# Concrete Autoencoders for Differentiable Feature Selection and Reconstruction

**Abubakar Abid** [* 1]   **Muhammad Fatih Balin** [* 2]   **James Zou** [3]

## Abstract

We introduce the *concrete autoencoder*, an end-to-end differentiable method for global feature selection, which efficiently identifies a subset of the most informative features and simultaneously learns a neural network to reconstruct the input data from the selected features. Our method is unsupervised, and is based on using a concrete selector layer as the encoder and using a standard neural network as the decoder. During the training phase, the temperature of the concrete selector layer is gradually decreased, which encourages a user-specified number of discrete features to be learned. During test time, the selected features can be used with the decoder network to reconstruct the remaining input features. We evaluate concrete autoencoders on a variety of datasets, where they significantly outperform state-of-the-art methods for feature selection and data reconstruction. In particular, on a large-scale gene expression dataset, the concrete autoencoder selects a small subset of genes whose expression levels can be use to impute the expression levels of the remaining genes. In doing so, it improves on the current widely-used expert-curated L1000 landmark genes, potentially reducing measurement costs by 20%. The concrete autoencoder can be implemented by adding just a few lines of code to a standard autoencoder.

## 1. Introduction

High-dimensional datasets often pose a challenge for machine learning algorithms. Feature selection methods aim to reduce dimensionality of data by identifying the subset of relevant features in a dataset. A large number of algorithms have been proposed for feature selection in both supervised and unsupervised settings (Kohavi & John, 1997;

---
[*]Equal contribution   [1]Department of Electrical Engineering, Stanford University, Stanford, United States [2]Bogazici University, Istanbul, Turkey [3]Department of Biomedical Data Sciences, Stanford University, Stanford, United States. Correspondence to: James Zou <jamesz@stanford.edu>.

Wang et al., 2013). These methods provide insight into the relationship between features in complex data and can simplify the process of training downstream models. Feature selection is particularly useful if the data with the full set of features is expensive or difficult to collect, as it can eliminate the need to measure irrelevant or redundant features.

As a motivating example, consider a dataset that consists of the expression of various genes across tissue samples. Such "omics" measurements are increasingly carried out to fully characterize biological samples at the individual and single-cell level (Bock et al., 2016; Huang et al., 2017). Yet it remains expensive to conduct all of the biological assays that are needed to characterize such samples. It is natural to ask: *what are the most important features in this dataset? Are there redundant features that do not need to be measured?* The idea of only measuring a subset of biological features and then reconstructing the remaining features is not new; in fact, this line of thinking has motivated the identification of the landmark genes, also known as the L1000 genes, which are a small subset of the over 20,000 human genes. The expression levels of the L1000 are strongly correlated with the expression levels of other genes, and thus this subset can be measured cheaply and then used to impute the remaining gene expression levels (Lamb et al., 2006).

The problem of feature selection is different from the more general problem of dimensionality reduction. Standard techniques for dimensionality reduction, such as principal components analysis (Hotelling, 1933) and autoencoders (Hinton & Salakhutdinov, 2006), are be able to represent data with fewer dimensions while preserving maximal variance or minimizing reconstruction loss. However, such methods do not select a set of features present in the original dataset, and thus cannot be directly used to eliminate redundant features and reduce experimental costs. We emphasize the *unsupervised* nature of this problem: specific prediction tasks may not be known ahead of time, and thus it is important to develop methods that can identify a subset of features while allowing imputation of the remaining features with minimal distortion for arbitrary downstream tasks.

In this paper, we propose a new end-to-end method to perform feature subset selection and imputation that leverages the power of deep autoencoders for discrete feature selection. Our method, the *concrete autoencoder*, uses a relaxation of the discrete distribution, the Concrete distribution (Maddi-

| (a) | (b) | (c) | (d) |
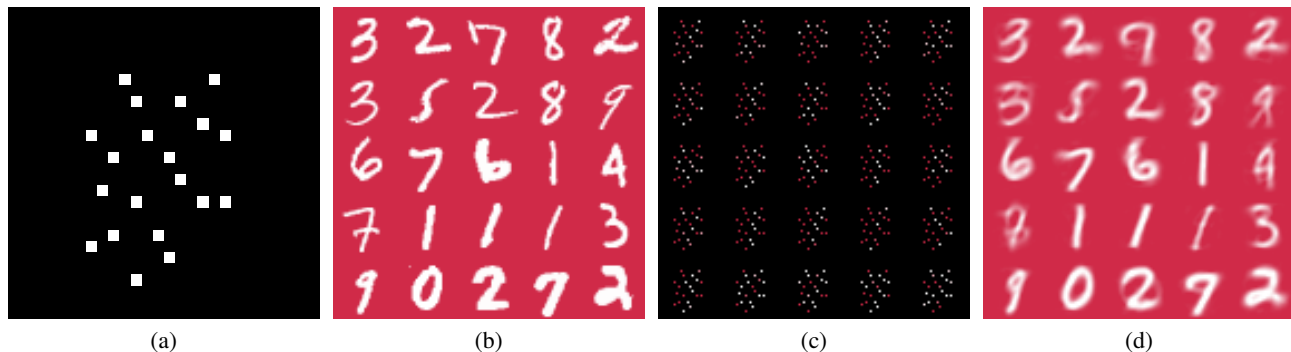
*Figure 1.* **Demonstrating concrete autoencoders on the MNIST dataset.** Here, we show the results of using concrete autoencoders to select in an unsupervised manner the $k = 20$ most informative pixels of images in the MNIST dataset. (a) The 20 selected features (out of the 784 pixels) on the MNIST dataset are shown in white. (b) A sample of input images in MNIST dataset with the top 2 rows being training images and the bottom 3 rows being test images. (c) The same input images with only the selected features shown as colored dots. (d). The reconstructed versions of the images, using only the 20 selected pixels, shows that generally the digit is identified correctly and some stylistic features, such as the swirl in the digit "2", are captured. (*cf.* figures in Appendix A which show the results of applying concrete autoencoder to individual classes of digits.)

