, `comp.windows.x` and `rec.sport.hockey`, which resulted in 31225 documents. Then, for

1. Available at `http://qwone.com/~jason/20Newsgroups/`.

Algorithm	Noise	Connections	R^2		Reconstruction Match			
					Observation		Label	
			Mean (std)	p-value	Mean (std)	p-value	Mean (std)	p-value
Path Lasso	No	4	0.98 (0.0013)	-	0.37 (0.015)	-	1.0 (0.0014)	-
Standard Lasso	No	4	0.98 (0.0020)	0.28	0.36 (0.022)	0.070	1.0 (0.0014)	0.68
Sparse PCA	No	4	0.48 (0.0018)	**0.00098**	0.067 (0.019)	**0.00098**	0.30 (0.014)	**0.0029**
Autoencoder	No	8	0.98 (0.0026)	0.019	0.35 (0.013)	**0.0046**	1.0 (0.0021)	0.16
Sparse AE	No	8	0.98 (0.00027)	0.50	0.34 (0.0076)	**0.0039**	1.0 (0.0013)	0.018
PCA	No	8	0.48 (0.0014)	**0.00098**	0.067 (0.015)	**0.00098**	0.43 (0.044)	**0.0029**
Path Lasso	Yes	4	0.89 (0.017)	-	0.38 (0.038)	-	0.88 (0.019)	-
Standard Lasso	Yes	4	0.58 (0.097)	**0.00098**	0.11 (0.030)	**0.00098**	0.39 (0.11)	**0.0029**
Sparse PCA	Yes	4	0.50 (0.0016)	**0.00098**	0.11 (0.0061)	**0.0029**	0.37 (0.010)	**0.0029**
Autoencoder	Yes	8	0.86 (0.011)	**0.0029**	0.33 (0.014)	**0.0095**	0.86 (0.016)	**0.0088**
Sparse AE	Yes	8	0.85 (0.0038)	**0.0020**	0.30 (0.015)	**0.00098**	0.85 (0.0035)	**0.0020**
PCA	Yes	8	0.50 (0.0043)	**0.00098**	0.12 (0.0088)	**0.00098**	0.40 (0.030)	**0.00098**

Table 1: Number of remaining connections, explained variance, reconstruction match and p-value for six the algorithms when reducing 16 clusters from four to two dimensions with and without added noise. P-values are for the one-sided paired rank test, that tests whether path lasso performs better than the competing algorithm, with p-values smaller than 1 % marked in bold. For the three sparse algorithms, the number of connections is always four, by construction; for three the dense algorithms it is always eight. Especially for noisy data, path lasso outperforms the competing algorithms.

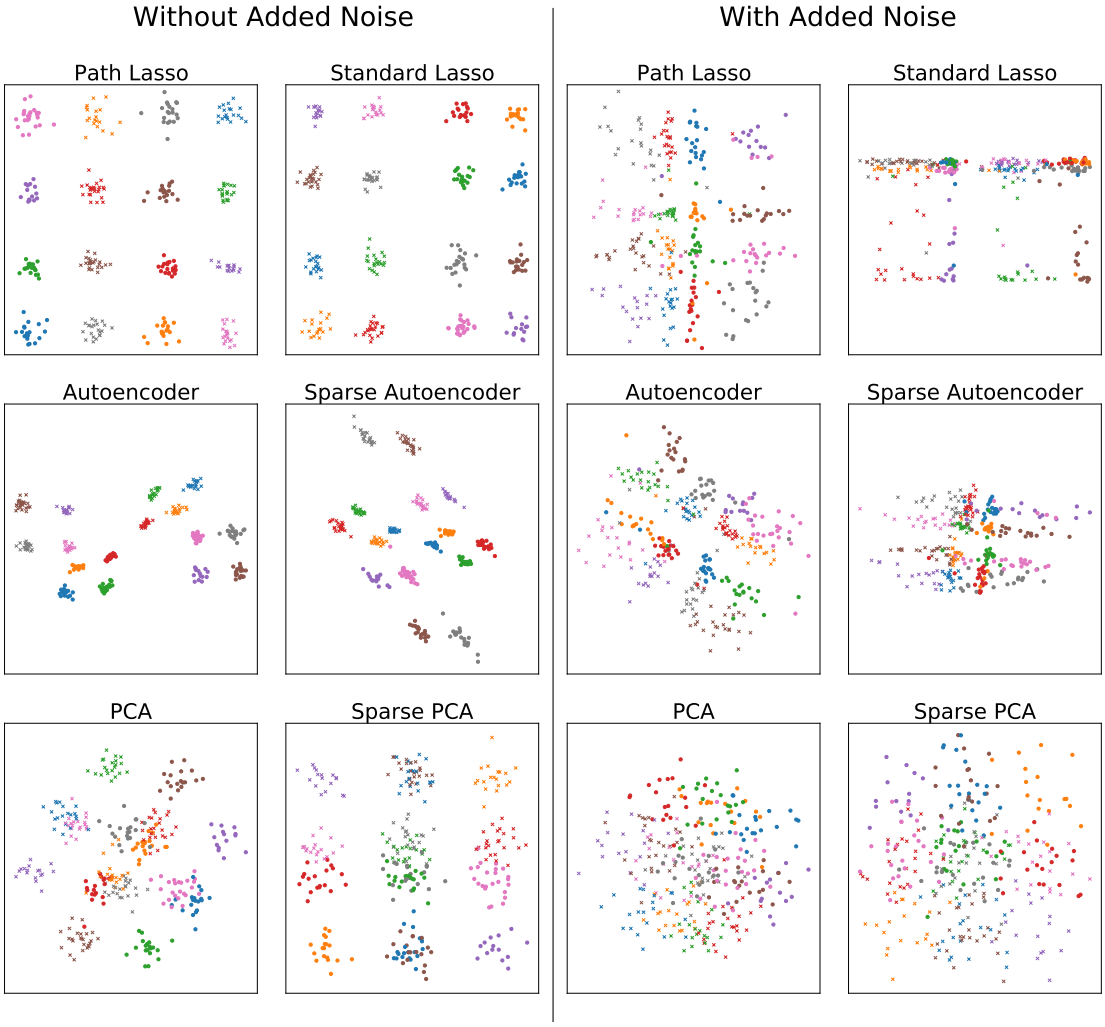


Figure 2: Reduction of 16 clusters from four to two dimension with and without added noise, using six different algorithms. Clusters that are diagonal to each other in $\mathbb{R}^4$ have the same color, but different markers.

each word the tf-idf score was calculated, and the 100 words with the highest score were kept, resulting in a 31225×100 data matrix. Using a standard autoencoder, path lasso, standard lasso and sparse PCA, the 100 dimensional data set was mapped down to two, four and 25 dimensions, where the 25D case was added for comparing the test measures at moderate dimensionality reductions. For the autoencoder based algorithms, the layer widths were 100, 50, 2 (4, 25), 50 and 100 nodes.

Two different sparsity levels were used; One sparse, to be able to compare interpretability, and one almost dense, to be able to compare to a standard autoencoder. In each case the penalties were set to obtain (approximately) the same sparsity, measured as number of connections, for all algorithms.

