THESIS FOR THE DEGREE OF LICENTIATE OF PHILOSOPHY

Lasso Regularized Neural Networks

Oskar Allerbo

UNIVERSITY OF GOTHENBURG

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2020

Lasso Regularized Neural Networks
Oskar Allerbo
Gothenburg 2020

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31 772 1000

Typeset with LaTeX
Printed by Stema Specialtryck AB, Borås, Sweden, 2020

Tryckssak
3041 0234

Lasso Regularized Neural Networks

Oskar Allerbo

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg

Abstract

With their flexibility and ability to capture complex, non-linear dependencies in data, artificial neural networks, ANNs, have obtained state of the art performance in a large number of disciplines. Due to their many parameters ANNs are however difficult to interpret. This thesis, that is based on two research papers, investigates how ANNs can be regularized with the lasso penalty to eliminate redundant parameters, resulting in clean-cut, interpretable network architectures.

In Paper I, we use a one hidden layer, feedforward neural network regularized with adaptive lasso to obtain a sparse network architecture. We then interpret this architecture as a data driven non-parametric additive model, as an alternative to a model based on domain knowledge.

In Paper II, we develop a lasso penalty between nodes in non-adjacent layers in a feedforward neural network. When then apply it to an autoencoder, to perform non-linear, sparse dimensionality reduction. We show that our algorithm achieves a lower reconstruction error than linear models, such as sparse principal component analysis, and higher interpretability than non-sparse algorithms, such as autoencoders.

With this thesis we contribute to the understanding of how lasso regularization can be used to produce ANNs with more interpretable architectures, and with maintained flexibility.

Keywords: sparse neural networks, lasso regularization, additive models, dimensionality reduction

List of publications

This thesis contains the following papers:

I. **Allerbo, O.**, Jornsten, R. (2020). Flexible, Non-parametric Modeling Using Regularized Neural Networks. *Manuscript*
II. **Allerbo, O.**, Jornsten, R. (2020). Non-linear, Sparse Dimensionality Reduction via Path Lasso Penalized Autoencoders. *Manuscript*

Acknowledgements

Thank you Jonatan and Anders for helping me navigate as a new PhD student.
Thank you Rebecka for taking time to supervise me, despite your busy schedule.
Thank you current and past PhD students, including hang arounds, for participation in and discussions about beer drinking, blood donation, course assignments, life, linguistics, mathematics, plants, religion, running and teaching; Adam, Anders, Andreas, Anna, Anna, Anton, Barbara, Edvin, Felix, Gabrijela, Helga, Henrik, Hossein, Kristian, Ivar, Jimmy, Jimmy, Johannes, Jonatan, Jonathan, Juan, Linnea, Malin, Marco, Mikael, Niek, Olle, Olof, Sandra and Valentina.
Thank you Ville for encouraging me to apply for the position.
Thank you Annika for all support and for approving my irregular parental leave applications.
Thank you clarity.
Thank you Ulf for helping me when I pour water into my computer.
Thank you Aila and Erik for all types of support in all types of matters.
Thank you Fia and Thomas for making teaching as smooth as possible and for being open to my requests.
Thank you Johan, Loredana, Lotta and Marie for helping me with all administrative things, despite my incapability of remembering the procedures.
Thank you Annika, Giuseppe, Jakob, Jeff, Kristin, Maria and Petter for good and flexible teaching.
Thank you Selma for all understanding and support.
Thank you Malte for being loving and curious; I am sorry about "Pappa jobbar så mycket".

Contents

1 Introduction

Artificial neural networks (ANNs), is a very flexible model class, capable of capturing complex, non-linear relations in data. One reason for the great flexibility is the many parameters. Even if some parameters may be redundant, given enough data, ANNs can still perform very well on prediction tasks. If, however, interpretation is a major concern this plenitude of parameters might be an obstacle.

This licentiate thesis investigates how ANNs with different versions of the lasso penalty (to be defined in section 1.2.1) can be used to create sparse ANNs, that are still flexible, yet more interpretable.

The rest of this chapter is structured as follows: Section 1.1 describes ANNs, what they look like and how to fit them to data; section 1.2 describes regularization, why it is needed, different versions and how it might affect fitting of the model; section 1.3 describes variable importance, how a given output is affected by a given input; and section 1.4 describes dimensionality reduction, how to identify a low-dimensional representation that summarizes the variation of a high-dimensional data set.

1.1 Feedforward Neural Networks

The first ANNs were proposed by McCulloch and Pitts (1943). Hebbian learning, which is a form of unsupervised learning where the data is unlabeled, was introduced shortly thereafter (Hebb (1949)). Backpropagation (Werbos (1974)), which is essentially the chain rule, enabled the use of multi-layered networks, but the computing power at the time limited their usefulness. However, as computational power increased, so did the usefulness of ANNs, and from around 2010, with the publications of Hinton et al. (2006) and Krizhevsky et al. (2012), they have provided state of the art performance in a large variety of

tasks, see e.g. Abiodun et al. (2018) for a review.

Although ANNs exist in many flavours with different architectures, such as convolutional neural networks (LeCun et al. (1989)) and recurrent neural networks (Hochreiter and Schmidhuber (1997)), this thesis focuses on the simplest architecture, the feedforward neural network. In the following, we see how it can be thought of as a generalization of linear regression, before looking at how to infer its parameters to fit the model to data.

1.1.1 Network Architecture

For $\boldsymbol{x}, \boldsymbol{\theta} \in \mathbb{R}^p$, linear regression can be written as

$$\mathbb{E}[y] = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots \theta_p x_p = \theta_0 \cdot 1 + \boldsymbol{\theta}^\top \boldsymbol{x}$$

and drawn as a graph with the y and x's as node values and the θ's as link weights, as sketched in Figure 1.1a.

Logistic regression, sketched in Figure 1.1b, is very similar to linear regression, but with addition of the non-linear sigmoid function, $\sigma(x) = 1/(1 + e^{-x})$:

$$\mathbb{E}[y] = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots \theta_p x_p) = \sigma(\theta_0 \cdot 1 + \boldsymbol{\theta}^\top \boldsymbol{x}).$$

Another modification to standard linear regression is multivariate linear regression; instead of a $\boldsymbol{\theta}$-vector in $\mathbb{R}^p$, we get a $\boldsymbol{\Theta}$-matrix in $\mathbb{R}^{q \times p}$ and for $i \in \{1, \dots q\}$

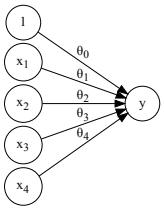
$$\mathbb{E}[y_i] = \theta_{i0} + \theta_{i1} x_1 + \theta_{i2} x_2 + \cdots \theta_{ip} x_p, \; \mathbb{E}[\boldsymbol{y}] = \boldsymbol{\theta_0} + \boldsymbol{\Theta} \cdot \boldsymbol{x}$$

as sketched in Figure 1.1c.

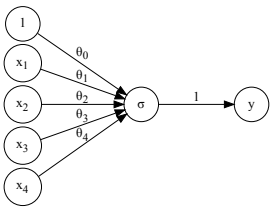
A feedforward neural network can be thought of as multiple instances of logistic and multivariate regression stacked on each other, so that the input of each sigmoid function is a linear combination of outputs of other sigmoid functions, as sketched in Figure 1.1d. For some reason when discussing neural networks, the names of the parameter often differ from those used in linear and logistic regression; instead of $\boldsymbol{\theta_0}$, $\boldsymbol{b}$, as in bias, is used, and instead of $\boldsymbol{\Theta}$, $\boldsymbol{W}$, as in weights. For a feedforward neural network with $L + 1$ layers including the x- and the y-layers, the equation then becomes

$$\mathbb{E}[\boldsymbol{y}] = \sigma(\boldsymbol{W_L} \cdot \sigma(\boldsymbol{W_{L-1}} \cdot \sigma(\dots \sigma(\boldsymbol{W_1} \cdot \boldsymbol{x} + \boldsymbol{b_1}) \dots) + \boldsymbol{b_{L-1}}) + \boldsymbol{b_L})$$

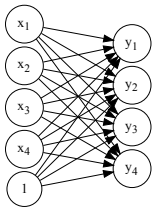
where the sigmoid functions, that are often called the activation functions, are taken element-wise.



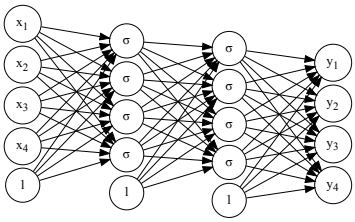
(a) Linear regression



(b) Logistic regression



(c) Multivariate regression



(d) Feedforward neural network

Figure 1.1: Network representations of linear regression, logistic regression, multivariate linear regression and a feedforward neural network. In the figures all layers have four nodes, and there are two hidden layer in the neural network, but there is nothing magic about theses numbers. In the two last subfigures the θ's are omitted to enable readability.

Finally, there is nothing that says that we need to use the sigmoid function as activation function, or that we need to use the same activation function in each layer. Other common choices of activation functions are the identity function, which reduces the neural network to multivariate linear regression; $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$, which is a scaled version of the sigmoid function with range $[-1, 1]$ instead of $[0, 1]$; and the rectified linear unit, $\text{ReLu}(x) = \max(0, x)$.

1.1.2 Network Training

Fitting a model to data, something that is often referred to as training the model, is an optimization problem in terms of the difference between true value, $\boldsymbol{y}$, and the model value, $\hat{\boldsymbol{y}}(\boldsymbol{x})$. This difference is often referred to as the

loss, or the risk, denoted by L and is to be minimized with respect to the model parameters:

$$\min_{\{\boldsymbol{W_i}\}_{i=1}^{L},\{\boldsymbol{b_i}\}_{i=1}^{L}} L(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{x})).$$

Common choices of loss function are the squared difference,

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{x})) := \sum_i (y_i - \hat{y}_i(x_i))^2,$$

for regression and the cross entropy,

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{x})) := \sum_i y_i \log(\hat{y}_i(x_i)),$$

for classification, where $y_i \in \{0, 1\}$.

Using backpropagation it is straight forward, but tedious, to express $\frac{\partial L}{\partial \theta}$ for all elements θ in all weight matrices and bias vectors, which means that the optimization problem can be solved by setting the derivative to zero for all parameters. One of the simplest algorithms for finding the zeros of the derivative is gradient descent, which was suggested by Cauchy in 1847. It is an iterative algorithm, where in each iteration every parameter is updated a small step, α, called the learning rate, in the direction of the negative gradient, i.e. downhill in the multidimensional optimization landscape:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t; \boldsymbol{x}, \boldsymbol{y})$$

If the problem is convex (which is typically not the case for neural networks) the algorithm converges to the global optimum. If it is not, the algorithm converges to some local optimum that depends on the starting point. For ANNs, it has turned out that good results are generally obtained when the parameters are initialized with values close to zero, but they cannot all be exactly zero, because then many of the parameters will get the exact same gradient values and will continue to have the same value throughout training, thus reducing the model capacity (Bengio (2012)).

Many versions of gradient descent exist to improve training speed. Most notable is perhaps Adam (Kingma and Ba (2014)) , that uses individual learning rates for the different parameters and something called momentum. When moving through the optimization landscape using momentum, the history of the optimization path is taken into account, limiting the response to sudden changes of direction. One can think of it as a ball rolling downhill; once the ball has gained speed, it is not too prone to respond to minor slope changes.

Another very common modification of gradient descent is stochastic gradient descent, which is based on work by Robbins and Monro (1951). This is obtained by evaluating the loss not on the entire training data, but on a random subset thereof, thus inserting some randomness. In addition to faster calculations, this results in a slightly different optimization landscape at each iteration and has proven successful to avoid local minima.

1.2 Regularization

When estimating a parameter from data, it is of interest to know how much the estimated parameter value, $\hat{\theta}$, differs from the true, unknown, value, θ. One very common way to measure this is with the mean squared error (MSE), $\mathbb{E}[(\theta - \hat{\theta})^2]$, i.e. the expected squared difference. Remembering that $\hat{\theta}$ is random while θ is not, the MSE can be decomposed into the variance and the bias as:

$$\begin{aligned}\mathbb{E}[(\theta - \hat{\theta})^2] &= \mathbb{E}[\hat{\theta}^2] \underbrace{-\mathbb{E}[\hat{\theta}]^2 + \mathbb{E}[\hat{\theta}]^2}_{=0} - 2\mathbb{E}[\hat{\theta}]\theta + \theta^2 = \text{Var}(\hat{\theta}) + \mathbb{E}[\hat{\theta} - \theta]^2 \\ &= \text{Var}(\hat{\theta}) + \text{Bias}(\theta, \hat{\theta})^2.\end{aligned} \tag{1.1}$$

Traditionally one has restricted oneself to unbiased estimators, but it becomes more common, especially if high variance is expected to be a problem, to allow for biased estimators, where an increase in the bias can be compensated by a larger decrease in the variance.

The simplest example of a biased model is probably ridge regression (Hoerl and Kennard (1970)), where, for n observations and p variables, the objective function is given by

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2 = \sum_{i=1}^{n}(y_i - \boldsymbol{\theta}^\top \boldsymbol{x_i})^2 + \lambda \sum_{j=1}^{p} \theta_j^2.$$

Here λ is called a hyper-parameter and is used to regularize the model, which means making it less flexible, thus reducing the variance but increasing the bias. As seen, ridge regression favours small values of the θ's which makes sense, since θ_j is the slope of x_j. The smaller the slope, the less does the function value change when changing x_j, leading to a more stable behavior.

Introducing hyper-parameters, the loss becomes a function not only of the data and the parameters, $L(\boldsymbol{x}, \boldsymbol{\theta})$, but also of the hyper-parameters, $L(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\lambda})$, and parameter estimation becomes a two-step procedure: First the data is split into

training, validation and testing data, $\boldsymbol{x}_{\text{train}}$, $\boldsymbol{x}_{\text{val}}$ and $\boldsymbol{x}_{\text{test}}$. Then, for some fixed value of $\boldsymbol{\lambda_i}$, $i \in \{1, \dots I\}$, the model is optimized by minimizing $L(\boldsymbol{x}_{\text{train}}, \boldsymbol{\theta}, \boldsymbol{\lambda_i})$,

$$\boldsymbol{\theta_i^*} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ L(\boldsymbol{x}_{\text{train}}, \boldsymbol{\theta}, \boldsymbol{\lambda_i}).$$

The optimal $\boldsymbol{\lambda_i}$ is determined by evaluating the model on the previously unseen validation data:

$$\boldsymbol{\lambda^*} = \underset{\boldsymbol{\lambda_i}, i \in \{1, \dots I\}}{\operatorname{argmin}}\ L(\boldsymbol{x}_{\text{val}}, \boldsymbol{\theta_i^*}, \boldsymbol{\lambda_i}),$$

where $\boldsymbol{\theta_i^*}$ has to be recalculated for each value of $\boldsymbol{\lambda_i}$. Finally, in order to estimate how well the model will do on new data, it can be evaluated on the testing data for the optimal combination of $\boldsymbol{\theta}$ and $\boldsymbol{\lambda}$.