son et al., 2016), and the reparametrization trick (Kingma & Welling, 2013) to differentiate through an arbitrary (e.g. reconstruction) loss and select input features to minimize this loss. A visual example of results from our method is shown in Fig. 1, where the concrete autoencoder selects the 20 most informative pixels (out of a total of 784) on the MNIST dataset, and reconstructs the original images with high accuracy.

We test concrete autoencoders on a variety of datasets, and find that they generally outperform state-of-the-art methods for feature selection and data reconstruction. We have made the code for our algorithm and experiments available on a public repository[1].

**Related Works** Feature selection methods are generally divided into three classes: filter, wrapper, and embedded methods. Filter methods rank the features of the dataset using statistical tests, such as the variance, in order to select features that individually maximize the desired criteria (Battiti, 1994; Duda et al., 2012). Filter methods generally do not consider interactions between features and do not provide a way to impute the remaining features; one must train a separate algorithm to do so. Wrapper methods select subsets of features that maximize a objective function, which is optimized over the choice of input features using a black-box optimization method, such as sequential search or genetic algorithms (Kohavi & John, 1997; Goldberg & Holland, 1988). Since wrapper methods evaluate subsets of features, they are able to detect potential relationships between features, but usually at the expense

of increased computation time. Embedded methods also consider relationships between features but generally do so more efficiently as they incorporate feature selection into the learning phase of another algorithm. A well-known example is the Lasso (Tibshirani, 1996), which can be used to select features for regression by varying the strength of $\ell_1$ regularization.

Many embedded unsupervised feature selection algorithms use regularization as the means to select discrete features. The popular UDFS algorithm Yang et al. (2011) incorporates $\ell_{2,1}$ regularization on a set of weights applied to the input to select features most useful for local discriminative analysis. Similarly, the MCFS algorithm (Cai et al., 2010) uses regularization to solve for the features which preserve the clustering structure in the data. The recently-proposed AEFS (Han et al., 2017) algorithm also uses $\ell_{2,1}$ regularization on the weights of the encoder that maps the input data to a latent space and optimizes these weights for their ability to reconstruct the original input.

In this paper, we select discrete features using an embedded method but without resorting to regularization. Rather, we use a relaxation of the discrete random variables, the Concrete distribution (Maddison et al., 2016), which allows a low-variance estimate of the gradient through discrete stochastic nodes. By using Concrete random variables, we can directly parametrize the selection of input features, and differentiate through the parameters. As we show through experiments in Section 4, this leads to lower reconstruction errors on real-world datasets compared to the aforementioned regularization-based methods.

---

[1]Code available at: https://github.com/mfbalin/Concrete-Autoencoders

## 2. Problem Formulation

We now describe the problem of global feature selection. Although global feature selection is relevant for both unsupervised and supervised settings, we describe here the unsupervised case, which is the primary focus of this paper, and defer discussion of the supervised case to Appendix F.

Consider a data-generating probability distribution $p(\boldsymbol{x})$ over a $d$-dimensional space. The goal is to learn a subset $S \subseteq \{1, 2 \ldots d\}$ of features of specified size $|S| = k$ and also learn a reconstruction function $f_\theta(\cdot) : \mathbb{R}^k \to \mathbb{R}^d$, such that the expected loss between the reconstructed sample $f_\theta(\boldsymbol{x}_S)$ and the original sample $\boldsymbol{x}$ is minimized, where $\boldsymbol{x}_S \in \mathbb{R}^k$ consists of those elements $\boldsymbol{x}_i$ such that $i \in S$. In other words, we would like to optimize

$$\underset{S,\theta}{\arg\min} \, \mathrm{E}_{p(x)}[\|f_\theta(\boldsymbol{x}_S) - \boldsymbol{x}\|_2] \qquad (1)$$