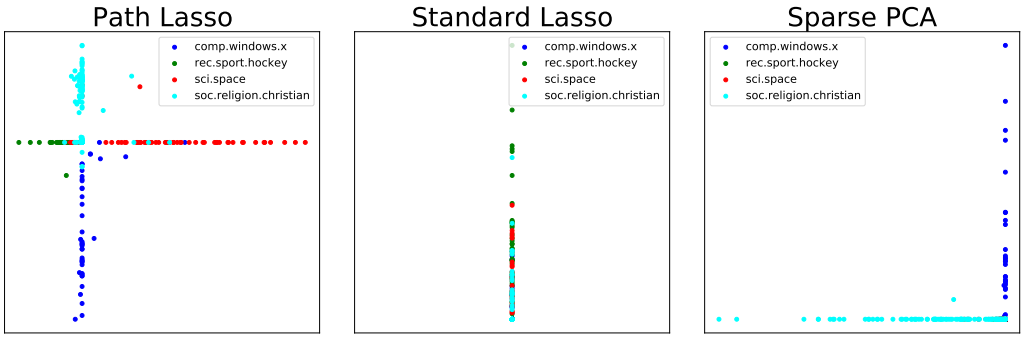
The latent spaces in the sparse case are shown in Figure 3. In the 4D case, standard lasso only uses one latent dimension and is incapable of distinguishing between the categories, sparse PCA maps one category to each latent dimension, while path lasso has a tendency to map two categories to each latent dimension, one to the positive and one to the negative axis. This is further accentuated when compressing to only two dimensions. While path lasso is able to identify all four categories, sparse PCA only identifies two of them. Standard lasso still only uses one dimension and is not able to identify any categories at all. The use of only one latent dimension by standard lasso is likely attributed to the flexibility of the autoencoder, and without the structure in the penalty imposed by the path lasso algorithm, there is less incentive to use all parts of the latent space. The same tendency is visible in Figure 4c.

To see which original dimensions (words) contribute to which latent dimensions, the non-zero elements of $\boldsymbol{W}_{\text{PL}}$ can be used, but since all elements in this matrix are non-negative, it gives no information about the sign. Instead the corresponding signed matrix was created according to $\boldsymbol{W_2} \cdot \boldsymbol{W_1} + (\boldsymbol{W_4} \cdot \boldsymbol{W_3})^\top$, where $(\boldsymbol{W_1}, \boldsymbol{W_2})$ and $(\boldsymbol{W_3}, \boldsymbol{W_4})$ are the weight matrices of the encoder and the decoder, respectively. The signed words in the latent dimensions are presented in Tables 2 and 3. The assignment of words to the 4 (8) latent half-axes done by path lasso and sparse PCA is consistent with the results in Figure 3. Path lasso also seems to identify a subcategory of `comp.windows.x` related to e-mails.

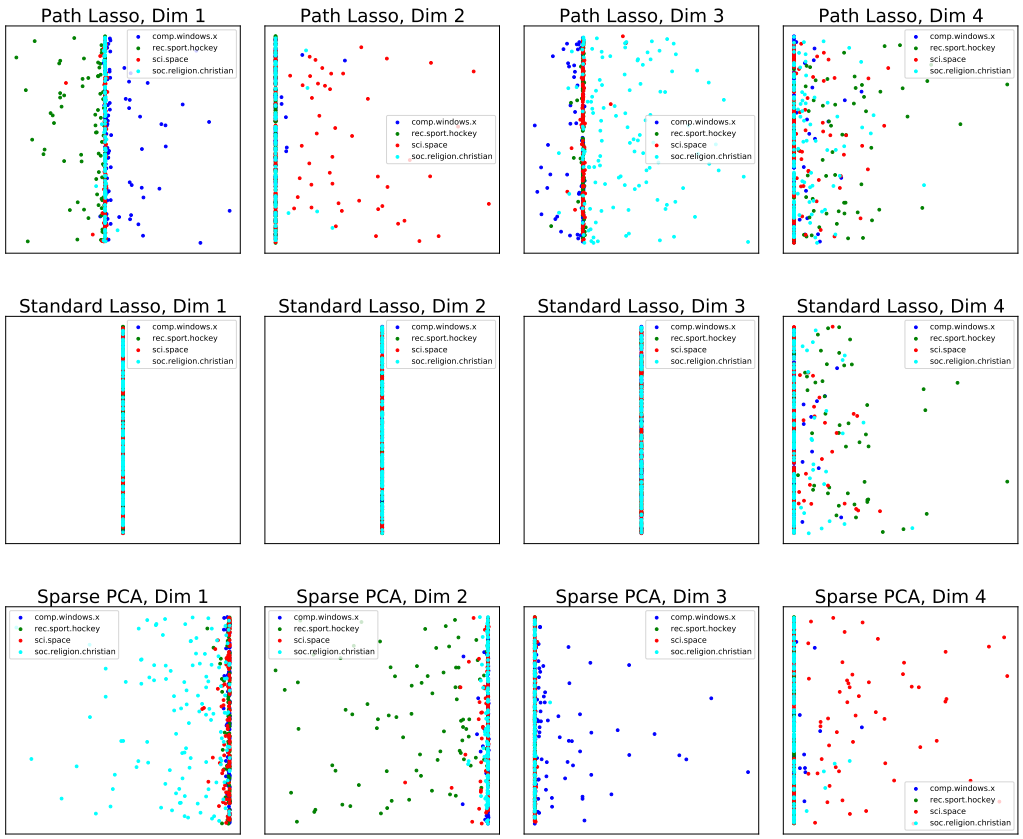
In Tables 4 and 5, remaining connections, explained variance and reconstruction match are presented for both sparsity levels. We conclude that path lasso performs best among the sparse algorithms both in terms of explained variance and reconstruction match. Compared to the standard autoencoder, path lasso performs slightly better in terms of reconstruction error and observation reconstruction match, which is in line with the results in Section 3.1. It is also noticeable that with increasing latent dimensionality the advantage of non-linear over linear models decreases.

3.3 Images - AT&T Face Database

We also tested the algorithm on the AT&T face database (Samaria and Harter, 1994), which contains 400 grayscale images of faces. The images were compressed to a size of 60×50 pixels, after which the 3000 dimensional images where reduced to 5 dimensions. Both the encoder and the decoder had 1000 units wide hidden layers, and to assure that a pixel that was disconnected from all the latent dimensions got a value of zero, no bias parameters were used.



(a) Latent 2D space for the newsgroup data



(b) Latent 4D space for the newsgroup data

Figure 3: Latent spaces of the news data, when compressed down to two and to four latent dimensions. In 3b the y-axis value is just a random number, added to increase readability. Path lasso uses the latent space more efficiently than the other algorithms, being capable of mapping one category to each half-axis.

Algorithm	Dimension	Axis	Words
Path Lasso	1 of 2	Negative	team, hockey, better, probably
		Positive	space
	2 of 2	Negative	window, hi
		Positive	god, church
Standard Lasso	1 of 2	Negative	-
		Positive	-
	2 of 2	Negative	-
		Positive	people, good, team, new, did, hockey, make, going, better, probably, lot
Sparse PCA	1 of 2	Negative	god, people, jesus, believe, bible, christ, faith, life
		Positive	-
	2 of 2	Negative	-
		Positive	window, application

Table 2: Positive and negative words in the two latent dimension, with sign calculated as $\text{sign}(W_2 \cdot W_1 + (W_4 \cdot W_3)^\top)$.

Again, each algorithm was penalized to obtain (approximately) the same number of connections, at two different sparsity levels. One low, with almost all of the $5 \cdot 3000 = 15000$ connections kept to be able to compare to a dense, l_2-regularized autoencoder; and one high, with only a fourth of the connections kept, to compare the algorithms at more extreme sparsity levels.

The results are summarized in Table 6. Since the images are unlabeled the label reconstruction match was replaced by a 10 nearest neighbor reconstruction match, where an observation is considered correctly reconstructed if its own original observation is among the ten closest original observations.

Path lasso and standard lasso do much better than sparse PCA in terms of reconstruction, and at low sparsity they are on par with the dense autoencoder. For observation reconstruction match, path lasso outperforms the other two algorithms, and at low sparsity also the standard autoencoder. For the 10 nearest neighbors reconstruction match, all non-linear algorithms do very well at low sparsity, while at high sparsity path lasso does better than the other algorithms.

The five eigenfaces, calculated as $\text{Dec}(e_i)$ where Dec is the decoder and e_i is the i-th standard basis vector in $\mathbb{R}^5$, are shown in the left column of Figure 4. For PCA, the decoder function corresponds to multiplication with the loadings matrix, whose i-th column is the i-th eigenface. A pixel with value zero is colored red. Sparse PCA distributes its non-zero pixels more evenly among the latent dimensions, while again standard lasso does not use all 5 latent dimensions. The right column shows the reconstruction of some of the images in the test set, using the four different algorithms. The reconstructed images using path lasso look more diversified than for the other sparse algorithms, something that is in line with the superior reconstruction match of path lasso.