As mentioned, the role of the hyper-parameters is to reduce the complexity of the model. If a model is allowed to be very complex it can easily fit perfectly on the training data, but that does not mean that it will generalize well to new data, a phenomenon known as over-fitting. On the other hand, a simpler model may not capture all the details of the training data (which might be sample specific), but only the more evident tendencies (which are more likely to be present in all data realizations) and thus generalize better. This can be thought of as a trade-off between the variance and bias in Equation 1.1. A simple, highly regularized, model will have a higher bias and a lower variance than a complex, non-regularized model, and somewhere in between is the optimal trade-off.

1.2.1 Lasso Regularization

Lasso regularization was first proposed in Tibshirani (1996), and is formulated as

$$\|\boldsymbol{y} - \boldsymbol{X\theta}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_1 = \sum_{i=1}^{n}(y_i - \boldsymbol{\theta}^\top \boldsymbol{x_i})^2 + \lambda \sum_{j=1}^{p}|\theta_j|, \tag{1.2}$$

i.e. the squared l_2-norm in ridge regression is replaced by the l_1-norm. Just as for ridge regression, small parameter values are encouraged, but, as a consequence of the non-differentiability of the absolute value at zero, it can set some parameters exactly to zero, thus eliminating them from the model. Hence lasso can be seen as a combination of regularization and model selection.

There are several different versions of the lasso, among them the adaptive lasso (Zou (2006)) and the group lasso (Yuan and Lin (2006)). Adaptive lasso is

formulated as

$$\sum_{i=1}^{n}(y_i - \boldsymbol{\theta}^\top \boldsymbol{x_i})^2 + \lambda \sum_{j=1}^{p} \frac{|\theta_j|}{|\hat{\theta}_j|^\gamma}, \tag{1.3}$$

where $\hat{\theta}_j$ is the ridge estimate of θ_j and $\gamma > 0$ is a new hyper-parameter. Hence, each parameter is assigned an individual penalty, and parameters with ridge estimates close to zero are penalized harder than parameters with large ridge estimates.

The group lasso is formulated as follows:

$$\|\boldsymbol{y} - \boldsymbol{X\theta}\|_2^2 + \lambda \sum_{k=1}^{G} \|\boldsymbol{\theta_k}\|_2 = \sum_{i=1}^{n}(y_i - \boldsymbol{\theta}^\top \boldsymbol{x_i})^2 + \lambda \sum_{k=1}^{G} \sqrt{\sum_{j \in g_k} \theta_j^2}, \tag{1.4}$$

where each parameter is assigned to one of G groups, g_k contains the indices of group k and $\boldsymbol{\theta_k}$ denotes a sub-vector of $\boldsymbol{\theta}$, including the elements that are represented in g_k. The parameters in a group are penalized together, either all parameters in the group are included in the model, or none is.

1.2.2 Proximal Gradient Descent

Because of the non-uniqueness of the derivative of the absolute value at zero, gradient descent will not produce exact zeros if used for optimizing Equations 1.2 to 1.4. Instead proximal gradient descent (Rockafellar (1976)) has to be used. If the objective function f can be decomposed into $f(\boldsymbol{\theta}) = g(\boldsymbol{\theta}) + h(\boldsymbol{\theta})$, where g is differentiable and h is not, a standard gradient descent step followed by a proximal gradient descent step is defined as

$$\boldsymbol{\theta_{t+1}} = \text{prox}_{\alpha h}(\boldsymbol{\theta_t} - \alpha \nabla g(\boldsymbol{\theta_t})).$$

Here, prox is the proximal operator for h, defined as

$$\text{prox}_{\alpha h}(\boldsymbol{\theta}) := \underset{\boldsymbol{x}}{\text{argmin}} \frac{1}{2\alpha}\|\boldsymbol{x} - \boldsymbol{\theta}\|_2^2 + h(\boldsymbol{x}),$$

and α is the learning rate just as before. The proximal operators for ridge regression, lasso and group lasso are stated in Equation 1.5. Note that the proximal operator for adaptive lasso is the same as for standard lasso with $\lambda_i = \lambda/|\hat{\theta}_i|^\gamma$.

$$
\begin{aligned}
&\text{Ridge:} && \text{prox}_{\alpha\lambda\|\cdot\|_2^2}(\theta_i) = \frac{\theta_i}{1+\alpha\lambda} \\
&\text{Lasso:} && \text{prox}_{\alpha\lambda\|\cdot\|_1}(\theta_i) = \text{sign}(\theta_i)\cdot\max(0, |\theta_i| - \alpha\lambda) \\
& && = \theta_i \cdot \max\left(0, 1 - \frac{\alpha\lambda}{|\theta_i|}\right) \\
&\text{Group Lasso:} && \text{prox}_{\alpha\lambda\sum_k \|\cdot\|_2}(\theta_i) = \theta_i \cdot \max\left(0, 1 - \frac{\alpha\lambda}{\sqrt{\sum_{\theta_j \in g_i} \theta_j^2}}\right)
\end{aligned}
\tag{1.5}
$$

where g_i is the group that θ_i belongs to. It should be noted that since the squared l_2-norm is differentiable, standard gradient descent can be used, so in that case there is no need to use proximal gradient descent.

As seen, for lasso, if $|\theta_i| < \alpha\lambda$, applying the proximal operator will result in an exact zero, while for ridge, if $\theta_i \neq 0$, applying the proximal operator can only result in a very small value, but never an exact zero. Furthermore, if there is one group for each θ_i, the proximal operators for lasso and group lasso coincide.

1.3 Variable Importance and Interpretability

Variable importance is the process of inferring how important a specific input variable is for the prediction of a specific output variable, where he importance of input x_j for output y_i is sometimes denoted as $I(y_i, x_j)$. Because of the complexity of ANNs, it is often difficult to determine how an input value propagates through the network and due to the non-linearity, the importance of one variable usually depends on its own, and other variables', values in a non-trivial way.

One of the most straight forward ways to estimate the importance of a variable, and which is totally model agnostic, is to perturb its value while keeping all other inputs fixed, either successively varying it, as done by Lek et al. (1996), or by adding noise, as in Scardi and Harding Jr (1999).

Treating an ANN less as a black box, there are methods that look at the network weights, such as Olden and Jackson (2002), which looks at the products of the weight matrices, and Garson (1991), which uses the absolute values of the weights to associate the output weights with the input neurons.

Even more sophisticated are derivative based methods, where $I(y_i, x_j)$ is a function of $\frac{\partial y_i}{\partial x_j}$. These include saliency maps (Simonyan et al. (2014)), with $I(y_i, x_j) := \left|\frac{\partial y_i}{\partial x_j}\right|$; gradient times input (Shrikumar et al. (2016)), with $I(y_i, x_j) := \frac{\partial y_i}{\partial x_j} \cdot x_j$; and integrated gradient (Sundararajan et al. (2017)), which integrates $\frac{\partial y_i}{\partial \bar{x}_j}$ between x_j and a baseline $\bar{x}_j$ (which is often zero).

Lasso regularization can be thought of as providing some sort of variable importance, by eliminating model parameters so that $I(y_i, x_j)$ is either equal to, or distinct from, zero, regardless of the data. This is of course limited, since it only tells whether there is a connection or not, not how strong it is. On the other hand, when drawing the model as a graph, as in Figure 1.1, if some parameters, i.e. some links, are zero, this will make the graph sparser and (hopefully) more interpretable.

1.4 Dimensionality Reduction

Often when a data set contains many variables, or dimensions, some of them represent approximately the same type of information, e.g. both the weight and the height of a person are measures of its size, and can be expected to be correlated. To increase the interpretability and decrease the size of the data, it is sometimes desirable to reduce its dimensionality, by combining the original dimensions into fewer, so called latent, dimensions in such a way that as much information as possible is conserved.

1.4.1 Principal Component Analysis

The most widely used dimensionality reduction algorithm is probably Principal Component Analysis (PCA, Pearson (1901)). This is a linear technique where the data matrix is rotated so that the first principal axis is the direction along which the variance of the data is maximized, the second axis the direction along which the yet unexplained variance is maximized and so on. These directions correspond to the eigenvectors of the sample covariance matrix, ordered by the size of their eigenvalues. Once these eigenvectors are found, dimensionality reduction is done by simply discarding as many vectors as desired, starting with the ones corresponding to the smallest eigenvalues.

One potential drawback of PCA is that all original dimensions are represented in all latent dimension, which complicates interpretability. A solution to this is

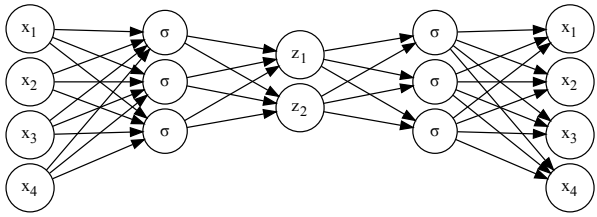


Figure 1.2: Autoencoder mapping from 4 two 2 dimensions. Original dimensions are denoted by x and latent dimensions by z.

sparse PCA (Zou et al. (2006)), which uses lasso regularization to ensure that each original dimension is connected only to a subset of the latent dimensions.

1.4.2 Autoencoders

Another potential drawback of PCA is that it is linear and thus can capture only linear dependencies. A more flexible dimensionality reduction algorithm is the neural network based autoencoder (Kramer (1991)). The architecture of this network is hour glass shaped as sketched in Figure 1.2, with the same number of dimensions (nodes) in the input and output layers, and a smaller number of nodes in the middle layer. Then the same data is given as input and output, so that the network just reconstructs the input, but via a low-dimensional representation.

2 Summary of Papers

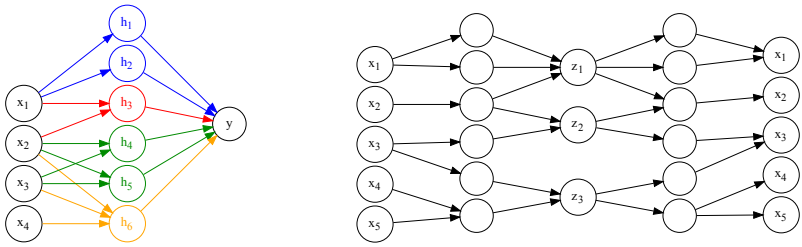
2.1 Paper I

We investigated how adaptive lasso could be used to increase the interpretability of a neural network by eliminating less important links. We found that when using one hidden layer and one output node, it could be used for non-parametric additive modelling, as sketched in Figure 2.1a, where the network corresponds to

$$\mathbb{E}[y] = f_1(x_1) + f_2(x_1, x_2) + f_3(x_2, x_3) + f_4(x_2, x_3, x_4)$$

where f_1 is realized by the hidden nodes h_1 and h_2, f_2 by h_3, f_3 by h_4 and h_5, and f_4 by h_6.

We applied the algorithm to a data set with air pollution in the U.K., presented in Wood et al. (2017), and compared it to the model suggested by the authors, where the functions were manually selected. We concluded that our algorithm selected similar functions with the same predictive performance. Replacing adaptive lasso with standard lasso resulted in functions with more input variables, thus being less interpretable. From the 11 available covariates, the most complex function selected when using adaptive lasso contained three covariates, while the corresponding number for standard lasso was nine.



(a) Sketch of network in Paper I

(b) Sketch of network in Paper II

Figure 2.1: Sketches of network architectures produced in the two papers. Figure 2.1a shows a neural network with one hidden layer, regularized with adaptive lasso. The four subfunctions are color coded to increase readability. In Figure 2.1b a path lasso regularized autoencoder is sketched. The z-nodes are connected only to a subset of the x-nodes and vice versa.

2.2 Paper II

In this paper we developed something we called path lasso, which builds on group lasso and penalizes the number of connected nodes in two non-adjacent layers. The sketch in Figure 2.1b shows that the only node in the third layer connected to node x_1 is z_1. We then showed how the penalty on a path, consisting of multiple links between two nodes, can be translated to individual penalties on the links using non-negative matrix factorization.

We applied path lasso to an autoencoder as sketched in Figure 2.1b to obtain non-linear, sparse dimensionality reduction. Compared to sparse PCA our algorithm is more flexible, which allows it to use the latent space more efficiently and compared to a standard autoencoder it is more interpretable since each latent dimension is a function only of a subset of the original dimensions.

We tested the path lasso penalized autoencoder on synthetic, text and image data.

Bibliography

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., and Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

Garson, D. G. (1991). Interpreting neural network connection weights.

Hebb, D. O. (1949). *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall.

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

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

Lek, S., Belaud, A., Baran, P., Dimopoulos, I., and Delacoste, M. (1996). Role of some environmental variables in trout abundance models using neural networks. *Aquatic Living Resources*, 9(1):23–29.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Olden, J. D. and Jackson, D. A. (2002). Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, 154(1-2):135–150.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Rockafellar, R. T. (1976). Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898.

Scardi, M. and Harding Jr, L. W. (1999). Developing an empirical model of phytoplankton primary production: a neural network case study. *Ecological modelling*, 120(2-3):213–223.

Shrikumar, A., Greenside, P., Shcherbina, A., and Kundaje, A. (2016). Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps.

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*.

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

Werbos, P. (1974). Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*.

Wood, S. N., Li, Z., Shaddick, G., and Augustin, N. H. (2017). Generalized additive models for gigadata: modeling the uk black smoke network daily data. *Journal of the American Statistical Association*, 112(519):1199–1210.

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

Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429.

Zou, H., Hastie, T., and Tibshirani, R. (2006). Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286.

PAPER I

I

FLEXIBLE, NON-PARAMETRIC MODELING USING REGULARIZED NEURAL NETWORKS

Oskar Allerbo
Mathematical Sciences
University of Gothenburg and Chalmers University of Technology
allerbo@chalmers.se
Rebecka Jörnsten *
Mathematical Sciences
University of Gothenburg and Chalmers University of Technology
jornsten@chalmers.se

ABSTRACT

Neural networks excel in terms of predictive performance, with little or no need for manual screening of variables or guided definition of network architecture. However, these flexible and data adaptive models are often difficult to interpret. Here, we propose a new method for enhancing interpretability, that builds on proximal gradient descent and adaptive lasso, PrAda-net. In contrast to other lasso-based algorithms, PrAda-net penalizes all network links individually and, by removing links with smaller weights, automatically adjusts the size of the neural network to capture the complexity of the underlying data generative model, thus increasing interpretability. In addition, the compact network obtained by PrAda-net can be used to identify relevant dependencies in the data, making it suitable for non-parametric statistical modelling with automatic model selection. We demonstrate PrAda-net on simulated data, where we compare the test error performance, variable importance and variable subset identification properties of PrAda-net to other lasso-based approaches. We also apply Prada-net to the massive U.K. black smoke data set, to demonstrate the capability of using Prada-net as an alternative to generalized additive models (GAMs), which often require domain knowledge to select the functional forms of the additive components. Prada-net, in contrast, requires no such pre-selection while still resulting in interpretable additive components.

1 Introduction

Generalized additive models, GAMs ([1]), combine the flexibility of non-parametric and semi-parametric modeling with the interpretability of generalized linear models. In GAMs, the univariate response variable is modeled as a sum of smooth functions of one or several covariates, where the functions may be parametric (e.g. polynomials), non-parametric (e.g. splines), or combinations of the two. For example, in

$$\mathbb{E}[y] = f_1(x_1) + f_2(x_2) + f_3(x_3) + f_{1,2}(x_1, x_2), \tag{1}$$

the response, y, is modelled as a sum of one bivariate and three univariate functions. These functions need to be estimated and since they can be chosen from a large class, GAMs become very flexible and capable of capturing complex, non-linear relations in the data.

Neural networks also have the capacity to learn complex dependencies from data and can capture non-linear relationships between input and output variables, as well as interactions between variables. Contrary to GAMs though, the results do not easily transfer to an interpretable model, since there will be links between all inputs and outputs through the inner layers. However, in all statistical models, some parameters contribute more to the prediction than others, and there is reason to believe that if the less important links are removed from the network architecture, a GAM-like structure would appear, as sketched in Figure 1.

*corresponding author

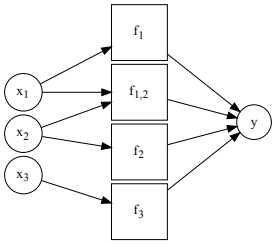


Figure 1: Desired architecture for an interpretable neural network representing $\mathbb{E}[y] = f_1(x_1) + f_2(x_2) + f_3(x_3) + f_{1,2}(x_1, x_2)$. The circles represent single nodes and the squares represent combinations of nodes.

A popular approach to eliminate parameters from a model is $l1$-penalization, hereon after referred to as lasso ([2]), which, together with its extensions such as the group lasso ([3]) and the adaptive lasso ([4]), has been used in neural networks to reduce complexity and increase interpretability. Group lasso is used to penalize all the weights in a predefined group, typically all the inputs or outputs from a node or, in the case of convolutional neural networks, a channel ([5], [6], [7], [8], [9], [10]). An alternative to group lasso is to add extra parameters that are multiplied to all the weights in a group and then apply standard lasso to these parameters ([11], [12], [13], [14]). Adaptive lasso, where each parameter is assigned a unique penalty, has been used in [15] and [16] to improve estimation stability, but without focus on interpretability. Works cited above that focus on interpretability are [9], where Granger causality between time series is restrained; [10], where multiple decoders are connected only to a subset of the latent variables generated by variational autoencoders; and, most relevant for our work, [13], Distillation Guided Routing (DGR) and [14], Explainable Neural Networks (xNN).

In DGR, unimportant channels in a trained convolutional neural network are removed by multiplying the outputs from each channel with a lasso penalized parameter, balancing removal of channels with maintained performance. Since DGR removes entire channels, it can never remove unimportant links from important nodes. In Figure 1 this corresponds to penalizing only the weights between the f's and y.

In xNN, parallel neural networks with lasso penalized inputs are used to model each function in an additive model. xNN removes individual links, but only between the inputs and the, still black-box, parallel networks. In Figure 1, each f-box would correspond to a neural network of pre-defined capacity, where the inputs of the functions are chosen by the model, but where function complexities are pre-defined.

Here, we propose PrAda-net, a novel method that combines adaptive lasso for individual penalties of network parameters with proximal gradient descent. PrAda-net addresses two problems with lasso regularization of neural networks. First, lasso penalizes all model parameters equally, yet it is reasonable to assume that in an overparameterized neural network some weights contribute more to the final result than others, and should ideally be penalized less. Second, gradient descent algorithms cannot handle the non-differentiability of the lasso penalty at zero, something that has to be circumvented, e.g. by defining $\left.\frac{\partial |x|}{\partial x}\right|_{x=0} := 0$ or by using some smooth approximation, such as $|x| \approx \sqrt{x^2 + \varepsilon}$. However, such approximations result in no parameters set exactly to zero, introducing the need for thresholding, which requires tuning as well.

PrAda-net can be thought of as a generalization of xNN, where the number of hidden nodes to realize a function is adaptively chosen, depending on the function complexity. Therefore, it both improves the interpretability and reduces the number of parameters. Compared to DGR, PrAda-net penalizes not only the outputs, but also the inputs, of the functions, resulting in a clear-cut identification of functional components.

The rest of this paper is structured as follows: In Section 2, we present the PrAda-net algorithm. In Section, 3 we illustrate the method on simulated data, comparing the predictive and interpretable performance of PrAda-net to that of standard lasso and DGR. We also apply PrAda-net to U.K. black smoke data. We compare the model components automatically selected by PrAda-net to the large-scale generalized additive model recently presented in [17]. We show how the method is able to automatically select the network complexity to generate a concise, interpretable architecture,

thus making it suitable for non-parametric additive modelling. Finally, in Section 4, we discuss our findings and possible future directions for research.

2 Method

In this paper the following notation will be used: The input nodes of a neural network will be denoted by the vector $\boldsymbol{x}$, with elements x_i, and the single output node will be denoted by y. Functions realized by hidden nodes of the network will be denoted by $f_i(\boldsymbol{x}|\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ denotes the vector of network parameters, which will interchangeably will be referred to as parameters, weights and links. Usually, $\boldsymbol{\theta}$ will be implicit, writing only $f_i(\boldsymbol{x})$. A hat over a parameter or a function, e.g. $\hat{y}$, means that this is the reconstructed parameter or function inferred by the network; when it is clear from the context, the hat will be omitted.

As sketched in Figure 1, the architecture of a sparse neural network can be interpreted as an additive model, where each function is defined as the set of hidden nodes connected to the same set of input nodes. In order to use lasso to obtain such an architecture, the two limitations stated above have to be overcome. First, lasso penalizes all parameters equally, while ideally, if the true model were known, one would want a very small penalty on the true model parameters, and a large penalty on parameters that should not be included. It was shown in [4] that under certain conditions, standard lasso is not consistent and as a remedy the adaptive lasso was proposed. In adaptive lasso, each parameter obtains an individual penalty based on its ordinary least square (OLS) estimate, penalizing parameters with small OLS values more.

Second, lasso regularized models trained with gradient descent do not obtain explicit zeroes, which is a consequence of the non-uniqueness of the derivative of the lasso penalty at zero. Proximal gradient descent ([18]), on the other hand, leverages on this non-uniqueness and obtains exact zeros. Like standard gradient descent, proximal gradient descent is an iterative method, with the difference that the non-differentiable parts of the objective function are handled in an additional, proximal, step. If the objective function, $f(\boldsymbol{x})$, can be decomposed into $f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$, 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

$$\boldsymbol{x}^{t+1} = \text{prox}_{\alpha h}(\boldsymbol{x}^t - \alpha \nabla g(\boldsymbol{x}^t))$$

where α is a step size and prox is the proximal operator, that depends on h. In the case of lasso with penalty parameter λ, when $h(\boldsymbol{x}) = \lambda||\boldsymbol{x}||_1 = \lambda \sum_i |x_i|$, the proximal operator decomposes component-wise and is, with $z_i := (x_i - \alpha \nabla g(x_i))$ denoting the output of the standard gradient descent step,

$$\text{prox}_{\alpha h}(z_i) = \text{sign}(z_i) \cdot \max(|z_i| - \alpha\lambda, 0).$$

I.e., each z_i is additively shrunk towards zero, and once it changes sign it is set to exactly zero.

2.1 The PrAda-net Algorithm

PrAda-net builds on the assumption that once a neural network is trained, even if all nodes in two subsequent layers are connected, some of the weights are more important than others, and that these important weights are larger in absolute value then the less important ones. Thus, adding an adaptive lasso penalty, i.e. an individual penalty based on the current value of the weight, and continue training will penalize the small, unimportant, weights more than the important ones, ideally setting all unimportant weights to zero, while penalizing the important weights less, thus reducing bias. Applying this will change the objective function as below.

$$\underset{\boldsymbol{\theta}}{\text{argmin}} f(\boldsymbol{x}|\boldsymbol{\theta}) \implies \underset{\boldsymbol{\theta}}{\text{argmin}} \left(f(\boldsymbol{x}|\boldsymbol{\theta}) + \lambda \sum_i \frac{|\theta_i|}{|\hat{\theta}_i|^\gamma} \right) \tag{2}$$

where $\hat{\theta}_i$ denotes the value of the weight just before adding the penalty and $\gamma > 0$ is a tunable parameter, that we here set to 2 which is common practice in adaptive lasso.

In order to get exact zeros one would like to use proximal gradient descent for this phase of the training. However, this algorithm is not compatible with momentum based optimization algorithms, such as Adam ([19]). Furthermore, the choice of the step size α, is not as obvious as in Adam, where the default parameters have shown to work very well in most cases. We circumvent these issues by training PrAda-net in three stages: In the first stage, the network is trained without lasso penalty with a standard optimizer until convergence or some other stopping criterion. In the second stage, the adaptive lasso penalty is turned on, using the same optimizer as in the first stage with $\frac{\partial |\theta|}{\partial \theta}\Big|_{\theta=0} := 0$, thus obtaining some weights close to, but distinct from, zero. Before the third stage the adaptive lasso penalty is updated based on the

current weight values and then proximal gradient descent is used. Even with small values of α and λ (10^{-5} was used for both α and λ), small weights will be heavily penalized and proximal gradient descent converges in relatively few iterations. PrAda-net is summarized in Algorithm 1.

Algorithm 1 PrAda-net

1: Train the neural network until convergence or other stopping criterion, with optimizer of choice.
2: Add an adaptive lasso penalty to each weight according to Equation 2. Continue training with the chosen optimizer and $\frac{\partial|\theta|}{\partial\theta}\Big|_{\theta=0} := 0$.
3: Update the adaptive lasso penalty. Continue training with proximal gradient descent.

2.2 Identifying Linear Functions

As activation function for the hidden nodes, tanh was used, which, as can be seen from its Maclaurin expansion $\tanh(x) = x + \mathcal{O}(x^3)$, is almost linear if the absolute value of x is small. For x_1 close to zero, $\tanh(x_1 + x_2) \approx x_1 + \tanh(x_2)$, and therefore, if the output is linear in some input variable (x_1 in this example), no separate hidden node is needed to model this linear function. Instead, the function can be incorporated into some other node, thus increasing compactness but decreasing interpretability. In order to identify which nodes that contain linear functions, the following postprocessing step was added: For each identified function, the partial derivatives with respect to all its inputs were calculated and, if for some input, the derivative was close to constant, i.e. with variance lower than some σ^2_{max}, that input is removed from the function and into a new, linear, function, realized by a new node. We consistently used $\sigma^2_{\text{max}} = 0.01$.

2.3 Choosing the Lasso Penalty Parameter

Choosing the optimal regularization parameter, λ^*, is a trade off between interpretability and model performance. When performance is the sole priority, the optimal regularization is often chosen so that it minimizes the mean test error across multiple splits of the data into training and test sets (cross-validation). Here, we are willing to sacrifice some performance to gain interpretability and use a similar approach as in standard lasso regression packages ([20]). We thus choose our λ^* as the largest one of those whose resulting mean test error is within two standard deviations of the minimum test error across all λ. Figure 2c in Section 3.1 illustrates this.

3 Experiments

All experiments in this section were done on neural networks with one hidden layer with tanh as activation function, and one output node with a linear activation function. The mean squared error was used as loss function and the Adam optimizer was used in the two first stages of the PrAda-net algorithm. The data was randomly split into 90 % training and 10 % testing data. When choosing λ^*, 20 random splits of the data were made for each λ value. In order to escape suboptimal minima, five different initializations were used for each training. All computations were done on a NVIDIA V100, 32GB, GPU.

Three experiments were performed. In the first experiment, synthetic data was generated from the first four Legendre polynomials; in the second experiment, we used a data set with black smoke levels in the U.K. recently analyzed by using a large-scale generalized additive model in [17]; and in the third experiment synthetic data was generated from the model inferred by PrAda-net on the U.K. black smoke data.

3.1 Legendre Polynomials

Our first experiment was done on the sum of the first four Legendre polynomials, which are orthogonal on the interval $[-1, 1]$ and are given by

$$\begin{aligned} P_1(x) &= x \\ P_2(x) &= \frac{1}{2}(3x^2 - 1) \\ P_3(x) &= \frac{1}{2}(5x^3 - 3x) \\ P_4(x) &= \frac{1}{8}(34x^4 - 30x^2 + 3). \end{aligned} \tag{3}$$

Table 1: Legendre polynomials: Mean and one standard deviation of test error and estimated variable importance ($\frac{1}{|\mathcal{X}|}\sum_{\mathcal{X}}\left|\frac{\partial f(x)}{\partial x_i}\right|$) together with the true asymptotic variable importance ($\frac{1}{2}\int_{-1}^{1}|\frac{\partial P_i(x_i)}{\partial x_i}|dx_i$).

	True Var.Imp. $\frac{1}{2}\int_{-1}^{1}\|\frac{\partial P_i(x_i)}{\partial x_i}\|dx_i$	PrAda-net	Standard Lasso	DGR
Test error	-	0.04 ± 0.02	0.13 ± 0.03	0.16 ± 0.03
x_1	1	0.75 ± 0.33	0.69 ± 0.43	0.63 ± 0.09
x_2	1.5	1.26 ± 0.09	1.21 ± 0.05	0.65 ± 0.12
x_3	1.89	1.42 ± 0.16	0.96 ± 0.11	0.32 ± 0.12
x_4	2.23	1.72 ± 0.3	0.43 ± 0.24	0.25 ± 0.08
x_5	0	0.0 ± 0.0	0.01 ± 0.01	0.07 ± 0.04

For the simulations, five random variables, $x_1, \dots x_5$, were generated from $\mathcal{U}[-1, 1]$, each with 1000 realizations, resulting in a 1000×5 x-matrix, from which a 1000×1 y-matrix was created according to

$$y = P_1(x_1) + P_2(x_2) + P_3(x_3) + P_4(x_4) + \varepsilon \tag{4}$$

where $\varepsilon \sim \mathcal{N}(0, 0.1^2)$ is added noise. Note that y is independent of x_5. A neural network with five input nodes and 50 hidden nodes was trained by applying Algorithm 1 with λ^* chosen as explained in Section 2.3.