Algorithm	Dimension	Axis	Words
Path Lasso	1 of 4	Negative	game, year, hockey, play, games, players, season, nhl
		Positive	window, use, need, server, program, motif, using, windows, application, widget
	2 of 4	Negative	-
		Positive	space
	3 of 4	Negative	know, thanks, edu, mail, hi, list
		Positive	god, don, think, people, jesus, say, believe, church, christians, christian, bible, christ, faith, life
	4 of 4	Negative	-
		Positive	team, new, hockey, going, better, probably, lot
Standard Lasso	1 of 4	Negative	-
		Positive	-
	2 of 4	Negative	-
		Positive	-
	3 of 4	Negative	-
		Positive	-
	4 of 4	Negative	god, like, know, don, think, space, does, christ, time, window, thanks, use, jesus, way, say, sun, believe, need, want, problem, edu, server, hi, right, church, program, using, work, nasa, life, christians, true, bible, help, mail, used, actually
		Positive	just, people, good, team, new, did, hockey, make, going, better, probably, lot
Sparse PCA	1 of 4	Negative	god, don, think, people, does, jesus, say, believe, church, things, question, said, christians, true, christian, bible, come, world, point, christ, faith, life
		Positive	-
	2 of 4	Negative	game, good, team, year, hockey, play, games, players, season, better, best, nhl
		Positive	use
	3 of 4	Negative	-
		Positive	window, server, program, motif, using, windows, application, widget
	4 of 4	Negative	-
		Positive	space, nasa, earth

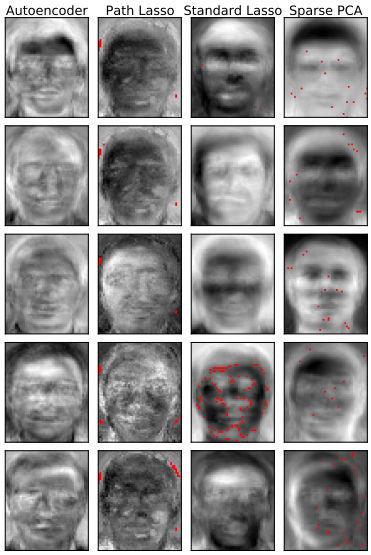
Table 3: Positive and negative words in the four latent dimension, with sign calculated as $\text{sign}(W_2 \cdot W_1 + (W_4 \cdot W_3)^\top)$.

Dimensions	Algorithm	Connections	R^2	Reconstruction Match	
				Observation	Label
2	Path lasso	9	0.13	0.021	0.47
2	Standard Lasso	11	0.015	0.0021	0.25
2	Sparse PCA	10	0.028	0.0084	0.29
4	Path lasso	46	0.20	0.027	0.55
4	Standard Lasso	47	0.015	0.0021	0.25
4	Sparse PCA	46	0.11	0.017	0.43
25	Path Lasso	102	0.55	0.34	0.66
25	Standard Lasso	104	0.069	0.0063	0.33
25	Sparse PCA	104	0.40	0.14	0.56

Table 4: Number of remaining connections, explained variance and reconstruction match for the three algorithms on the newsgroup data at high sparsity. Path lasso performs best both in terms of explained variance and reconstruction match.

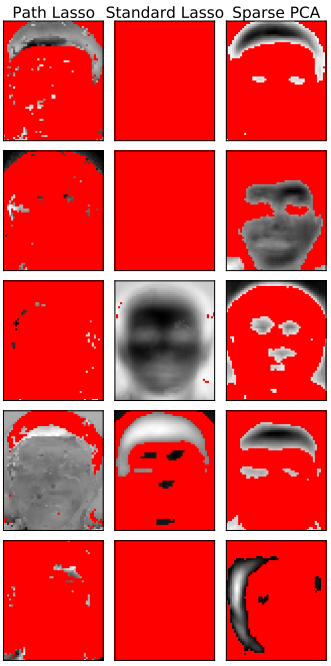
Dimensions	Algorithm	Connections	R^2	Reconstruction Match	
				Observation	Label
2	Autoencoder	200	0.24	0.048	0.56
2	Path Lasso	194	0.27	0.055	0.54
2	Standard Lasso	196	0.073	0.0042	0.35
2	Sparse PCA	195	0.051	0.0063	0.32
4	Autoencoder	400	0.33	0.080	0.58
4	Path Lasso	396	0.34	0.10	0.56
4	Standard Lasso	398	0.19	0.019	0.45
4	Sparse PCA	397	0.14	0.015	0.43
25	Autoencoder	2500	0.60	0.47	0.71
25	Path Lasso	2475	0.61	0.49	0.72
25	Standard Lasso	2476	0.44	0.16	0.61
25	Sparse PCA	2486	0.44	0.15	0.57

Table 5: Number of remaining connections, explained variance and reconstruction match for the four algorithms on the newsgroup data at low sparsity. Path lasso performs better than standard lasso and sparse PCA both in terms of explained variance and reconstruction match. Compared to the standard autoencoder path lasso performs slightly better in terms of explained variance and observation reconstruction match.

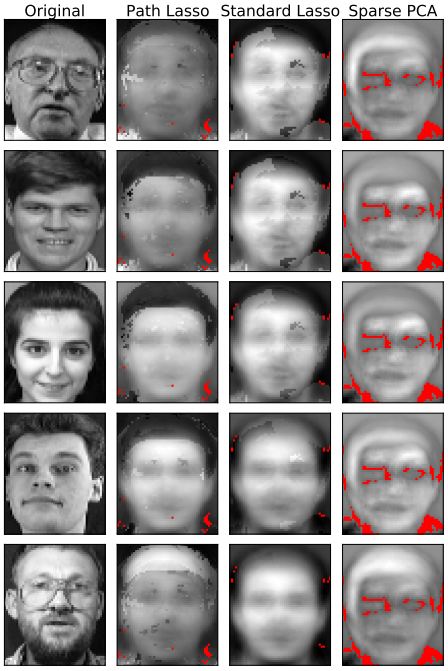


(a) Eigenfaces at low sparsity

(b) Reconstructions at low sparsity



(c) Eigenfaces at high sparsity



(d) Reconstructions at high sparsity

Figure 4: Eigenfaces and examples of reconstructions of test faces at small and large penalization. Zeros are indicated with red. The results of the dense autoencoder are presented together with the low sparsity results. The reconstructed images using path lasso look more diverse than those of the other sparse algorithms. At high sparsity standard lasso does not use all 5 latent dimensions.

Algorithm	Connections	R^2	Reconstruction Match	
			Observation	10 Nearest Neighbors
Autoencoder	15000	0.73	0.66	0.97
Path Lasso	14923	0.72	0.70	0.96
Standard Lasso	14936	0.70	0.55	0.96
Sparse PCA	14923	-0.24	0.16	0.60
Path Lasso	3770	0.57	0.21	0.75
Standard Lasso	3771	0.44	0.025	0.26
Sparse PCA	3770	-1.2	0.10	0.46

Table 6: Number of remaining connections, explained variance and reconstruction match for the four algorithms on the image data. Path lasso outperforms the competing algorithms in terms of observation reconstruction match, and at low sparsity even beats the fully connected autoencoder.

4. Conclusions

We proposed path lasso, a penalty that creates structured sparsity in feedforward neural networks, by using a group lasso penalty to remove all connections between two nodes in two non-adjacent layers. We applied path lasso to an autoencoder to obtain sparse, non-linear dimensionality reduction, and showed that this non-linearity makes path lasso much more flexible than sparse PCA, leading to a lower reconstruction error and a higher reconstruction match. Thanks to its higher flexibility path lasso is also able to use the latent space more efficiently, something that proved essential when the latent dimensions were few. Compared to an autoencoder with individually lasso penalized links, path lasso performed better in terms of reconstruction, reconstruction match and interpretation of the latent space. In addition, at low sparsity levels path lasso resulted in a better reconstruction match than the standard autoencoder.