The results are shown in Figure 2. As seen in 2a, out of the original 50 hidden nodes, only 10 are used in the final architecture and the hidden units are split into four different functions, each with only one x_i as input. Figure 2b shows the reconstructed $\hat{P}_i(x)$'s (left-most panels) and also how the model decomposes each $\hat{P}_i(x)$ into a sum of up to four subfunctions, each represented by one node in the hidden layer. Figure 2c demonstrates the penalty parameter selection for this simulation.

We compared the test error and the variable importance, measured using saliency maps ([21]) for PrAda-net, standard lasso and DGR. To make standard lasso more competitive, we did not apply lasso regularization from the start. Instead, we utilized the PrAda-net algorithm with the same penalty for all links, i.e. $\gamma = 0$ in Equation 2. DGR was adapted to feedforward regression networks by looking at nodes instead of channels and by using the squared error instead of the cross-entropy loss.

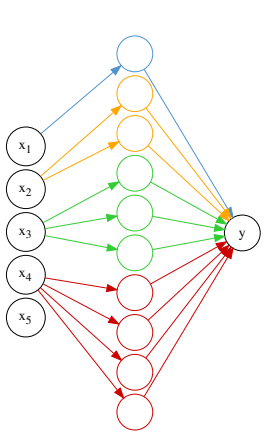
For saliency maps, the importance of variable i, evaluated at $\boldsymbol{x}$, is defined as $I_i(\boldsymbol{x}) := \left|\frac{\partial f(\boldsymbol{x})}{\partial x_i}\right|$, which in our case reduces to $I_i(\boldsymbol{x}) = \left|\frac{\partial P_i(x_i)}{\partial x_i}\right|$. Since for non-linear functions the value of the derivative depends on $\boldsymbol{x}$, in order to get a global importance measure, we averaged over all $\boldsymbol{x}$'s in the test set $\mathcal{X}$, $I_i := \frac{1}{|\mathcal{X}|}\sum_{\mathcal{X}} I_i(\boldsymbol{x})$. This can be seen as a Monte Carlo integral, meaning that as $|\mathcal{X}| \to \infty$, $I_i \to \frac{1}{2}\int_{-1}^{1}|\frac{\partial P_i(x_i)}{\partial x_i}|dx_i$, where $\frac{1}{2}$ is the probability density function of the uniform distribution that x_i is sampled from. Thus, for large sample sizes, the true value of the variable importance is given by the analytical integral.

50 runs with different noise realizations were performed with PrAda-net penalized with λ^*, and the other two models penalized to equal complexity. In the case of standard lasso, the regularization was chosen to obtain the same number of parameters in both models, while for DGR this would be an unfair comparison, since penalization is done at node level. Instead, for DGR, regularization was chosen to have the same number of nodes as PrAda-net, allowing it to use many more parameters than PrAda-net and standard lasso.

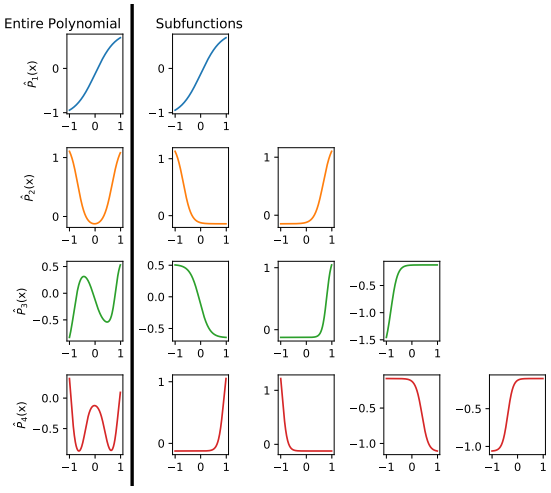
The results are summarized in Table 1. PrAda-net outperforms both lasso and DGR, both in terms of test error and variable importance. This effect is especially notable for the higher order functions. PrAda-net is also the only model that correctly identifies the noise variable as such.

We also investigated how well the algorithms could identify the four different functions in the true model. In this comparison, DGR was left out since all input nodes are connected to all nodes in the hidden layer, resulting in only $f(x_1, x_2, x_3, x_4, x_5)$ being identified. Standard lasso was penalized with two different strengths, one to get the same number of parameters as for PrAda-net and one to get the same number of functions.

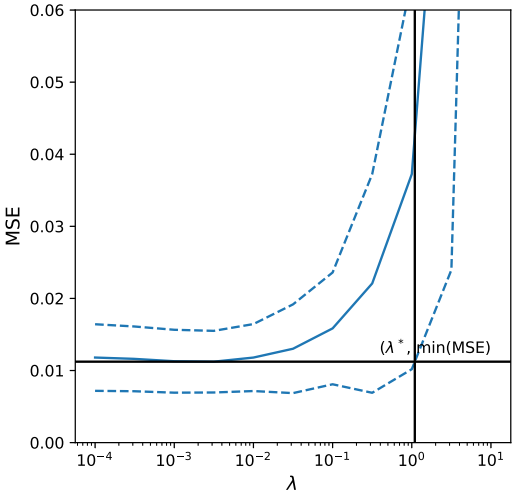
Table 2 shows that PrAda-net is able to identify the true functions in the model as well as their complexity, measured in number of nodes. That is, PrAda-net assigns more nodes to the higher order polynomials. The true functions are identified much more often by PrAda-net than by standard lasso. Standard lasso with the same complexity erroneously includes higher-order interaction terms (e.g. $f(x_1, x_2, x_3)$) rather than increasing the presence or function complexity of the main effect terms (e.g. $f(x_2)$). Standard lasso with the same number of functions successfully identifies $f_1(x_1)$ and $f_2(x_2)$ but fails to identify $f_3(x_3)$ and $f_4(x_4)$.



(a)



(b)



(c)

Figure 2: Identifying the sum of the four first Legendre polynomials. 2a shows the inferred reduced neural network, with the hidden nodes and links color coded according to the identified functions. In 2b, the leftmost column shows, from top to bottom, $\hat{P}_1(x_1)$, $\hat{P}_2(x_2)$, $\hat{P}_3(x_3)$ and $\hat{P}_4(x_4)$, while the other columns show their decompositions into the subfunctions realized by single hidden nodes. 2c shows, the mean $\pm$ two standard deviations of the test error obtained by 20 bootstrap runs for each λ. λ^* is chosen as the value where the lower interval intersects the lowest mean value.

Table 2: Legendre polynomials: Presence (proportion of simulations where function is identified) and average complexity of functions across 50 simulation runs. The true model components (main effects) in bold. Complexity of identified functions is measured in number of nodes.

	PrAda-net		Standard Lasso (same number (of functions)		Standard Lasso (same complexity)	
Function	Presence	Comp-lexity	Presence	Comp-lexity	Presence	Comp-lexity
$\boldsymbol{f(x_1)}$	1.0	1.0	1.0	1.72	0.12	1.0
$f(x_1, x_2)$	-	-	-	-	0.12	1.17
$f(x_1, x_2, x_3)$	-	-	-	-	0.14	1.0
$f(x_1, x_2, x_3, x_4)$	-	-	-	-	0.82	1.46
$f(x_1, x_2, x_3, x_4, x_5)$	-	-	-	-	0.44	1.0
$f(x_1, x_2, x_3, x_5)$	-	-	-	-	0.02	1.0
$f(x_1, x_3)$	-	-	-	-	0.96	1.06
$f(x_1, x_3, x_4)$	-	-	-	-	0.64	1.25
$f(x_1, x_3, x_4, x_5)$	-	-	-	-	0.06	1.0
$f(x_1, x_4)$	-	-	-	-	0.42	1.0
$\boldsymbol{f(x_2)}$	1.0	1.96	0.92	2.0	0.14	1.14
$f(x_2, x_3)$	-	-	0.08	2.0	0.02	1.0
$f(x_2, x_3, x_4)$	-	-	-	-	0.04	1.0
$f(x_2, x_3, x_5)$	-	-	-	-	0.02	1.0
$f(x_2, x_4)$	0.04	1.0	-	-	0.08	1.0
$f(x_2, x_5)$	-	-	-	-	0.02	1.0
$\boldsymbol{f(x_3)}$	1.0	2.62	0.1	1.6	0.2	1.6
$f(x_3, x_4)$	0.04	1.0	0.02	1.0	0.1	1.2
$\boldsymbol{f(x_4)}$	1.0	3.98	0.06	1.0	0.96	1.83

3.2 Black Smoke Data

To illustrate the usefulness of PrAda-net on real, large-scale data with a complex structure, we applied it to the U.K. black smoke network daily data set, recently presented in [17] and available on the author's homepage[2]. The data set, collected over four decades, is massive, comprising 10 million observations of air pollution data (daily concentration of black smoke particulates in μgm^{-3}) measured at more than 2000 monitoring stations. In addition to the pollution data, `bs`, the following eleven covariates are available: year, `y`; day of year, `doy`; day of week, `dow`; location as kilometers east, `e` and north, `n`; height, `h`; cubic root transformed rainfall, `r`; daily minimum and maximum temperature $\texttt{T}^0$ and $\texttt{T}^1$; and mean temperatures the previous two days, $\overline{\texttt{T}}1$ and $\overline{\texttt{T}}2$.

In [17], the authors first perform a separate spatial modeling and then a separate temporal modeling of the data to propose some candidate model components. In a final modeling step, the components are combined through a generalized additive modeling approach, including interactions between the proposed model components. Specifically, the log transformed black smoke level is modelled as a sum of fourteen functions, each containing up to three of the eleven covariates. In addition to these functions, the model includes an offset for each station site type k (rural, industrial, commercial, city/town center or mixed), α_k; a station specific random effect, b; and a time correlation model for the error term following an AR process, e. The final model is given by

$$\begin{aligned}\log(\texttt{bs}) = {} & f_1(\texttt{y}) + f_2(\texttt{doy}) + f_3(\texttt{dow}) + f_4(\texttt{y}, \texttt{doy}) + f_5(\texttt{y}, \texttt{dow}) + f_6(\texttt{doy}, \texttt{dow}) \\ & + f_7(\texttt{n}, \texttt{e}) + f_8(\texttt{n}, \texttt{e}, \texttt{y}) + f_9(\texttt{n}, \texttt{e}, \texttt{doy}) + f_{10}(\texttt{n}, \texttt{e}, \texttt{dow}) + f_{11}(\texttt{h}) \\ & + f_{12}(\texttt{T}^0, \texttt{T}^1) + f_{13}(\overline{\texttt{T}}1, \overline{\texttt{T}}2) + f_{14}(\texttt{r}) + \alpha_k + b + e.\end{aligned} \tag{5}$$

For functional forms, cubic splines were used for the temporal components, thin-plate splines for the spatial components and tensor product smoothers for the interactions. In summary, this analysis builds on a substantial and non-trivial preliminary screening and modeling of the data.

In contrast, our approach is to let PrAda-net with 100 hidden nodes automatically decide which functions to use. To make the results reasonably comparable we use a penalization parameter for PrAda-net to obtain, approximately, the same number of functions as in Equation 5. Note, however, that this generally produces slightly fewer functions for

[2] `https://www.maths.ed.ac.uk/~swood34/data/black_smoke.RData`

PrAda-net than the model in [17] since we do not post-process to combine functions that are decomposed into main and interaction terms by PrAda-net. E.g. the single function $f(e, n)$ in [17] is decomposed into two main effects, $f(e)$ and $f(n)$, and one interaction effect, $f(e, n)$, by PrAda-net (see Equation 6 and Table 3) and is thus realized by three functions. It is non-trivial to conduct such post-processing for all possible main and higher order interactions so we elect to choose a conservative model for PrAda-net.

We ran PrAda-net for 20 different random splits of the data which resulted in, in total, 52 different functions, 6 of which contained more than three covariates, see Table 3. The medoid model, where the distance between models was measured by the Jaccard distance between the set of included functions, is given by

$$\begin{aligned}\log(\texttt{bs})_{\text{PrAda}} = &\underbrace{f_1(\texttt{y})}_{10} + \underbrace{f_2(\texttt{doy})}_{4} + \underbrace{f_3(\texttt{dow})}_{2} + \underbrace{f_4(\texttt{T}^0)}_{1} + \underbrace{f_5(\texttt{T}^1)}_{2} + \underbrace{f_6(\bar{\texttt{T}}\texttt{2})}_{1} + \underbrace{f_7(\texttt{r})}_{1} + \underbrace{f_8(\texttt{e})}_{3} \\ &+ \underbrace{f_9(\texttt{n})}_{8} + \underbrace{f_{10}(\texttt{h})}_{2} + \underbrace{f_{11}(\texttt{doy}, \texttt{T}^1)}_{2} + \underbrace{f_{12}(\texttt{e}, \texttt{n})}_{12} + \underbrace{f_{13}(\texttt{e}, \texttt{h})}_{1} + \underbrace{f_{14}(\texttt{n}, \texttt{h})}_{1} \\ &+ \underbrace{f_{15}(\texttt{T}^0, \texttt{T}^1, \bar{\texttt{T}}\texttt{1})}_{1} + \underbrace{f_{16}(\texttt{e}, \texttt{n}, \texttt{h})}_{3} + \alpha_k + b + e\end{aligned} \tag{6}$$

where the number under a function denotes its complexity, measured in the number of hidden nodes used to realize it. Six of the sixteen functions in the PrAda-net medoid model overlap with the manually selected functions in [17]. Most notably, the pre-identified interaction terms between temporal covariates and between temporal and spatial covariates are not selected by PrAda-net, which, on the other hand, selects a more complex spatial interaction, $f_{16}(\texttt{e}, \texttt{n}, \texttt{h})$.

For standard lasso, the 20 data splits resulted in 143 different functions, 103 of which contained more than 3 covariates, see Table 3, with medoid function

$$\begin{aligned}\log(\texttt{bs})_{\text{Lasso}} = &\underbrace{f_1(\texttt{dow})}_{2} + \underbrace{f_2(\texttt{e})}_{1} + \underbrace{f_3(\texttt{n})}_{1} + \underbrace{f_4(\texttt{h})}_{1} + \underbrace{f_5(\texttt{doy}, \bar{\texttt{T}}\texttt{2})}_{1} + \underbrace{f_6(\texttt{e}, \texttt{n})}_{1} + \underbrace{f_7(\texttt{r}, \texttt{h})}_{1} \\ &+ \underbrace{f_8(\texttt{n}, \texttt{h})}_{1} + \underbrace{f_9(\texttt{y}, \texttt{r}, \texttt{e})}_{1} + \underbrace{f_{10}(\texttt{y}, \texttt{e}, \texttt{n})}_{1} + \underbrace{f_{11}(\texttt{y}, \texttt{doy}, \texttt{T}^1, \texttt{r})}_{1} \\ &+ \underbrace{f_{12}(\texttt{y}, \texttt{T}^1, \bar{\texttt{T}}\texttt{1}, \texttt{r})}_{1} + \underbrace{f_{13}(\texttt{y}, \texttt{e}, \texttt{n}, \texttt{h})}_{1} + \underbrace{f_{14}(\texttt{T}^0, \bar{\texttt{T}}\texttt{1}, \bar{\texttt{T}}\texttt{2}, \texttt{h})}_{1} + \underbrace{f_{15}(\texttt{T}^1, \texttt{r}, \texttt{n}, \texttt{h})}_{1} \\ &+ \underbrace{f_{16}(\texttt{T}^0, \texttt{T}^1, \texttt{e}, \texttt{n}, \texttt{h})}_{1} + \underbrace{f_{17}(\texttt{y}, \texttt{doy}, \texttt{T}^0, \texttt{T}^1, \texttt{r}, \texttt{h})}_{1} + \underbrace{f_{18}(\texttt{y}, \texttt{dow}, \bar{\texttt{T}}\texttt{1}, \texttt{r}, \texttt{n}, \texttt{h})}_{1} \\ &+ \alpha_k + b + e.\end{aligned} \tag{7}$$

Compared to standard lasso, PrAda-net has higher precision, i.e. it detects fewer unique functions more frequently, indicating a more stable selection performance. This is confirmed by the medoid model being more similar to the other models for PrAda-net than for standard lasso. The average Jaccard similarity of the PrAda-net medoid model is 0.63, compared with 0.18 for standard lasso. PrAda-net also identifies functions containing fewer covariates (order of interaction) thus resulting in more interpretable models compared to standard lasso. All the $2 \cdot 20$ models had an explained variance of $R^2 \approx 0.79$, which is the same as reported by [17].

3.3 Synthetic Black Smoke Data

To test PrAda-net on a more complex model than the synthetic data in Section 3.1, we generated data from the PrAda-net medoid model for the black smoke data. 10000 data points were sampled, according to $\mathcal{N}(0, \Sigma_S)$, where Σ_S is the Spearman rank correlation matrix of the variables in the black smoke data set, resulting in a 10000×11 x-matrix, from which a 10000×1 y-matrix was created according to Equation 6, without α_k, b, and e and with added noise distributed as $\mathcal{N}(0, 0.1^2)$. 50 simulations, with different noise realizations, were performed using a hidden layer with 100 units. For PrAda-net, λ^* was chosen as described in Section 2.3 and for standard lasso, the regularization strength was chosen to obtain the same number of functions as for PrAda-net. The results are summarized in Table 4, where all functions present in more than 20 % of the simulations are shown and functions present in more than 50 % of the simulations are marked bold.

PrAda-net identifies 14 of the 16 true model components together with 4 false positives, the corresponding numbers for standard lasso are 6 and 17, respectively. There is a tendency that more complex functions, i.e. functions consisting of more than one hidden node, are more easily identified. Except for $f_5(\texttt{T}^1)$ and $f_{10}(\texttt{h})$, PrAda-net identifies all the functions realized by more than one hidden node in 100 % of the simulations; for both algorithms all the falsely included

Table 3: Identified functions in 20 bootstrap runs using PrAda-net and standard lasso. Only the functions present in more than half of the runs are shown.

	PrAda-net		Standard lasso	
Identified functions in total	52		143	
Functions with more than 3 covariates	6		103	
Most frequent functions with frequencies	$f(\texttt{doy})$	1.0	$f(\texttt{e},\texttt{n})$	0.8
	$f(\texttt{y})$	1.0	$f(\texttt{dow})$	0.8
	$f(\texttt{T}^1)$	1.0	$f(\texttt{n})$	0.7
	$f(\texttt{n})$	1.0	$f(\texttt{n},\texttt{h})$	0.6
	$f(\texttt{e})$	1.0	$f(\texttt{h})$	0.55
	$f(\texttt{e},\texttt{n},\texttt{h})$	1.0	$f(\texttt{e})$	0.55
	$f(\texttt{e},\texttt{n})$	1.0		
	$f(\texttt{dow})$	0.95		
	$f(\texttt{h})$	0.9		
	$f(\texttt{n},\texttt{h})$	0.85		
	$f(\texttt{e},\texttt{h})$	0.8		
	$f(\texttt{T}^0)$	0.75		
	$f(\texttt{doy},\texttt{T}^1)$	0.6		
	$f(\bar{\texttt{T}}\texttt{2})$	0.55		

Table 4: Presence and average complexity of functions across 50 simulation runs for PrAda-net (left) and standard lasso (right), with true model components above the line and falsely included components below. Only functions present in more than 20 % of the runs are presented, functions present in more than 50 % of the runs are bold.

PrAda-net		
Function	Presence	Complexity
$\boldsymbol{f(\texttt{y})}$	1.0	5.06
$\boldsymbol{f(\texttt{doy})}$	1.0	20.92
$\boldsymbol{f(\texttt{dow})}$	1.0	2.9
$\boldsymbol{f(\texttt{e})}$	1.0	2.36
$\boldsymbol{f(\texttt{n})}$	1.0	3.0
$\boldsymbol{f(\texttt{doy},\texttt{T}^1)}$	1.0	2.22
$\boldsymbol{f(\texttt{e},\texttt{n})}$	1.0	13.44
$\boldsymbol{f(\texttt{n},\texttt{h})}$	1.0	1.18
$\boldsymbol{f(\texttt{e},\texttt{n},\texttt{h})}$	1.0	2.22
$\boldsymbol{f(\texttt{T}^1)}$	0.98	1.27
$\boldsymbol{f(\texttt{T}^0)}$	0.94	1.23
$\boldsymbol{f(\texttt{e},\texttt{h})}$	0.72	1.44
$f(\texttt{h})$	0.48	1.04
$f(\texttt{r})$	0.22	1.0
$f(\texttt{doy},\bar{\texttt{T}}\texttt{1})$	0.4	1.05
$f(\texttt{doy},\texttt{e})$	0.3	1.07
$f(\texttt{dow},\texttt{e})$	0.24	1.17
$f(\texttt{T}^1,\bar{\texttt{T}}\texttt{1})$	0.2	1.0

Standard lasso		
Function	Presence	Complexity
$\boldsymbol{f(\texttt{y})}$	1.0	15.8
$\boldsymbol{f(\texttt{e},\texttt{n})}$	1.0	4.16
$\boldsymbol{f(\texttt{e},\texttt{n},\texttt{h})}$	0.96	1.27
$\boldsymbol{f(\texttt{doy})}$	0.82	1.76
$\boldsymbol{f(\texttt{n})}$	0.82	1.37
$\boldsymbol{f(\texttt{h})}$	0.56	1.0
$\boldsymbol{f(\texttt{dow},\texttt{e},\texttt{h})}$	0.7	1.2
$\boldsymbol{f(\texttt{T}^0,\texttt{e},\texttt{n},\texttt{h})}$	0.58	1.03
$\boldsymbol{f(\texttt{doy},\texttt{T}^1,\bar{\texttt{T}}\texttt{2},\texttt{e},\texttt{h})}$	0.52	1.27
$\boldsymbol{f(\texttt{doy},\texttt{T}^0,\texttt{T}^1,\bar{\texttt{T}}\texttt{2},\texttt{r},\texttt{e},\texttt{n},\texttt{h})}$	0.52	1.0
$f(\texttt{dow},\texttt{e})$	0.44	1.05
$f(\texttt{doy},\texttt{T}^0,\texttt{T}^1,\bar{\texttt{T}}\texttt{1},\bar{\texttt{T}}\texttt{2},\texttt{r},\texttt{e},\texttt{n},\texttt{h})$	0.34	1.18
$f(\texttt{doy},\texttt{T}^0,\texttt{T}^1,\texttt{e},\texttt{h})$	0.32	1.0
$f(\texttt{doy},\texttt{dow},\texttt{e},\texttt{h})$	0.3	1.07
$f(\texttt{doy},\texttt{T}^1,\bar{\texttt{T}}\texttt{2},\texttt{e})$	0.3	1.0
$f(\texttt{doy},\bar{\texttt{T}}\texttt{1},\texttt{e},\texttt{h})$	0.3	1.0
$f(\texttt{doy},\texttt{T}^0,\texttt{T}^1,\bar{\texttt{T}}\texttt{1},\bar{\texttt{T}}\texttt{2},\texttt{h})$	0.3	1.0
$f(\texttt{doy},\texttt{T}^0,\texttt{T}^1,\bar{\texttt{T}}\texttt{2},\texttt{e},\texttt{h})$	0.3	1.2
$f(\texttt{T}^0,\texttt{T}^1,\bar{\texttt{T}}\texttt{1},\bar{\texttt{T}}\texttt{2},\texttt{r},\texttt{e},\texttt{n},\texttt{h})$	0.26	1.0
$f(\texttt{T}^1,\texttt{n})$	0.24	1.0
$f(\texttt{T}^0,\texttt{e},\texttt{n})$	0.24	1.0
$f(\texttt{doy},\texttt{T}^0,\texttt{T}^1,\bar{\texttt{T}}\texttt{1},\bar{\texttt{T}}\texttt{2},\texttt{r},\texttt{e},\texttt{h})$	0.24	1.0
$f(\texttt{doy},\texttt{T}^1,\texttt{e},\texttt{h})$	0.22	1.0

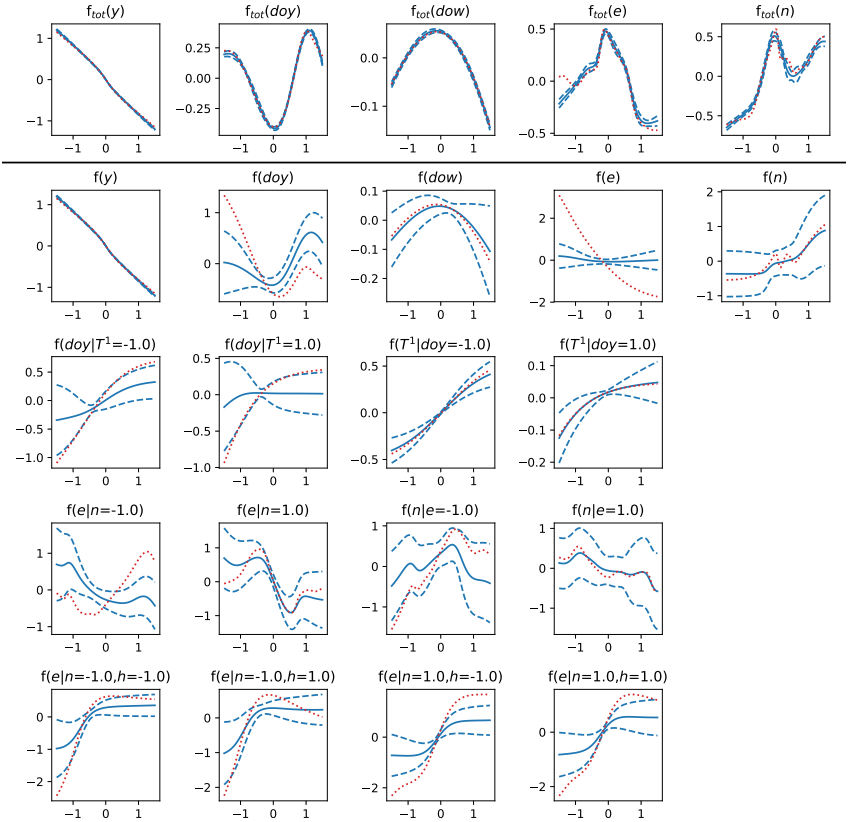


Figure 3: PrAda-net: Comparison of true (red and dotted) and the mean $\pm$ two standard deviations of the identified functions (blue). For the multi-dimensional functions, in each plot all but one of the input variables are kept fixed. The first row shows the total dependency of the covariate, including all functions, while the rest of the rows show the most frequently identified functions.

functions also have a complexity close to one node. Apart from $f(\texttt{doy})$, PrAda-net seems to be closer to the true complexity than standard lasso.

In Figures 3 and 4, the means and standard deviations for the most frequently identified functions are plotted together with the true functions. Since the intercept can be realized in any function, all functions were translated to have zero mean. For functions depending on more than one covariate, all covariates except one were fixed at ± 1 to be able to plot in one dimension. For PrAda-net we see that for total the contribution of a covariate, the true and identified functions coincide very well (first row). This is also true for most identified functions with a few exceptions. However, the differences between true and identified functions seem to be balanced by those of other functions including the same covariate. See e.g. $f(\texttt{doy})$ and $f(\texttt{doy}|\texttt{T}^1)$, where, for small values of doy, the true $f(\texttt{doy})$ is much larger than the estimated function, while the true $f(\texttt{doy}|\texttt{T}^1)$ is much smaller than the estimated function.

Standard lasso exhibits poor overlap with the true model components, even in terms of total contribution. The functions identified by standard lasso align poorly with the true model components.

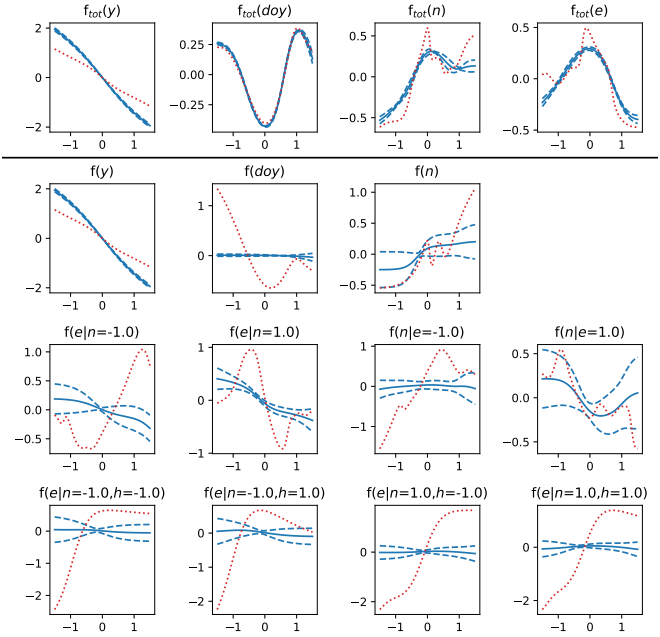


Figure 4: Standard lasso: Comparison of true (red and dotted) and the mean $\pm$ two standard deviations of the identified functions (blue). For the multi-dimensional functions, in each plot all but one of the input variables are kept fixed. The first row shows the total dependency of the covariate, including all functions, while the rest of the rows show the most frequently identified functions.