Using path lasso for non-linear, sparse dimensionality reduction, flexibility and interpretability can be combined in a new way, enabling compression to fewer dimensions, with preserved interpretability. In this paper, we used the path lasso penalty in an autoencoder. However, path lasso can be used in many other types of feedforward neural networks with applications including non-linear, sparse multivariate regression, and non-linear, sparse network models. Such investigations are left for future work.

Code is available at `https://github.com/allerbo/path_lasso`.

Acknowledgments

We would like to thank the anonymous reviewers, whose suggestions helped improve and clarify the manuscript.

This research was supported by funding from the Swedish Research Council (VR), the Swedish Foundation for Strategic Research, the Wallenberg AI, Autonomous Systems and Software Program (WASP), and the Chalmers AI Research Center (CHAIR).

Appendix A. Accelerating the Non-Negative Matrix Factorization

To increase the speed of the matrix factorization step in Equation (7) the following three approaches were used: Substitution, parallelization and Boolean matrix factorization.

A.1 Substitution

By first applying an off-the-shelf optimization method to $f(\boldsymbol{\theta}) = g(\boldsymbol{\theta}) + \lambda \boldsymbol{W}_{\text{PL}}$, where $g(\boldsymbol{\theta})$ is the reconstruction error, we obtain a solution where some of the paths have very small values, although non-zero. We then, as in adaptive lasso, use this solution to initialize a second optimization stage, using proximal gradient descent, with individual penalties for each connection, where the penalties depend on the magnitude of the connection after the first stage:

$$\lambda_{i_L, i_0} := \frac{\lambda}{((\hat{\boldsymbol{W}}_{\text{PL}})_{i_L i_0})^\gamma}$$

where $\hat{\boldsymbol{W}}_{\text{PL}}$ is the value of $\boldsymbol{W}_{\text{PL}}$ after the optimization with the off-the-shelf optimizer and $\gamma > 0$. Just as in Section 2.5, $\gamma = 2$ was used. Choosing a relatively small value of λ, connections that are not close to zero after the first stage will be left more or less unpenalized, while connections close to zero will be penalized hard. Another advantage of this two stage procedure is that we are able to further increase the speed in the first stage by leveraging on momentum based optimization methods, which are not available for proximal gradient descent.

A.2 Parallelization

In general, the larger the weight matrices, the more time consuming is the factorization of the penalized path matrix into penalized weight matrices. By splitting the weight matrices into blocks, the factorization can instead be done in parallel for smaller sub-matrices. In the simplest example, three layers are split into two parts each. This corresponds to two weight matrices, whose rows and columns are both split into two blocks each, and is shown in Equation (8) below. Let $\boldsymbol{A} \in (\mathbb{R}_{\geq 0})^{d_3 \times d_2}$ and $\boldsymbol{B} \in (\mathbb{R}_{\geq 0})^{d_2 \times d_1}$ denote the absolute valued weight matrices and let $\boldsymbol{C} + \boldsymbol{D} \in (\mathbb{R}_{\geq 0})^{d_3 \times d_1}$ denote the penalized path matrix, where the sum of the paths in $(\boldsymbol{C} + \boldsymbol{D})_{ij}$ is split into $\boldsymbol{C}_{ij}$, containing the sum of some of the paths, and $\boldsymbol{D}_{ij}$, containing the sum of the remaining paths. Which paths belong to $\boldsymbol{C}$ and which belong to $\boldsymbol{D}$ depends on how $\boldsymbol{A}$ and $\boldsymbol{B}$ are split. Then, the factorization of $\boldsymbol{C} + \boldsymbol{D}$ into $\boldsymbol{A} \cdot \boldsymbol{B}$ can be divided into $2^3 = 8$ smaller factorizations which can be solved in parallel, i.e.

$$\left[\begin{array}{c|c} \boldsymbol{C_{11}} + \boldsymbol{D_{11}} & \boldsymbol{C_{12}} + \boldsymbol{D_{12}} \\ \hline \boldsymbol{C_{21}} + \boldsymbol{D_{21}} & \boldsymbol{C_{22}} + \boldsymbol{D_{22}} \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{A_{11}} & \boldsymbol{A_{12}} \\ \hline \boldsymbol{A_{21}} & \boldsymbol{A_{22}} \end{array}\right] \cdot \left[\begin{array}{c|c} \boldsymbol{B_{11}} & \boldsymbol{B_{12}} \\ \hline \boldsymbol{B_{21}} & \boldsymbol{B_{22}} \end{array}\right] \tag{8}$$

can be split into

$$\left[\begin{array}{c|c} \boldsymbol{C_{11}} & \boldsymbol{C_{12}} \\ \hline \boldsymbol{C_{21}} & \boldsymbol{C_{22}} \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{A_{11}} & \boldsymbol{0} \\ \hline \boldsymbol{A_{21}} & \boldsymbol{0} \end{array}\right] \cdot \left[\begin{array}{c|c} \boldsymbol{B_{11}} & \boldsymbol{B_{12}} \\ \hline \boldsymbol{0} & \boldsymbol{0} \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{A_{11}} \cdot \boldsymbol{B_{11}} & \boldsymbol{A_{11}} \cdot \boldsymbol{B_{12}} \\ \hline \boldsymbol{A_{21}} \cdot \boldsymbol{B_{11}} & \boldsymbol{A_{21}} \cdot \boldsymbol{B_{12}} \end{array}\right]$$

$$\left[\begin{array}{c|c} \boldsymbol{D_{11}} & \boldsymbol{D_{12}} \\ \hline \boldsymbol{D_{21}} & \boldsymbol{D_{22}} \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{0} & \boldsymbol{A}_{12} \\ \hline \boldsymbol{0} & \boldsymbol{A}_{22} \end{array}\right] \cdot \left[\begin{array}{c|c} \boldsymbol{0} & \boldsymbol{0} \\ \hline \boldsymbol{B_{21}} & \boldsymbol{B_{22}} \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{A_{12}} \cdot \boldsymbol{B_{21}} & \boldsymbol{A_{12}} \cdot \boldsymbol{B_{22}} \\ \hline \boldsymbol{A_{22}} \cdot \boldsymbol{B_{21}} & \boldsymbol{A_{22}} \cdot \boldsymbol{B_{22}} \end{array}\right]$$

or equivalently

$$\boldsymbol{C_{11}} = \boldsymbol{A_{11}} \cdot \boldsymbol{B_{11}},\ \boldsymbol{C_{12}} = \boldsymbol{A_{11}} \cdot \boldsymbol{B_{12}},\ \boldsymbol{C_{21}} = \boldsymbol{A_{21}} \cdot \boldsymbol{B_{11}},\ \boldsymbol{C_{22}} = \boldsymbol{A_{21}} \cdot \boldsymbol{B_{12}}$$
$$\boldsymbol{D_{11}} = \boldsymbol{A_{12}} \cdot \boldsymbol{B_{21}},\ \boldsymbol{D_{12}} = \boldsymbol{A_{12}} \cdot \boldsymbol{B_{22}},\ \boldsymbol{D_{21}} = \boldsymbol{A_{22}} \cdot \boldsymbol{B_{21}},\ \boldsymbol{D_{22}} = \boldsymbol{A_{22}} \cdot \boldsymbol{B_{22}}.$$

Here, $\boldsymbol{A_{ij}} \in (\mathbb{R}_{\geq 0})^{d_{3i} \times d_{2j}}$, $\boldsymbol{B_{ij}} \in (\mathbb{R}_{\geq 0})^{d_{2i} \times d_{1j}}$ and $\boldsymbol{C_{ij}}, \boldsymbol{D_{ij}} \in (\mathbb{R}_{\geq 0})^{d_{3i} \times d_{1j}}$, where $i, j \in \{1, 2\}$ and $d_{ki} + d_{kj} = d_k$ for $k = 1, 2, 3$, are all block matrices. This results in two, potentially different, solutions for each $\boldsymbol{A_{ij}}$ and $\boldsymbol{B_{ij}}$, that need to be aggregated. To be conservative, and set a link to zero only when it is zero in both solutions, we use the maximum value element-wise for aggregation:

$$(\boldsymbol{A_{ij}})_{kl} = \max_{n \in \{1,2\}} (\boldsymbol{A_{ij}^n})_{kl}$$

where $\boldsymbol{A_{ij}^n}$ is the n-th solution of $\boldsymbol{A_{ij}}$. Remember that all elements in all matrices are non-negative.