4 Conclusions and Discussion

We proposed a simple neural network prediction model based on proximal gradient descent and adaptive lasso, PrAda-net. PrAda-net was found to improve on lasso penalized neural networks both in terms of test error performance and in terms of generating interpretable models. For additive models, PrAda-net was able to identify the function components of the models as well as to express the function complexity by using multiple hidden nodes. PrAda-net was also found to be superior to lasso and DGR in terms of capturing variable importance. We illustrated that PrAda-net can be used as an alternative to generalized additive models, to analyze complex data sets with temporal and spatial covariates, reducing the need for manually intensive preprocessing and screening of massive data.

Even if there is no limitation to the depth of the network PrAda-net can be applied to, we only used one hidden layer in this paper, thereby loosing the possibility of making the network even more compact. However, according to the universal approximation theorem ([22]) this does not limit the complexity of the functions that can be modelled. It would, however, be interesting to apply PrAda-net to deeper networks with regularization penalties constructed to guarantee an ordering of main to higher order effects in the layers, where structural constraints may be needed to generate interpretable networks from more complex architectures.

References

[1] Trevor J Hastie and Robert J Tibshirani. *Generalized additive models*, volume 43. CRC press, 1990.

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

[3] 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.

[4] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429, 2006.

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

[6] Jian Wang, Chen Xu, Xifeng Yang, and Jacek M Zurada. A novel pruning algorithm for smoothing feedforward neural networks based on group lasso method. *IEEE transactions on neural networks and learning systems*, 29(5):2012–2024, 2018.

[7] Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Mattan Erez, and Sujay Shanghavi. Prunetrain: Gradual structured pruning from scratch for faster neural network training. *arXiv preprint arXiv:1901.09290*, 2019.

[8] Huaqing Zhang, Jian Wang, Zhanquan Sun, Jacek M Zurada, and Nikhil R Pal. Feature selection for neural networks using group lasso regularization. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[9] Alex Tank, Ian Cover, Nicholas J Foti, Ali Shojaie, and Emily B Fox. An interpretable and sparse neural network model for nonlinear granger causality discovery. *arXiv preprint arXiv:1711.08160*, 2017.

[10] 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.

[11] Guillaume Leclerc, Manasi Vartak, Raul Castro Fernandez, Tim Kraska, and Samuel Madden. Smallify: Learning network size while training. *arXiv preprint arXiv:1806.03723*, 2018.

[12] Kai Sun, Shao-Hsuan Huang, David Shan-Hill Wong, and Shi-Shang Jang. Design and application of a variable selection method for multilayer perceptron neural network with lasso. *IEEE transactions on neural networks and learning systems*, 28(6):1386–1396, 2017.

[13] Yulong Wang, Hang Su, Bo Zhang, and Xiaolin Hu. Interpret neural networks by identifying critical data routing paths. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8906–8914, 2018.

[14] Joel Vaughan, Agus Sudjianto, Erind Brahimi, Jie Chen, and Vijayan N Nair. Explainable neural networks based on additive index models. *arXiv preprint arXiv:1806.01933*, 2018.

[15] Meiling Xu and Min Han. Adaptive elastic echo state network for multivariate time series prediction. *IEEE transactions on cybernetics*, 46(10):2173–2183, 2016.

[16] Junfei Qiao, Lei Wang, and Cuili Yang. Adaptive lasso echo state network based on modified bayesian information criterion for nonlinear system modeling. *Neural Computing and Applications*, pages 1–15, 2018.

[17] Simon N Wood, Zheyuan Li, Gavin Shaddick, and Nicole H Augustin. Generalized additive models for gigadata: modeling the uk black smoke network daily data. *Journal of the American Statistical Association*, 112(519):1199–1210, 2017.

[18] R Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

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

[20] Trevor Hastie and Junyang Qian. Glmnet vignette. *Retrieve from http://www. web. stanford. edu/˜ hastie/Papers/Glmnet_Vignette. pdf. Accessed September*, 20:2016, 2014.

[21] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[22] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

PAPER II

II

NON-LINEAR, SPARSE DIMENSIONALITY REDUCTION VIA PATH LASSO PENALIZED AUTOENCODERS

Oskar Allerbo
Mathematical Sciences
University of Gothenburg and Chalmers University of Technology
allerbo@chalmers.se

Rebecka Jörnsten *
Mathematical Sciences
University of Gothenburg and Chalmers University of Technology
jornsten@chalmers.se

ABSTRACT

Dimensionality reduction is often used to explore and analyze complex and high-dimensional data. The purpose of the dimensionality reduction can be to create a latent low-dimensional space for visualization, to pre-process data for predictive modeling, or for clustering. For complex data, linear dimensionality reduction techniques like PCA may not be able to summarize the data with a low-dimensional latent space. Non-linear dimension techniques, like kernel-PCA and autoencoders, while sufficiently flexible to capture complex structure, suffer from loss of interpretability, since each latent variable is dependent of all input dimensions. Here we present path lasso which we apply to an autoencoder for a structured regularization in order to enhance interpretability. Specifically, we penalize each *path* through the encoder from an input to a latent variable, thus restricting how many input variables are represented in each latent dimension. The path lasso 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.

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 ([1]). Since the introduction of sparse PCA, several variants have been presented, such as non-negative sparse PCA ([2]), where the loadings are all non-negative; multilinear sparse PCA ([3]), that operates on tensors instead of vectors; and robust PCA ([4], [5]), 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 ([6]) and autoencoders ([7]). 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

*corresponding author

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 [8], [9], [10] and [11], as well as for sparse autoencoders (e.g. [12]), 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 n being the number of observations. For $Z \in \mathbb{R}^{n \times d_z}$, $X \in \mathbb{R}^{n \times d_x}$ and $W \in \mathbb{R}^{d_x \times d_z}$, the latent representation Z is obtained from the original representation X as

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

Then sparsity in the sense original to latent dimensions means that W is sparse, while sparsity in the sense observations to latent dimensions means that 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 W being zero, allowing the remaining rows to be dense.

The sparse autoencoder generalizes Equation 1 to $Z = \text{Enc}(X)$, where Enc is the non-linear encoder and, to produce a sparse 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 W, which requires a sparse architecture.

Neural networks with sparse architectures are usually obtained using different versions of lasso, or l_1-, regularization ([13]), and there are numerous examples of these. In [14], group lasso ([15]) is used to remove all the links to or from a node, while in [16] group lasso is combined with exclusive lasso ([17]) 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 [18], with a linear encoder and a lasso regularized decoder; [19], with a lasso regularized encoder and a linear decoder; and [20], 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 a 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 do 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 describes different ways to accelerate training and Section 2.6 describes how we use path lasso in an autoencoder.

2.1 Review of Lasso Penalties

The lasso algorithm, when applied, sets some of the model parameters equal to zero, 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'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 $\gamma > 0$, 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

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

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 $\boldsymbol{\theta} \in \mathbb{R}^d$, $\boldsymbol{\theta}_{\boldsymbol{g}}$ 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

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

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 ([21]) 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 a step size 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 component-wise and is, with $(x)^+ := \max(x, 0)$,

$$\mathrm{prox}_{\alpha h}(\theta_i) = \mathrm{sign}(\theta_i) \cdot (|\theta_i| - \alpha\lambda)^+ = \theta_i \cdot \left(1 - \frac{\alpha\lambda}{|\theta_i|}\right)^+.$$

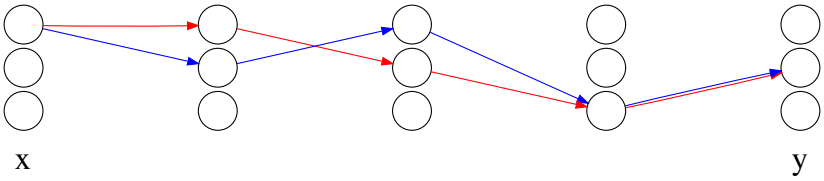


Figure 1: Illustration of two of the 3^3 (where the exponent is the number of inner layers and the base is their width) possible paths between node one in layer one and node two in layer five. Each arrow denotes a link, which corresponds to an element in a weight matrix.

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

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

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

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. One can note that if each group consists of only one θ, the proximal operator for group lasso coincides with the proximal operator for standard lasso.

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

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

where $\{W_l\}_{l=1}^L$, $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$, 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 2 as a graph, each weight matrix element, $w_{i_1 i_2}^l := (W_l)_{i_1 i_2}$, 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 in [22], we define the value of a path as the product of its absolute valued links,

$$p_{i_L i_0}^k := |w_{i_L i_{L-1}^k}^L| \cdot |w_{i_{L-1}^k i_{L-2}^k}^{L-1}| \cdot \ldots |w_{i_1^k i_0}^1|, \qquad k = 1, 2, \ldots, \prod_{l=1}^{L-1} d_k,$$

where each k corresponds to a unique combination of the indices i_1 to i_{L-1}, which is indicated by the superscript k. 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.

In order to to use the group lasso to set all paths between two nodes to zero simultaneously, we define the matrix

$$W_{\text{GL}} := \sqrt{\prod_{l=1}^{L} (W_l)^2} = \sqrt{(W_L)^2 \cdot (W_{L-1})^2 \cdot \ldots (W_1)^2}, \tag{3}$$

where the square and the square root are taken element-wise. The following Lemma describes the connection between this matrix and the groups of all paths between two nodes in non-adjacent layers.

Lemma 1. *The element (i_L, i_0) in W_{GL} is the group lasso penalty on the group consisting of all paths between nodes x_{i_0} and y_{i_L}, i.e.*

$$(W_{GL})_{i_L i_0} = \sqrt{\sum_k (p^k_{i_L i_0})^2}$$

For a proof, see Appendix A.

Applying Lemma 1 to the proximal step for the k-th path connecting x_{i_0} and y_{i_L}, we get

$$p^k_{i_L i_0} \cdot \left(1 - \frac{\alpha\lambda}{\sqrt{\sum_k (p^k_{i_L i_0})^2}}\right)^+ = p^k_{i_L i_0} \cdot \left(1 - \frac{\alpha\lambda}{(W_{\text{GL}})_{i_L i_0}}\right)^+. \tag{4}$$

We call each element in W_{GL} a connection, i.e. $(W_{\text{GL}})_{i_L i_0}$ 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. Theorem 1 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 $\boldsymbol{x}$.

Theorem 1. *Let the vector $\boldsymbol{y}$ depend on the vector $\boldsymbol{x}$ as stated in Equation 2 and let W_{GL} be the path group lasso matrix defined in Equation 3. Then, if all weights and activations are bounded,*

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

For a proof, see Appendix A.

2.4 From Paths to Links

Equation 4 describes how all paths between two nodes in the networks are penalized towards zero, but it does not tell how this penalization affects the individual links. The following theorem describes how the penalized paths can be translated to penalized links.

Theorem 2. *The group lasso penalties on the paths can be transformed to standard lasso penalties on the links by first solving the matrix equation*

$$\prod_{l=1}^{L} |W_l| \odot \left(1 - \frac{\alpha\lambda}{W_{GL}}\right)^+ = \prod_{l=1}^{L} \tilde{W}_l, \tag{5}$$

with respect to $\tilde{W}_l$, and then set $(W_l)^{t+1} := \text{sign}((W_l)^t) \odot \tilde{W}_l$.
Then $(w^l_{ij})^{t+1} = \text{sign}((w^l_{ij})^t) \cdot \tilde{w}^l_{ij} =: \text{sign}((w^l_{ij})^t) \cdot (|(w^l_{ij})^t| - \alpha\lambda^l_{ij})^+$ for some $\lambda^l_{ij} > 0$, which is the proximal operator for standard lasso.
The absolute value, sign and division are taken element-wise and $\odot$ denotes element-wise multiplication.

The proof, which is presented in Appendix A, 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 $\tilde{w}^l_{i_l i_{l-1}} \leq |w^l_{i_l i_{l-1}}|$, or equivalently $\lambda^l_{i_l i_{l-1}} \geq 0$. We describe this algorithm in Appendix B.

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 $|W_l|$ as the seed for each $\tilde{W}_l$ in the matrix factorization.

An optimization step using proximal gradient descent on paths is summarized in Algorithm 1.

2.5 Accelerating the Algorithm

The matrix factorization step in Equation 5 is the bottle neck of Algorithm 1 and to increase its speed, three approaches are used: Substitution, parallelization and Boolean matrix factorization.

Algorithm 1 Proximal Path Lasso Step

1: Update all weights and biases using gradient descent:
$\theta^{t+\frac{1}{2}} \leftarrow \theta^t - \alpha \nabla f(\theta^t)$, for all elements θ in $\{W_l\}_{l=l}^{L}$ and $\{\boldsymbol{b_l}\}_{l=l}^{L}$.
2: Construct the group lasso penalty matrix:
$W_{\text{GL}} \leftarrow \sqrt{\prod_{l=1}^{L}(W_l^{t+\frac{1}{2}})^2}$
3: Penalize paths and translate the penalty to penalized links by solving Equation 5, using modified non-negative matrix factorization:
$\left\{\tilde{W}_l^{t+1}\right\}_{l=1}^{L} \leftarrow \text{NMF}\left(\prod_{l=1}^{L}|W_l^{t+\frac{1}{2}}| \odot \left(1 - \frac{\alpha\lambda}{W_{\text{GL}}}\right)^+\right)$
4: Restore signs:
$W_l^{t+1} \leftarrow \text{sign}(\tilde{W}_l^t) \odot \tilde{W}_l^{t+1}$

2.5.1 Substitution

By first applying an off-the-shelf optimization method to $f(\boldsymbol{\theta}) = g(\boldsymbol{\theta}) + \lambda W_{\text{GL}}$, 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. The penalties depend on the magnitude of the connection after the first stage:

$$\lambda_{i_L,i_0} := \frac{\lambda}{((\hat{W}_{\text{GL}})_{i_L i_0})^\gamma} \tag{6}$$

where $\hat{W}_{\text{GL}}$ is the value of W_{GL} after the first optimization stage and $\gamma > 0$. Throughout this paper, $\gamma = 2$ 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.

2.5.2 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 7 below. Let $A \in (\mathbb{R}_{\geq 0})^{d_3 \times d_2}$ and $B \in (\mathbb{R}_{\geq 0})^{d_2 \times d_1}$ denote the absolute valued weight matrices and let $C + D \in (\mathbb{R}_{\geq 0})^{d_3 \times d_1}$ denote the penalized path matrix, where the sum of the paths in $(C + D)_{ij}$ is split into C_{ij}, containing the sum of some of the paths, and D_{ij} containing the sum of the remaining paths. Which paths belong to C and which belong to D depends on how A and B are split. Then, the factorization of $C + D$ into $A \cdot B$ can be divided into $2^3 = 8$ smaller factorizations which can be solved in parallel, i.e.