This generalizes trivially to more layers and splits and the number of equations is always the product of the splits over all the layers.

A.3 Boolean Matrix Factorization

Another way to speed up the computations is early stopping of the matrix factorization. This may however lead to some of the elements on the right hand side in Equation (7) being very close, but not equal, to zero, introducing the need to threshold. We determine the optimal threshold value using Boolean matrix multiplication, which differs from ordinary matrix multiplication in the following way:

- All elements are in $\{0, 1\}$.
- Instead of ordinary sum, Boolean sum (or Boolean **or**), $\vee$, is used: $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$.

We define a Boolean matrix for each matrix in Equation (7) as:

$$(\boldsymbol{B})_{ij} := \mathbb{I}\left[\left(\prod_{l=L,\ldots,1} |\boldsymbol{W_l}| \odot \left(1 - \frac{\alpha\lambda}{\boldsymbol{W}_{\text{PL}}}\right)^+\right)_{ij} > 0\right]$$
$$(\boldsymbol{B_l})_{ij}(\tau) := \mathbb{I}\left[(\tilde{\boldsymbol{W}}_{\boldsymbol{l}})_{ij} > \tau\right],\ \tau \geq 0$$

where $\mathbb{I}[\cdot]$ is the (element-wise) indicator function which equals 1 when its argument is true and 0 otherwise. Thus, in the matrix on the left hand side in Equation (7), zero elements become zeros and all other elements become ones, while for the matrices on the right hand side, we use a threshold to decide which elements should be set to zero. In the first case, an element of one means that there is an non-zero connection, while in the second case it means that there is an non-zero link. If $\boldsymbol{B} = \bigvee_{l=L,\ldots,1} \boldsymbol{B_l}$, where $\bigvee$ denotes Boolean matrix multiplication, then the matrix factorization was successful in the sense that all connections in $\boldsymbol{W}_{\text{PL}}$ are represented by appropriate links in the weight matrices. Each differing element means that either two nodes are disconnected in $\boldsymbol{W}_{\text{PL}}$ but not in $\{\tilde{\boldsymbol{W}}_{\boldsymbol{l}}\}_{l=1}^{L}$ or vice versa. To

minimize this quantity, we minimize $\sum |\boldsymbol{B} - \bigvee_{l=L,\dots,1} \boldsymbol{B_l}(\tau)|$, with the absolute value taken element-wise, with respect to τ, and then threshold $\{\tilde{\boldsymbol{W}}_{\boldsymbol{l}}\}_{l=1}^{L}$ using the resulting τ:

$$\tau = \underset{\tau \geq 0}{\operatorname{argmin}} \sum_{\text{all elements}} \left| \mathbb{I}\left[\prod_{l=L,\dots,1} |\boldsymbol{W_l}| \odot \left(1 - \frac{\alpha\lambda}{\boldsymbol{W}_{\text{PL}}}\right)^{+} > 0 \right] - \bigvee_{l=L,\dots,1} \mathbb{I}[\tilde{\boldsymbol{W}}_{\boldsymbol{l}} > \tau] \right|$$
$$\tilde{\boldsymbol{W}}_{\boldsymbol{l}} \leftarrow \tilde{\boldsymbol{W}}_{\boldsymbol{l}} \odot \mathbb{I}[\tilde{\boldsymbol{W}}_{\boldsymbol{l}} > \tau]$$

Since the objective function is piecewise constant and thus hard to optimize, we simply evaluate it for 20 logarithmically spaced values of $\tau \in [10^{-10}, 1]$ and pick the best one.

Appendix B. Proofs

Proof of Proposition 3 Using the definition of matrix multiplication, with k running over all paths connecting x_{i_0} and y_{i_L}, we obtain

$$\begin{aligned}
\sqrt{\sum_{k \in g_{i_L i_0}} (p^k_{i_L i_0})^2} &= \sqrt{\sum_{i_{L-1}=1}^{d_{L-1}} \sum_{i_{L-2}=1}^{d_{L-2}} \cdots \sum_{i_1=1}^{d_1} (|w^L_{i_L i_{L-1}}| \cdot |w^{L-1}_{i_{L-1} i_{L-2}}| \cdot \ldots |w^1_{i_1 i_0}|)^2} \\
&= \sqrt{\sum_{i_{L-1}=1}^{d_{L-1}} \sum_{i_{L-2}=1}^{d_{L-2}} \cdots \sum_{i_1=1}^{d_1} (w^L_{i_L i_{L-1}})^2 \cdot (w^{L-1}_{i_{L-1} i_{L-2}})^2 \cdot \ldots (w^1_{i_1 i_0})^2} \\
&= \left(\sqrt{(\boldsymbol{W_L})^2 \cdot (\boldsymbol{W_{L-1}})^2 \cdot \ldots (\boldsymbol{W_1})^2} \right)_{i_L i_0} = (\boldsymbol{W}_{\text{PL}})_{i_L i_0}.
\end{aligned}$$

■

To prove Proposition 4 we begin with the following lemma:

Lemma 6 *Let $\{\boldsymbol{A_k}\}_{k=1}^n$ be a set of arbitrary matrices and $\{\boldsymbol{D_k}\}_{k=1}^n$ a set of diagonal matrices, where all elements in $\boldsymbol{A_k}$ and $\boldsymbol{D_k}$ are bounded. Let $f \geq 0$ an element-wise function where $f(x) = 0$ if and only if $x = 0$. Let the dimensions of the matrices be such that the matrix multiplications below make sense. Then*

$$(f(\boldsymbol{A_1}) \cdot f(\boldsymbol{A_2}) \cdot \ldots f(\boldsymbol{A_n}))_{ij} = 0 \implies (\boldsymbol{D_1} \cdot \boldsymbol{A_1} \cdot \boldsymbol{D_2} \cdot \boldsymbol{A_2} \cdot \boldsymbol{D_2} \cdot \ldots \boldsymbol{D_n} \cdot \boldsymbol{A_n})_{ij}.$$

Proof Denote $a^k_{ij} := (A_k)_{ij}$ and $d^k_{ij} := (D_k)_{ij}$. Then for the left hand side

$$(f(\boldsymbol{A_1}) \cdot f(\boldsymbol{A_2}) \cdot \ldots f(\boldsymbol{A_n}))_{ij} = \sum_{k_1} \sum_{k_2} \cdots \sum_{k_{n-1}} f(a^1_{ik_1}) \cdot f(a^2_{k_1 k_2}) \cdot \ldots f(a^n_{k_{n-1} j})$$

and for the right hand side

$$(\boldsymbol{D_1} \cdot \boldsymbol{A_1} \cdot \boldsymbol{D_2} \cdot \boldsymbol{A_2} \cdot \ldots \boldsymbol{D_n} \cdot \boldsymbol{A_n})_{ij} = \sum_{k_1} \sum_{k_2} \cdots \sum_{k_{n-1}} a^1_{ik_1} \cdot a^2_{k_1 k_2} \cdot \ldots a^n_{k_{n-1} j} \cdot d^1_{ii} \cdot d^2_{k_1 k_1} \cdot \ldots d^n_{k_{n-1} k_{n-1}}.$$

By the definition of f, $f(a^1_{ik_1}) \cdot f(a^2_{k_1k_2}) \cdot \ldots f(a^n_{k_{n-1}j}) \geq 0$ with equality only if at least one of $a^1_{ik_1},\ a^2_{k_1k_2}, \ldots a^n_{k_{n-1}j}$ is zero, which implies that $a^1_{ik_1} \cdot a^2_{k_1k_2} \cdot \ldots a^n_{k_{n-1}j} = 0$.

The sum

$$\sum_{k_1} \sum_{k_2} \cdots \sum_{k_{n-1}} f(a^1_{ik_1}) \cdot f(a^2_{k_1k_2}) \cdot \ldots f(a^n_{k_{n-1}j})$$

is zero if and only if all its terms are zero, which means that

$$a^1_{ik_1} \cdot a^2_{k_1k_2} \cdot \ldots a^n_{k_{n-1}j} = 0$$

for all combinations of indices summed over. Multiplying with some bounded constant does not change that fact, so

$$f(a^1_{ik_1}) \cdot f(a^2_{k_1k_2}) \cdot \ldots f(a^n_{k_{n-1}j}) = 0 \implies a^1_{ik_1} \cdot a^2_{k_1k_2} \cdot \ldots a^n_{k_{n-1}j} \cdot d^1_{ii} \cdot d^2_{k_1k_1} \cdot \ldots d^n_{k_{n-1}k_{n-1}} = 0.$$

Finally summing only zeros we get

$$\begin{aligned} 0 &= \sum_{k_1} \sum_{k_2} \cdots \sum_{k_{n-1}} a^1_{ik_1} \cdot a^2_{k_1k_2} \cdot \ldots a^n_{k_{n-1}j} \cdot d^1_{ii} \cdot d^2_{k_1k_1} \cdot \ldots d^n_{k_{n-1}k_{n-1}} \\ &= (\boldsymbol{D_1} \cdot \boldsymbol{A_1} \cdot \boldsymbol{D_2} \cdot \boldsymbol{A_2} \cdot \ldots \boldsymbol{D_n} \cdot \boldsymbol{A_n})_{ij} \end{aligned}$$

■

Proof of Proposition 4 With $\{\boldsymbol{o_l}\}_{l=0}^{L}$ (where $\boldsymbol{o_0} = \boldsymbol{x}$ and $\boldsymbol{o_L} = \boldsymbol{y}$) denoting the outputs and $\{\boldsymbol{i_l}\}_{l=1}^{L}$ the inputs of the activation functions $\{\Phi_l\}_{l=1}^{L}$, we can express Equation (4) recursively for $1 \leq l \leq L$ as

$$\begin{aligned} \boldsymbol{i_l} &= \boldsymbol{W_l o_{l-1}} + \boldsymbol{b_l} \\ \boldsymbol{o_l} &= \Phi_k(\boldsymbol{i_l}) \end{aligned}$$

Applying the chain rule we obtain

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{o_L}}{\partial \boldsymbol{o_0}} = \frac{\partial \boldsymbol{o_L}}{\partial \boldsymbol{i_L}} \cdot \frac{\partial \boldsymbol{i_L}}{\partial \boldsymbol{o_{L-1}}} \cdot \frac{\partial \boldsymbol{o_{L-1}}}{\partial \boldsymbol{i_{L-1}}} \cdot \frac{\partial \boldsymbol{i_{L-1}}}{\partial \boldsymbol{o_{L-2}}} \cdot \ldots \frac{\partial \boldsymbol{o_1}}{\partial \boldsymbol{i_1}} \cdot \frac{\partial \boldsymbol{i_1}}{\partial \boldsymbol{o_0}}, \tag{9}$$

where $\frac{\partial \boldsymbol{o_l}}{\partial \boldsymbol{i_l}}$ is a diagonal matrix, since Φ_l is an element-wise operator, and $\frac{\partial \boldsymbol{i_l}}{\partial \boldsymbol{o_{l-1}}} = \boldsymbol{W_l}$. We can now apply Lemma 6 with $f(x) = x^2$ to Equation (9) to obtain

$$\begin{aligned} (\boldsymbol{W}_{\mathrm{PL}})_{i_L i_0} = 0 &\iff ((\boldsymbol{W_L})^2 \cdot (\boldsymbol{W_{L-1}})^2 \cdot \ldots (\boldsymbol{W_1})^2)_{i_L i_0} = 0 \\ \implies 0 = \left(\frac{\partial \boldsymbol{o_L}}{\partial \boldsymbol{i_L}} \cdot \boldsymbol{W_L} \cdot \frac{\partial \boldsymbol{o_{L-1}}}{\partial \boldsymbol{i_{L-1}}} \cdot \boldsymbol{W_{L-1}} \cdot \ldots \frac{\partial \boldsymbol{o_1}}{\partial \boldsymbol{i_1}} \cdot \boldsymbol{W_1} \right)_{i_L i_0} &= \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \right)_{i_L i_0} \end{aligned}$$

where the equivalence comes from the definition of $\boldsymbol{W}_{\mathrm{PL}}$ and the implication from Lemma 6. ■

Proof of Proposition 5 Assuming the penalized path can be written as a product of penalized links, we get the following equation for the k-th path between nodes i_0 and i_L, in total $\prod_{l=0}^{L} d_k$ equations:

$$\begin{aligned}
&p^k_{i_L i_0} \cdot \left(1 - \frac{\alpha\lambda}{(\boldsymbol{W}_{\text{PL}})_{i_L i_0}}\right)^+ = |w^L_{i_L i^k_{L-1}}| \cdot |w^{L-1}_{i^k_{L-1} i^k_{L-2}}| \cdot \ldots |w^1_{i^k_1 i_0}| \cdot \left(1 - \frac{\alpha\lambda}{(\boldsymbol{W}_{\text{PL}})_{i_L i_0}}\right)^+ \\
&= \text{sign}(w^L_{i_L i^k_{L-1}}) \cdot (|w^L_{i_L i^k_{L-1}}| - \alpha\lambda^L_{i_L i^k_{L-1}})^+ \cdot \ldots \text{sign}(w^1_{i^k_1 i_0}) \cdot (|w^1_{i^k_1 i_0}| - \alpha\lambda^1_{i^k_1 i_0})^+ \\
&=: \text{sign}(w^L_{i_L i^k_{L-1}}) \cdot \tilde{w}^L_{i_L i^k_{L-1}} \cdot \ldots \text{sign}(w^1_{i^k_1 i_0}) \cdot \tilde{w}^1_{i^k_1 i_0}.
\end{aligned}$$

The second equality is our assumption, requiring that the proximal operator from Equation (3) can be written as a product of proximal operators from Equation (2). $\lambda^l_{i^k_l i^k_{l-1}} \in [0, |w^l_{i^k_l i^k_{l-1}}|/\alpha]$ is a, possibly unique, penalty for the weight $w^l_{i^k_l i^k_{l-1}}$ and $\tilde{w}^l_{i^k_l i^k_{l-1}} := (|w^l_{i^k_l i^k_{l-1}}| - \alpha\lambda^l_{i^k_l i^k_{l-1}})^+$. The superscript k on the indices marks that different paths go through different nodes in the inner layers.

Taking absolute values of all equations and summing over the equations where the paths belong to the same connection, i.e. they share values for i_0 and i_L and thus have the same path penalty, we obtain, for a given combination of i_0 and i_L

$$\left(\sum_{i_{L-1}=1}^{d_{L-1}} \cdots \sum_{i_1=1}^{d_1} |w^L_{i_L i_{L-1}}| \cdot \ldots |w^1_{i_1 i_0}|\right) \cdot \left(1 - \frac{\alpha\lambda}{(\boldsymbol{W}_{\text{PL}})_{i_L i_0}}\right)^+ = \sum_{i_{L-1}=1}^{d_{L-1}} \cdots \sum_{i_1=1}^{d_1} \tilde{w}^L_{i_L i_{L-1}} \cdot \ldots \tilde{w}^1_{i_1 i_0}$$

which, using the definition of matrix multiplication can be written as

$$\left(\prod_{l=L,\ldots,1} |\boldsymbol{W_l}| \odot \left(1 - \frac{\alpha\lambda}{(\boldsymbol{W}_{\text{PL}})_{i_L i_0}}\right)^+\right)_{i_L i_0} = \left(\prod_{l=L,\ldots,1} \tilde{\boldsymbol{W}}_{\boldsymbol{l}}\right)_{i_L i_0}.$$

Thus, solving Equation (7) results in (absolute) links values being shifted towards zero in such a way that when multiplied into paths, the paths have the correct penalization. The only thing left to do is to restore the signs of the links. ■

Appendix C. Modified Non-Negative Matrix Factorization

We solve the non-negative matrix factorization problem in Equation (7) with coordinate descent, using a modified version of the solver in python's scikit-learn module (Pedregosa et al., 2011), which in turn is based on work by Cichocki and Phan (2009) and Hsieh and Dhillon (2011). The aim is to minimize $||\boldsymbol{V} - \boldsymbol{W} \cdot \prod_{i=1}^{I} \boldsymbol{M_i} \cdot \boldsymbol{H}||^2_F$ for $I \in \mathbb{N}_0$, keeping each entry in $\boldsymbol{W}$, $\boldsymbol{M_i}$ and $\boldsymbol{H}$ between zero and its seed.