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

can be split into

$$\begin{aligned} \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right] &= \left[\begin{array}{c|c} A_{11} & 0 \\ \hline A_{21} & 0 \end{array}\right] \cdot \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline 0 & 0 \end{array}\right] = \left[\begin{array}{c|c} A_{11} \cdot B_{11} & A_{11} \cdot B_{12} \\ \hline A_{21} \cdot B_{11} & A_{21} \cdot B_{12} \end{array}\right] \\ \left[\begin{array}{c|c} D_{11} & D_{12} \\ \hline D_{21} & D_{22} \end{array}\right] &= \left[\begin{array}{c|c} 0 & A_{12} \\ \hline 0 & A_{22} \end{array}\right] \cdot \left[\begin{array}{c|c} 0 & 0 \\ \hline B_{21} & B_{22} \end{array}\right] = \left[\begin{array}{c|c} A_{12} \cdot B_{21} & A_{12} \cdot B_{22} \\ \hline A_{22} \cdot B_{21} & A_{22} \cdot B_{22} \end{array}\right] \end{aligned} \tag{7}$$

or equivalently

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

Here, $A_{ij} \in (\mathbb{R}_{\geq 0})^{d_{3i} \times d_{2j}}$, $B_{ij} \in (\mathbb{R}_{\geq 0})^{d_{2i} \times d_{1j}}$ and $C_{ij}, D_{ij} \in (\mathbb{R}_{\geq 0})^{d_{3i} \times d_{1j}}$, 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 A_{ij} and 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:

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

where A_{ij}^n is the n-th solution of 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.

2.5.3 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 5 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 5 as:

$$B_{ij} := \mathbb{I}\left[\left(\prod_{l=1}^{L} |W_l| \odot \left(1 - \frac{\alpha\lambda}{W_{\text{GL}}}\right)^{+}\right)_{ij} > 0\right]$$

$$(B_l)_{ij}(\tau) := \mathbb{I}\left[(\tilde{W}_l)_{ij} > \tau\right],\ \tau \geq 0$$

where $\mathbb{I}[\cdot]$ is the indicator function which equals 1 when its argument is true and 0 otherwise. Thus, in the matrix on the left hand side in Equation 5, 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 $B = \bigvee_{l=1}^{L} B_l$, where $\bigvee$ denotes Boolean matrix multiplication, then the matrix factorization was successful in the sense that all connections in W_{GL} are represented by appropriate links in the weight matrices. Each differing element means that either two nodes are disconnected in W_{GL} but not in $\{\tilde{W}_l\}_{l=1}^{L}$ or vice versa. To minimize this quantity, we minimize $\sum |B - \bigvee_{l=1}^{L} B_l(\tau)|$, with the absolute value taken element-wise, with respect to τ, and then threshold $\{\tilde{W}_l\}_{l=1}^{L}$ using the resulting τ:

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

$$\tilde{W}_l \leftarrow \tilde{W}_l \odot \mathbb{I}[\tilde{W}_l > \tau]$$

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.

2.6 Path Lasso for Dimensionality Reduction

For dimensionality reduction, path lasso was used together with an autoencoder with one hidden layer, with tanh activation, in the encoder and decoder respectively, trained with l_2-loss. The path penalties were applied between the input and latent variables, and between the latent and output variables. For a given observation, the original input is denoted by $\boldsymbol{x} \in \mathbb{R}^{d_x}$, the reconstructed output by $\hat{\boldsymbol{x}} \in \mathbb{R}^{d_x}$ and the latent representation by $\boldsymbol{z} \in \mathbb{R}^{d_z}$, where $d_z < d_x$.

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 and all paths connecting z_j and $\hat{x}_i$, i.e. $W_{\text{GL}} := \sqrt{(W_2)^2 \cdot (W_1)^2 + ((W_4)^2 \cdot (W_3)^2)^\top}$, where (W_1, W_2) and (W_3, W_4) are the weight matrices of the encoder and the decoder. This means that if an input is disconnected to a latent variable, so is the corresponding output.

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, similarly to in adaptive lasso, we first trained the network without path regularization to obtain individual penalties for each

connection during the path lasso stage. To encourage the algorithm to make equal use of the latent dimensions, we added an exclusive lasso penalty to the elements in W_{GL}, with as many groups as there are latent dimensions, each group being defined as the connections to a given latent dimension. This was trained without proximal methods analogously to in 2.5.1.

In a second stage, we added path regularization with an individual penalty for each connection, where the penalties depended on the magnitude of the connection just after the first training stage, again using Equation 6, to penalize unimportant connections harder than important ones. Then in a third stage we applied proximal gradient descent to obtain exact zeros. Finally, to improve performance, we again used an of-the-shelf optimizer, but now with the links set to zero in the previous stage kept to zero. This is summarized in Algorithm 2.

Algorithm 2 Path Lasso for Dimensionality Reduction

1: Train the neural network with optimizer of choice and an exclusive lasso penalty over the latent dimensions.
2: Add an individual path lasso penalty to each connection. Continue training with the same optimizer.
3: Update the individual penalties. Continue training with proximal gradient descent, Algorithm 1.
4: Continue training with optimizer from steps 1 and 2, without regularization, but with zeros kept fixed.

3 Experiments

In order to evaluate path lasso for dimensionality reduction we applied it to three different datasets, one synthetic 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.

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. It should however 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 these three 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. $d_{l_2}(\hat{x}_i, x_i) < d_{l_2}(\hat{x}_i, x_j)$, $i \neq j$. This is hereafter referred to as 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 $X_i \sim \mathcal{N}(\mu_i, 0.01 \cdot I_4)$, where I_4 is the identity matrix and μ_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 $\mathcal{N}(0, 0.3^2)$ distributed noise added.

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 was chosen. The results are presented in Table 1, where the p-values come from the one-sided paired t-test, that tests whether the value of the algorithm is better than that of path lasso. 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 reconstruction match, while all the four non-linear methods perform very well in terms of R^2.

The results with the best R^2 values are plotted in Figure 2, where clusters that are diagonal to each other (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

Table 1: Number of remaining connections, explained variance, reconstruction match and p-value for six different algorithms when reducing 16 clusters from four to two dimensions with (bottom) and without (top) added noise. P-values are for the one-sided paired t-test, that tests whether the value of the algorithm is better than that of path lasso, 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.

Algorithm	Connections	R^2		Reconstruction Match	
		Value	p-value	Value	p-value
Path Lasso	4	0.98 ± 0.0013	-	0.37 ± 0.015	-
Standard Lasso	4	0.98 ± 0.002	0.17	0.36 ± 0.022	0.069
Sparse PCA	4	0.48 ± 0.0018	$\mathbf{7 \cdot 10^{-23}}$	0.067 ± 0.019	$\mathbf{1.9 \cdot 10^{-11}}$
Autoencoder	8	0.98 ± 0.0026	0.034	0.35 ± 0.013	$\mathbf{0.00092}$
Sparse Autoencoder	8	0.98 ± 0.00027	0.58	0.34 ± 0.0076	$\mathbf{0.00011}$
PCA	8	0.48 ± 0.0014	$\mathbf{2.2 \cdot 10^{-23}}$	0.067 ± 0.015	$\mathbf{1.2 \cdot 10^{-12}}$
Path Lasso	4	0.89 ± 0.017	-	0.38 ± 0.038	-
Standard Lasso	4	0.58 ± 0.097	$\mathbf{3.7 \cdot 10^{-7}}$	0.11 ± 0.03	$\mathbf{7.6 \cdot 10^{-9}}$
Sparse PCA	4	0.50 ± 0.0016	$\mathbf{6.1 \cdot 10^{-14}}$	0.11 ± 0.0061	$\mathbf{1.4 \cdot 10^{-9}}$
Autoencoder	8	0.86 ± 0.011	$\mathbf{0.00065}$	0.33 ± 0.014	$\mathbf{0.0029}$
Sparse Autoencoder	8	0.85 ± 0.0038	$\mathbf{9.2 \cdot 10^{-5}}$	0.30 ± 0.015	$\mathbf{6.2 \cdot 10^{-5}}$
PCA	8	0.50 ± 0.0043	$\mathbf{8.3 \cdot 10^{-14}}$	0.12 ± 0.0088	$\mathbf{1.1 \cdot 10^{-9}}$

of the original four dimensions, which can be seen in the axis aligned data in the plots. The linear algorithms, PCA and sparse PCA, are not able to fully separate the sixteen clusters, but the four non-linear algorithms, based an autoencoders, are.

3.2 Text - 20 Newsgroup Dataset

To test the algorithm on text data, the 20 newsgroups data set[2], 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 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. In two experiments, the 100 dimensional data set was mapped down to four and two dimensions using path lasso, standard lasso and sparse PCA. For the two autoencoder based algorithms, the layer widths were 100, 50, 4 (2), 50 and 100 nodes. In each of the three cases the penalties were set to obtain (approximately) the same sparsity.

The latent spaces 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 can probably be attributed to the flexibility of the autoencoder, and without the structure in the penalty imposed by path lasso, there is less incentive to fully utilize 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 W_{GL} 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 $W_2 \cdot W_1 + (W_4 \cdot W_3)^\top$. The signed words in the latent dimensions are presented in Table 2. The assignment of words to the 8 (4) 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 Table 3, remaining connections, explained variance and reconstruction match is presented. We conclude that path lasso performs best both in terms of explained variance and reconstruction match.

[2]Available at `http://qwone.com/~jason/20Newsgroups/`

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

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

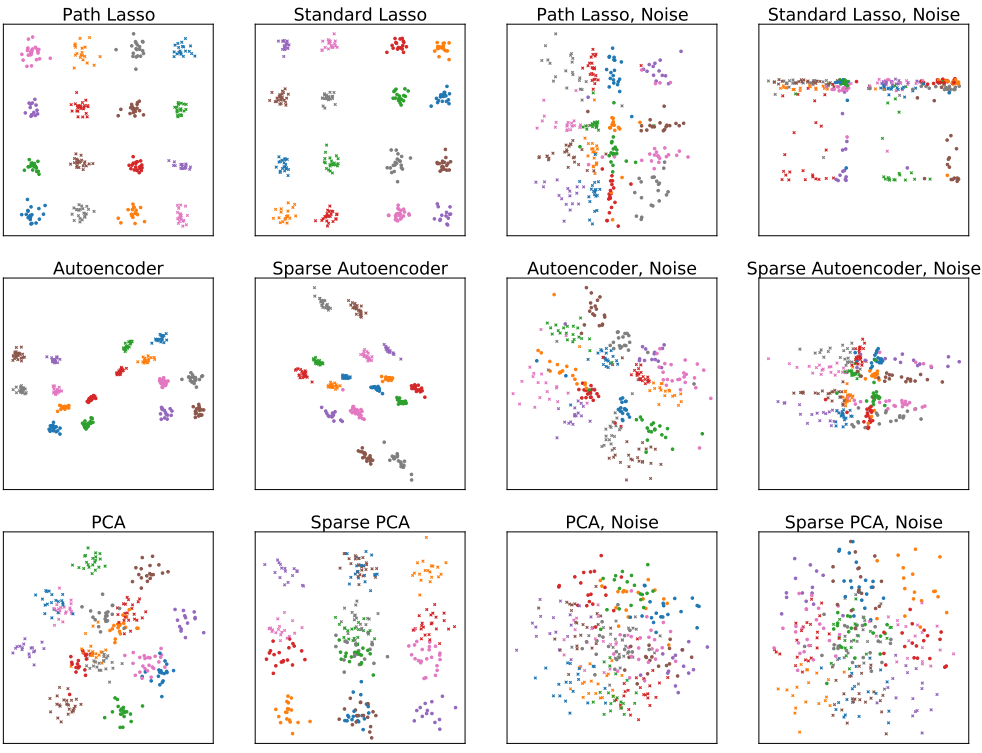
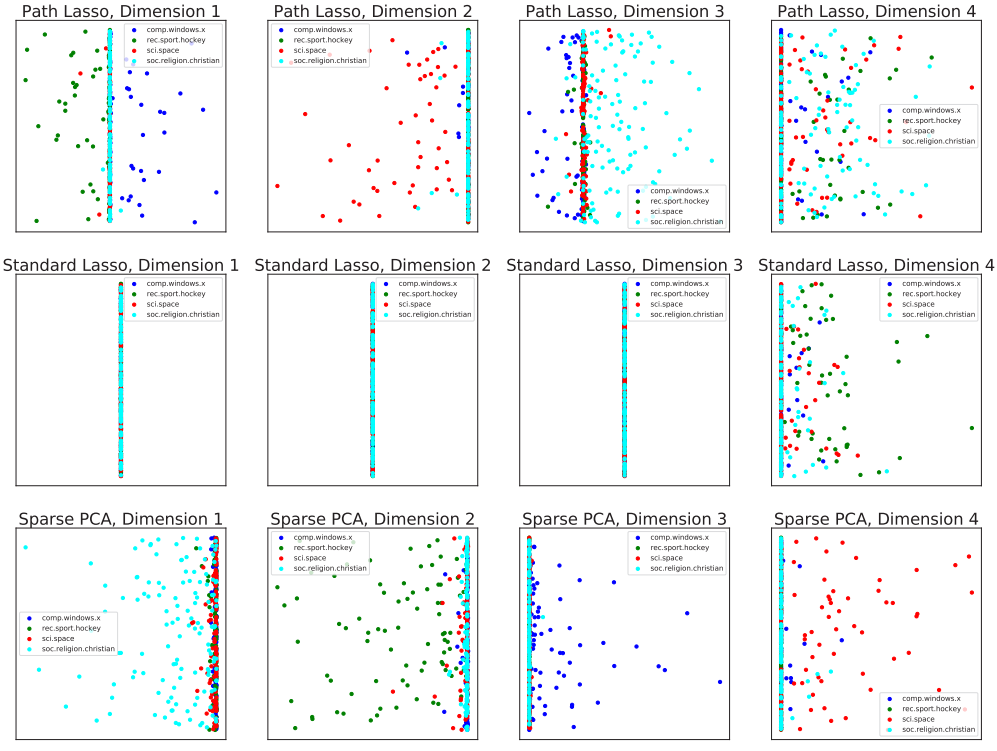


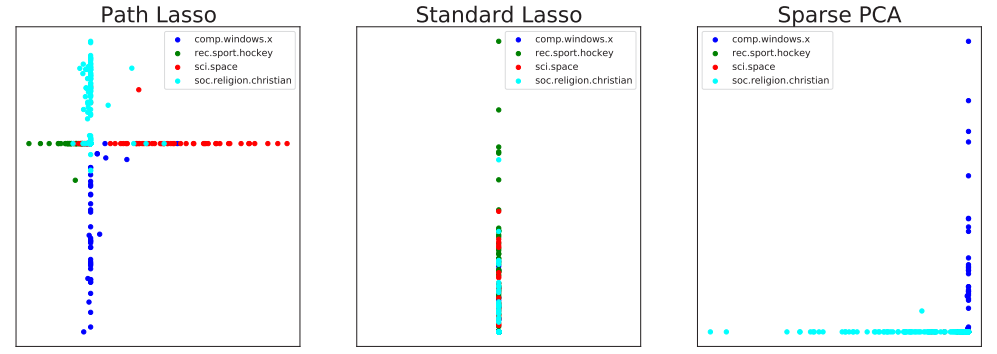
Figure 2: Reducing 16 clusters from four to two dimension with and without noise, using six different algorithms. Clusters that are diagonal to each other have the same color, but different markers.

Table 3: Number of remaining connections, explained variance and reconstruction match for the three algorithms on the newsgroup data.

Dimensions	Algorithm	Connections	R^2	Reconstruction Match
4	Path Lasso	49	0.2	0.029
4	Standard Lasso	51	0.015	0.0021
4	Sparse PCA	49	0.11	0.017
2	Path Lasso	9	0.13	0.019
2	Standard Lasso	11	0.015	0.0021
2	Sparse PCA	10	0.028	0.0084



(a) Latent 4D space for the newsgroup data



(b) Latent 2D space for the newsgroup data

Figure 3: Latent spaces of the news data, when compressed down to 4 and to 2 latent dimensions. In 3a the y-axis value is just a random number, added to increase readability.

Table 4: Number of remaining connections, explained variance and reconstruction match for the four algorithms on the image data.

Algorithm	Connections	R^2	Reconstruction Match
Autoencoder	15000	0.73	0.66
Path Lasso	14923	0.72	0.7
Standard Lasso	14936	0.7	0.55
Sparse PCA	14923	-0.24	0.16
Path Lasso	3770	0.57	0.21
Standard Lasso	3771	0.44	0.025
Sparse PCA	3770	-1.2	0.1

3.3 Images - AT&T Face Database

We also tested the algorithm on the AT&T face database ([23]), 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.

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 4. 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 reconstruction match, path lasso outperforms the other two algorithms, and at low sparsity even does better than the fully connected autoencoder.

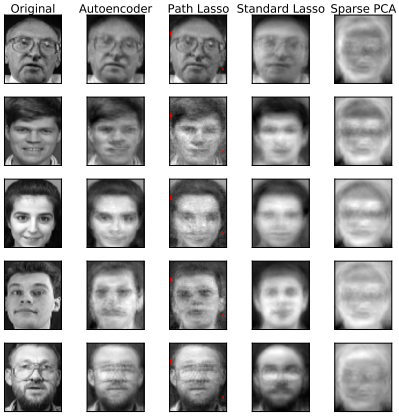
The five eigenfaces - calculated as $\text{Dec}(e_i)$, where Dec is the decoder and e_i is the i-th standard basis vector in $\mathbb{R}^5$, are shown in the left column of Figure 4. For PCA, the decoder function corresponds to multiplication with the loadings matrix, whose i-th column is the i-th eigenface. 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 utilize 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.

4 Conclusions

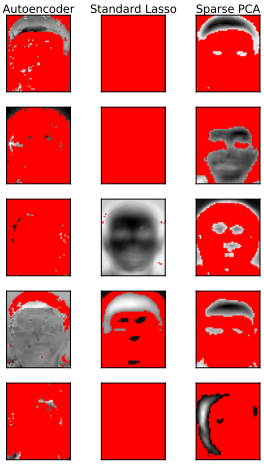
We proposed path lasso, that creates structured sparsity in feedforward neural networks, by using a group lasso penalty to remove all connections between two nodes in two non-adjacent layers. We applied path lasso to an autoencoder to obtain sparse, non-linear dimensionality reduction, and showed that 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 did 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. This investigation is left for future work.

(a) Eigenfaces at low sparsity



(b) Reconstruction of test faces at low sparsity



(c) Eigenfaces at high sparsity

(d) Reconstruction of test faces 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.

References

[1] Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.

[2] Ron Zass and Amnon Shashua. Nonnegative sparse pca. In *Advances in neural information processing systems*, pages 1561–1568, 2007.

[3] 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.

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

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

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

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

[8] Michael E Tipping. Sparse kernel principal component analysis. In *Advances in neural information processing systems*, pages 633–639, 2001.

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

[10] 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.

[11] 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.

[12] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] R Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

A Proofs

Proof of Lemma 1. Using the definition of matrix multiplication, with k running over all paths connecting x_{i_0} and y_{i_L}, we obtain

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

□

To prove Theorem 1 we begin with the following lemma:

Lemma 2. *Let $\{A_k\}_{k=1}^n$ be a set of arbitrary matrices and $\{D_k\}_{k=1}^n$ a set of diagonal matrices, where all elements in A_k and 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*

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

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

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

and for the right hand side

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

By the definition of f, $f(a^1_{ik_1}) \cdot f(a^2_{k_1 k_2}) \cdot \ldots f(a^n_{k_{n-1} j}) \geq 0$ with equality only if at least one of $a^1_{ik_1},\ a^2_{k_1 k_2}, \ldots a^n_{k_{n-1} j}$ is zero, which implies that $a^1_{ik_1} \cdot a^2_{k_1 k_2} \cdot \ldots a^n_{k_{n-1} j} = 0$.

The sum

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

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

$$a^1_{ik_1} \cdot a^2_{k_1 k_2} \cdot \ldots a^n_{k_{n-1} j} = 0$$

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

$$f(a^1_{ik_1}) \cdot f(a^2_{k_1 k_2}) \cdot \ldots f(a^n_{k_{n-1} j}) = 0 \implies a^1_{ik_1} \cdot a^2_{k_1 k_2} \cdot \ldots a^n_{k_{n-1} j} \cdot d^1_{ii} \cdot d^2_{k_1 k_1} \cdot \ldots d^n_{k_{n-1} k_{n-1}} = 0.$$

Finally summing only zeros we get

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

□

Proof of Theorem 1. 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 2 recursively for $1 \leq l \leq L$ as

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

Applying the chain rule we get

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{o_L}}{\partial \boldsymbol{o_0}} = \frac{\partial \boldsymbol{o_L}}{\partial \boldsymbol{i_L}} \cdot \frac{\partial \boldsymbol{i_L}}{\partial \boldsymbol{o_{L-1}}} \cdot \frac{\partial \boldsymbol{o_{L-1}}}{\partial \boldsymbol{i_{L-1}}} \cdot \frac{\partial \boldsymbol{i_{L-1}}}{\partial \boldsymbol{o_{L-2}}} \cdot \ldots \frac{\partial \boldsymbol{o_1}}{\partial \boldsymbol{i_1}} \cdot \frac{\partial \boldsymbol{i_1}}{\partial \boldsymbol{o_0}}, \tag{8}$$

where $\frac{\partial \boldsymbol{o_l}}{\partial \boldsymbol{i_l}}$ is a diagonal matrix, since Φ_l is an element-wise operator, and $\frac{\partial \boldsymbol{i_L}}{\partial \boldsymbol{o_{l-1}}} = W_l$. We can now apply Lemma 2 with $f(x) = x^2$ to Equation 8 to get

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

where the equivalence comes from the definition of W_{GL} and the implication from Lemma 2. □

Proof of Theorem 2. 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:

$$\begin{aligned}&p^k_{i_L i_0} \cdot \left(1 - \frac{\alpha\lambda}{(W_{\text{GL}})_{i_L i_0}}\right)^+ = |w^L_{i_L i^k_{L-1}}| \cdot |w^{L-1}_{i^k_{L-1} i^k_{L-2}}| \cdot \ldots |w^1_{i^k_1 i_0}| \cdot \left(1 - \frac{\alpha\lambda}{(W_{\text{GL}})_{i_L i_0}}\right)^+ \\ &= \text{sign}(w^L_{i_L i^k_{L-1}}) \cdot (|w^L_{i_L i^k_{L-1}}| - \alpha\lambda^L_{i_L i^k_{L-1}})^+ \cdot \ldots \text{sign}(w^1_{i^k_1 i_0}) \cdot (|w^1_{i^k_1 i_0}| - \alpha\lambda^1_{i^k_1 i_0})^+ \\ &=: \text{sign}(w^L_{i_L i^k_{L-1}}) \cdot \tilde{w}^L_{i_L i^k_{L-1}} \cdot \ldots \text{sign}(w^1_{i^k_1 i_0}) \cdot \tilde{w}^1_{i^k_1 i_0}.\end{aligned} \tag{9}$$

Here, the second equality is our assumption, $\lambda^l_{i^k_l i^k_{l-1}} \in [0, |w^l_{i^k_l i^k_{l-1}}|/\alpha]$ 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 have the same value for i_0 and i_L and thus the same path penalty, we obtain, for a given combination of i_0 and i_L

$$\left(\sum_{i_{L-1}=1}^{d_{L-1}} \cdots \sum_{i_1=1}^{d_1} |w^L_{i_L i_{L-1}}| \cdot \ldots |w^1_{i_1 i_0}|\right) \cdot \left(1 - \frac{\alpha\lambda}{(W_{\text{GL}})_{i_L i_0}}\right)^+ = \sum_{i_{L-1}=1}^{d_{L-1}} \cdots \sum_{i_1=1}^{d_1} \tilde{w}^L_{i_L i_{L-1}} \cdot \ldots \tilde{w}^1_{i_1 i_0} \tag{10}$$

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

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

Thus, solving Equation 5 results in the absolute values of links that are 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. □

B Modified Non-Negative Matrix Factorization

We solve the non-negative matrix factorization problem in Equation 5 with coordinate descent, using a modified version of the solver in python's scikit-learn module ([24]), which in turn is based on [25] and [26]. The aim is to minimize $||V - W \cdot \prod_{i=1}^{I} M_i \cdot H||_F^2$ for $I \in \mathbb{N}_0$, keeping each entry in W, M_i and H between zero and its seed.

Since coordinate descent updates each matrix separately, keeping the others fixed, we always have $W \cdot \prod_{i=1}^{I} M_i \cdot H = \tilde{W} \cdot \tilde{M} \cdot \tilde{H}$, where the three matrices on the right hand side are products of the matrices of the left hand side. Thus we need update rules for $\tilde{W}$ and $\tilde{M}$, since we can use the same algorithm for $\tilde{W}$ and $\tilde{H}$ by taking transpose. In [26] the case for two matrices is described, i.e. $I = 0$. Our contribution is the update rule for $\tilde{M}$ (and trivially adding the upper constraint on the solution).

B.1 Update Rule for $\tilde{W}$

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

$$\min_{\tilde{W}:\, 0 \le \tilde{W}_{ir} \le (\tilde{W}_0)_{ir}} \frac{1}{2}||V - \tilde{W}\tilde{H}||_F^2 + \sum_{i,r}\left(\lambda_1 \tilde{W}_{ir} + \frac{\lambda_2}{2}\tilde{W}_{ir}^2\right)$$

where $\tilde{W}_0$ is the seed. We update each element $\tilde{W}_{ir}$ by adding sE_{ir}, for an optimal s, where E_{ir} is a matrix with all zeros, except element (i, r), which is 1. Defining

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

we get

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

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

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

which is minimized at

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

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

$$(\tilde{W}_{ir})^{t+1} = \max((\tilde{W}_0)_{ir}, \min(0, (\tilde{W}_{ir})^t + s^*)).$$

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

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

$$\min_{\tilde{M}:\, 0 \le \tilde{M}_{pr} \le (\tilde{M}_0)_{pr}} \frac{1}{2}||V - \tilde{W}\tilde{M}\tilde{H}||_F^2 + \sum_{p,r}\left(\lambda_1 \tilde{M}_{pr} + \frac{\lambda_2}{2}\tilde{M}_{pr}^2\right)$$

$$
\begin{aligned}
g_{pr}^{\tilde{M}}(s) &:= \frac{1}{2}\sum_{i,j}(V_{ij} - (\tilde{W}(\tilde{M} + sE_{pr})\tilde{H})_{ij})^2 + \lambda_1(\tilde{M}_{pr} + s) + \frac{\lambda_2}{2}(\tilde{M}_{pr} + s)^2 \\
&= \frac{1}{2}\sum_{i,j}(V_{ij} - (\tilde{W}\tilde{M}\tilde{H})_{ij} - s\tilde{W}_{ip}\tilde{H}_{rj})^2 + \lambda_1(\tilde{M}_{pr} + s) + \frac{\lambda_2}{2}(\tilde{M}_{pr} + s)^2 \\
(g_{pr}^{\tilde{M}})'(s) &= \sum_{i,j}(-V_{ij}\tilde{W}_{ip}\tilde{H}_{rj} + (\tilde{W}\tilde{M}\tilde{H})_{ij}\tilde{W}_{ip}\tilde{H}_{rj} + 2s\tilde{W}_{ip}^2\tilde{H}_{rj}^2) + \lambda_1 + \lambda_2(\tilde{M}_{pr} + s) \\
(g_{pr}^{\tilde{M}})''(s) &= \sum_{i,j}(2\tilde{W}_{ip}\tilde{H}_{rj}^2) + \lambda_2 \\
(g_{pr}^{\tilde{M}})'(0) &= \sum_{i,j}(-V_{ij}\tilde{W}_{ip}\tilde{H}_{rj} + (\tilde{W}\tilde{M}\tilde{H})_{ij}\tilde{W}_{ip}\tilde{H}_{rj}) + \lambda_1 + \lambda_2\tilde{M}_{pr} \\
&= (-\tilde{W}^\top V\tilde{H}^\top + \tilde{W}^\top\tilde{W}\tilde{M}\tilde{H}\tilde{H}^\top)_{pr} + \lambda_1 + \lambda_2\tilde{M}_{pr} \\
(g_{pr}^{\tilde{M}})''(0) &= \sum_i(\tilde{W}^\top)_{pi}\tilde{W}_{ip}\sum_j\tilde{H}_{rj}(\tilde{H}^\top)_{jr} + \lambda_2 = (\tilde{W}^\top\tilde{W})_{pp}(\tilde{H}\tilde{H}^\top)_{rr} + \lambda_2 \\
s^* &= \frac{(\tilde{W}^\top V\tilde{H}^\top - \tilde{W}^\top\tilde{W}\tilde{M}\tilde{H}\tilde{H}^\top)_{pr} - \lambda_1 - \lambda_2\tilde{M}_{pr}}{(\tilde{W}^\top\tilde{W})_{pp}(\tilde{H}\tilde{H}^\top)_{rr} + \lambda_2} \\
(\tilde{M}_{pr})^{t+1} &= \max((\tilde{M}_0)_{pr}, \min(0, (\tilde{M}_{ir})^t + s^*)).
\end{aligned}
$$