Coordinate descent updates each matrix separately, keeping the others fixed. Since we can write $\boldsymbol{W} \cdot \prod_{i=1}^{I} \boldsymbol{M_i} \cdot \boldsymbol{H} = \tilde{\boldsymbol{W}} \cdot \tilde{\boldsymbol{M}} \cdot \tilde{\boldsymbol{H}}$, where the three matrices on the right hand side are products of the matrices of the left hand side, when updating a given matrix, we can always treat the problem as a product of only three matrices. Thus we need update rules

for $\tilde{\boldsymbol{W}}$ and $\tilde{\boldsymbol{M}}$, since we can use the same algorithm for $\tilde{\boldsymbol{W}}$ and $\tilde{\boldsymbol{H}}$ by taking transpose. Hsieh and Dhillon (2011) describe the case for two matrices, i.e. $I = 0$. Our contribution is the update rule for $\tilde{\boldsymbol{M}}$ (and trivially adding the upper constraint on the solution).

C.1 Update Rule for $\tilde{\boldsymbol{W}}$

For $\tilde{\boldsymbol{W}} \in (\mathbb{R}^+)^{d_i \times d_r}$ we want to solve, adding the possibility to add element-wise l_1- and l_2-penalties:

$$\min_{\tilde{\boldsymbol{W}}:\ 0 \leq (\tilde{\boldsymbol{W}})_{ir} \leq (\tilde{\boldsymbol{W}}_{\boldsymbol{0}})_{ir}} \frac{1}{2}||\boldsymbol{V} - \tilde{\boldsymbol{W}}\tilde{\boldsymbol{H}}||_F^2 + \sum_{i,r}\left(\lambda_1(\tilde{\boldsymbol{W}})_{ir} + \frac{\lambda_2}{2}(\tilde{\boldsymbol{W}})_{ir}^2\right)$$

where $\tilde{\boldsymbol{W}}_{\boldsymbol{0}}$ is the seed. We update each element $(\tilde{\boldsymbol{W}})_{ir}$ by adding $s\boldsymbol{E}_{\boldsymbol{ir}}$, for an optimal s, where $\boldsymbol{E}_{\boldsymbol{ir}}$ is a matrix with all zeros, except element (i, r), which is 1. Defining

$$g_{ir}^{\tilde{\boldsymbol{W}}}(s) := \frac{1}{2}\sum_j((\boldsymbol{V})_{ij} - ((\tilde{\boldsymbol{W}} + s\boldsymbol{E}_{\boldsymbol{ir}})\tilde{\boldsymbol{H}})_{ij})^2 + \lambda_1((\tilde{\boldsymbol{W}})_{ir} + s) + \frac{\lambda_2}{2}((\tilde{\boldsymbol{W}})_{ir} + s)^2$$

we get

$$\begin{aligned}
g_{ir}^{\tilde{\boldsymbol{W}}}(s) &= \frac{1}{2}\sum_j((\boldsymbol{V})_{ij} - ((\tilde{\boldsymbol{W}} + s\boldsymbol{E}_{\boldsymbol{ir}})\tilde{\boldsymbol{H}})_{ij})^2 + \lambda_1((\tilde{\boldsymbol{W}})_{ir} + s) + \frac{\lambda_2}{2}((\tilde{\boldsymbol{W}})_{ir} + s)^2 \\
&= \frac{1}{2}\sum_j((\boldsymbol{V})_{ij} - (\tilde{\boldsymbol{W}}\tilde{\boldsymbol{H}})_{ij} - s(\tilde{\boldsymbol{H}})_{rj})^2 + \lambda_1((\tilde{\boldsymbol{W}})_{ir} + s) + \frac{\lambda_2}{2}((\tilde{\boldsymbol{W}})_{ir} + s)^2 \\
(g_{ir}^{\tilde{\boldsymbol{W}}})'(s) &= \sum_j(-(\boldsymbol{V})_{ij}(\tilde{\boldsymbol{H}})_{rj} + (\tilde{\boldsymbol{W}}\tilde{\boldsymbol{H}})_{ij}(\tilde{\boldsymbol{H}})_{rj} + 2s(\tilde{\boldsymbol{H}})_{rj}^2) + \lambda_1 + \lambda_2((\tilde{\boldsymbol{W}})_{ir} + s) \\
(g_{ir}^{\tilde{\boldsymbol{W}}})''(s) &= \sum_j(2(\tilde{\boldsymbol{H}})_{rj}^2) + \lambda_2 \\
(g_{ir}^{\tilde{\boldsymbol{W}}})'(0) &= \sum_{i,j}(-(\boldsymbol{V})_{ij}(\tilde{\boldsymbol{H}}^\top)_{jr} + (\tilde{\boldsymbol{W}}\tilde{\boldsymbol{H}})_{ij}(\tilde{\boldsymbol{H}}^\top)_{jr}) + \lambda_1 + \lambda_2(\tilde{\boldsymbol{W}})_{ir} \\
&= (-\boldsymbol{V}\tilde{\boldsymbol{H}}^\top + \tilde{\boldsymbol{W}}\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top)_{ir} + \lambda_1 + \lambda_2(\tilde{\boldsymbol{W}})_{ir} \\
(g_{ir}^{\tilde{\boldsymbol{W}}})''(0) &= \sum_j(\tilde{\boldsymbol{H}})_{rj}(\tilde{\boldsymbol{H}}^\top)_{jr} + \lambda_2 = (\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top)_{rr} + \lambda_2.
\end{aligned}$$

Since $g_{ir}^{\tilde{\boldsymbol{W}}}(s)$ is quadratic in s, its Taylor expansion only contains terms up to and including s^2:

$$g_{ir}^{\tilde{\boldsymbol{W}}}(s) = g_{ir}^{\tilde{\boldsymbol{W}}}(0) + (g_{ir}^{\tilde{\boldsymbol{W}}})'(0) \cdot s + \frac{1}{2}(g_{ir}^{\tilde{\boldsymbol{W}}})''(0) \cdot s^2$$

which is minimized at

$$s^* = -\frac{(g_{ir}^{\tilde{\boldsymbol{W}}})'(0)}{(g_{ir}^{\tilde{\boldsymbol{W}}})''(0)} = \frac{(\boldsymbol{V}\tilde{\boldsymbol{H}}^\top - \tilde{\boldsymbol{W}}\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top)_{ir} - \lambda_1 - \lambda_2(\tilde{\boldsymbol{W}})_{ir}}{(\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top)_{rr} + \lambda_2}$$

and, to make sure $(\tilde{\boldsymbol{W}})_{ir} \in [0, (\tilde{\boldsymbol{W}}_{\boldsymbol{0}})_{ir}]$:

$$(\tilde{\boldsymbol{W}})_{ir}^{t+1} = \max((\tilde{\boldsymbol{W}}_{\boldsymbol{0}})_{ir}, \min(0, (\tilde{\boldsymbol{W}})_{ir}^t + s^*)).$$

C.2 Update Rule for $\tilde{M}$

For $\boldsymbol{M} \in (\mathbb{R}^+)^{d_p \times d_r}$ the corresponding equations become

$$\min_{\tilde{\boldsymbol{M}}:\ 0 \le (\tilde{\boldsymbol{M}})_{pr} \le (\tilde{\boldsymbol{M}}_{\boldsymbol{0}})_{pr}} \frac{1}{2}||\boldsymbol{V} - \tilde{\boldsymbol{W}}\tilde{\boldsymbol{M}}\tilde{\boldsymbol{H}}||_F^2 + \sum_{p,r}\left(\lambda_1(\tilde{\boldsymbol{M}})_{pr} + \frac{\lambda_2}{2}(\tilde{\boldsymbol{M}})_{pr}^2\right)$$

$$\begin{aligned}
g_{pr}^{\tilde{M}}(s) :=& \frac{1}{2}\sum_{i,j}((\boldsymbol{V})_{ij} - (\tilde{\boldsymbol{W}}(\tilde{\boldsymbol{M}} + s\boldsymbol{E}_{\boldsymbol{pr}})\tilde{\boldsymbol{H}})_{ij})^2 + \lambda_1((\tilde{\boldsymbol{M}})_{pr} + s) + \frac{\lambda_2}{2}((\tilde{\boldsymbol{M}})_{pr} + s)^2 \\
=& \frac{1}{2}\sum_{i,j}((\boldsymbol{V})_{ij} - (\tilde{\boldsymbol{W}}\tilde{\boldsymbol{M}}\tilde{\boldsymbol{H}})_{ij} - s(\tilde{\boldsymbol{W}})_{ip}(\tilde{\boldsymbol{H}})_{rj})^2 + \lambda_1((\tilde{\boldsymbol{M}})_{pr} + s) \\
&+ \frac{\lambda_2}{2}((\tilde{\boldsymbol{M}})_{pr} + s)^2 \\
(g_{pr}^{\tilde{M}})'(s) =& \sum_{i,j}(-(\boldsymbol{V})_{ij}(\tilde{\boldsymbol{W}})_{ip}(\tilde{\boldsymbol{H}})_{rj} + (\tilde{\boldsymbol{W}}\tilde{\boldsymbol{M}}\tilde{\boldsymbol{H}})_{ij}(\tilde{\boldsymbol{W}})_{ip}(\tilde{\boldsymbol{H}})_{rj} + 2s(\tilde{\boldsymbol{W}})_{ip}^2(\tilde{\boldsymbol{H}})_{rj}^2) \\
&+ \lambda_1 + \lambda_2((\tilde{\boldsymbol{M}})_{pr} + s) \\
(g_{pr}^{\tilde{M}})''(s) =& \sum_{i,j}(2(\tilde{\boldsymbol{W}})_{ip}(\tilde{\boldsymbol{H}})_{rj}^2) + \lambda_2 \\
(g_{pr}^{\tilde{M}})'(0) =& \sum_{i,j}(-(\boldsymbol{V})_{ij}(\tilde{\boldsymbol{W}})_{ip}(\tilde{\boldsymbol{H}})_{rj} + (\tilde{\boldsymbol{W}}\tilde{\boldsymbol{M}}\tilde{\boldsymbol{H}})_{ij}(\tilde{\boldsymbol{W}})_{ip}(\tilde{\boldsymbol{H}})_{rj}) + \lambda_1 + \lambda_2(\tilde{\boldsymbol{M}})_{pr} \\
=& (-\tilde{\boldsymbol{W}}^\top\boldsymbol{V}\tilde{\boldsymbol{H}}^\top + \tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{W}}\tilde{\boldsymbol{M}}\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top)_{pr} + \lambda_1 + \lambda_2(\tilde{\boldsymbol{M}})_{pr} \\
(g_{pr}^{\tilde{M}})''(0) =& \sum_i(\tilde{\boldsymbol{W}}^\top)_{pi}(\tilde{\boldsymbol{W}})_{ip}\sum_j(\tilde{\boldsymbol{H}})_{rj}(\tilde{\boldsymbol{H}}^\top)_{jr} + \lambda_2 = (\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{W}})_{pp}(\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top)_{rr} + \lambda_2 \\
s^* =& \frac{(\tilde{\boldsymbol{W}}^\top\boldsymbol{V}\tilde{\boldsymbol{H}}^\top - \tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{W}}\tilde{\boldsymbol{M}}\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top)_{pr} - \lambda_1 - \lambda_2(\tilde{\boldsymbol{M}})_{pr}}{(\tilde{\boldsymbol{W}}^\top\tilde{\boldsymbol{W}})_{pp}(\tilde{\boldsymbol{H}}\tilde{\boldsymbol{H}}^\top)_{rr} + \lambda_2} \\
(\tilde{\boldsymbol{M}})_{pr}^{t+1} =& \max((\tilde{\boldsymbol{M}}_{\boldsymbol{0}})_{pr}, \min(0, ((\tilde{\boldsymbol{M}})_{ir})^t + s^*)).
\end{aligned}$$

References

Samuel K Ainsworth, Nicholas J Foti, Adrian KC Lee, and Emily B Fox. oi-vae: Output interpretable vaes for nonlinear group factor analysis. In *International Conference on Machine Learning*, pages 119–128, 2018.

Oskar Allerbo and Rebecka Jörnsten. Flexible, non-parametric modeling using regularized neural networks. *arXiv preprint arXiv:2012.11369*, 2020.

Andrzej Cichocki and Anh-Huy Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 92(3):708–721, 2009.

Christophe Croux, Peter Filzmoser, and Heinrich Fritz. Robust sparse principal component analysis. *Technometrics*, 55(2):202–214, 2013.

Jason A Dabin, Alexander M Haimovich, Justin Mauger, and Annan Dong. Blind source separation with l1 regularized sparse autoencoder. In *2020 29th Wireless and Optical Communications Conference (WOCC)*, pages 1–5. IEEE, 2020.

Lingling Guo, Ping Wu, Jinfeng Gao, and Siwei Lou. Sparse kernel principal component analysis via sequential approach for nonlinear process monitoring. *IEEE Access*, 7:47550–47563, 2019.

Cho-Jui Hsieh and Inderjit S Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1064–1072, 2011.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

Zhihui Lai, Yong Xu, Qingcai Chen, Jian Yang, and David Zhang. Multilinear sparse principal component analysis. *IEEE Transactions on Neural Networks and Learning Systems*, 25(10):1942–1950, 2014.

Deyu Meng, Qian Zhao, and Zongben Xu. Improve robustness of sparse pca by l1-norm maximization. *Pattern Recognition*, 45(1):487–497, 2012.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401, 2015.

Andrew Ng et al. Sparse autoencoder. *CS294A Lecture Notes*, 72(2011):1–19, 2011.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

R Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898, 1976.

Ferdinando S Samaria and Andy C Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of 1994 IEEE Workshop on Applications of Computer Vision*, pages 138–142. IEEE, 1994.

Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

Alex J Smola, Olvi L Mangasarian, and Bernhard Schölkopf. Sparse kernel feature analysis. In *Classification, Automation, and New Media*, pages 167–178. Springer, 2002.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Michael E Tipping. Sparse kernel principal component analysis. In *Advances in Neural Information Processing Systems*, pages 633–639, 2001.

Duo Wang and Toshihisa Tanaka. Sparse kernel principal component analysis based on elastic net regularization. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3703–3708. IEEE, 2016.

Shanshan Wu, Alex Dimakis, Sujay Sanghavi, Felix Yu, Daniel Holtmann-Rice, Dmitry Storcheus, Afshin Rostamizadeh, and Sanjiv Kumar. Learning a compressed sensing measurement matrix via gradient unrolling. In *International Conference on Machine Learning*, pages 6828–6839. PMLR, 2019.

Jaehong Yoon and Sung Ju Hwang. Combined group and exclusive sparsity for deep neural networks. In *International Conference on Machine Learning*, pages 3958–3966, 2017.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

Ron Zass and Amnon Shashua. Nonnegative sparse pca. In *Advances in Neural Information Processing Systems*, pages 1561–1568, 2007.

Yang Zhou, Rong Jin, and Steven Chu-Hong Hoi. Exclusive lasso for multi-task feature selection. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 988–995, 2010.

Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, 2006.