In practice, we do not know $p(\boldsymbol{x})$; rather we have $n$ samples, generally assumed to be drawn i.i.d. from $p(\boldsymbol{x})$. These samples can be represented in a data matrix $X \in \mathbb{R}^{n \times d}$, and so the goal becomes choosing $k$ columns of $X$ such that sub-matrix $X_S \in \mathbb{R}^{n \times k}$, defined analogously to $\boldsymbol{x}_S$, can be used to reconstruct the original matrix $X$. Let us overload $f_\theta(X_S)$ to mean the matrix that results from applying $f_\theta(\cdot)$ to each of the rows of $X_S$ and stacking the resulting outputs. We seek to minimize the empirical reconstruction error:

$$\underset{S,\theta}{\arg\min} \|f_\theta(X_S) - X\|_F, \qquad (2)$$

where $\|\cdot\|_F$ denotes the Frobenius norm of the matrix. The principal difficulty in solving (2) is the optimization over the discrete set of features $S$, whose choices grow exponentially in $d$. Thus, even for simple choices of $f_\theta(\cdot)$, such as linear regression, the optimization problem in (2) is NP-hard to solve (Amaldi & Kann, 1998).

Furthermore, the complexity of $f_\theta(\cdot)$ can significantly affect reconstruction error and even the choice of $S$. More expressive and non-linear choices for $f_\theta(\cdot)$, such as neural networks, will naturally allow for lower reconstruction error, potentially at the expense of a more difficult optimization problem. We seek to develop a method that can approximate the solution for any given class of functions $f_\theta(\cdot)$, from linear regression to deep fully-connected neural networks.

Finally, we note that the choice of mean-squared reconstruction error as the metric for optimization in (1) and (2) stems from the fact that it is a smooth differentiable function that serves as a proxy for many downstream analyses, such as clustering performance and classification accuracy. However, other differentiable metrics, such as a variational approximation of the mutual information between $f_\theta(X_S)$ and $X$, may be considered as well (Chen et al., 2018).

## 3. Proposed Method

The concrete autoencoder is an adaption of the standard autoencoder (Hinton & Salakhutdinov, 2006) for discrete feature selection. Instead of using series of fully-connected layers for the encoder, we propose a *concrete selector layer* with a user-specified number of nodes, $k$. This layer selects stochastic linear combinations of input features during training, which converge to a discrete set of $k$ features by the end of training and during test time.

The way in which input features are combined depends on the *temperature* of this layer, which we modulate using a simple annealing schedule. As the temperature of the layer approaches zero, the layer selects $k$ individual input features. The decoder of a concrete autoencoder, which serves as the reconstruction function, is the same as that of a standard autoencoder: a neural network whose architecture can be set by the user based on dataset size and complexity. In effect, then, the concrete autoencoder is a method for selecting a discrete set of features that are optimized for an arbitrarily-complex reconstruction function. We describe the ingredients for our method in more detail in the next two subsections.

### 3.1. Concrete Selector Layer

The concrete selector layer is based on Concrete random variables (Maddison et al., 2016; Jang et al., 2016). A Concrete random variable can be sampled to produce a continuous relaxation of the one-hot vector. The extent to which the one-hot vector is relaxed is controlled by a temperature parameter $T \in (0, \infty)$. To sample a Concrete random variable in $d$ dimensions with parameters $\boldsymbol{\alpha} \in \mathbb{R}_{>0}^d$ and $T$, one first samples a $d$-dimensional vector of i.i.d. samples from a Gumbel distribution (Gumbel, 1954), $\boldsymbol{g}$. Then each element of the sample $\boldsymbol{m}$ from the Concrete distribution is defined as:

$$\boldsymbol{m}_j = \frac{\exp((\log \boldsymbol{\alpha}_j + \boldsymbol{g}_j)/T)}{\sum_{k=1}^d \exp((\log \boldsymbol{\alpha}_k + \boldsymbol{g}_k)/T)}, \qquad (3)$$

where $\boldsymbol{m}_j$ refers to the $j^{\text{th}}$ element in a particular sample vector. In the limit $T \to 0$, the concrete random variable smoothly approaches the discrete distribution, outputting one hot vectors with $\boldsymbol{m}_j = 1$ with probability $\boldsymbol{\alpha}_j / \sum_p \boldsymbol{\alpha}_p$. The desirable aspect of the Concrete random variable is that it allows for differentiation with respect to its parameters $\boldsymbol{\alpha}$ via the reparametrization trick (Kingma & Welling, 2013).

We use Concrete random variables to select input features in the following way. For each of the $k$ nodes in the concrete selector layer, we sample a $d$-dimensional Concrete random variable $\boldsymbol{m}^{(i)}, i \in \{1 \ldots k\}$ (note that the superscript here indexes the node in the selector layer, whereas the subscript earlier referred to the element in the vector). The $i^{\text{th}}$ node in the selector layer $u^{(i)}$ outputs $\boldsymbol{x} \cdot \boldsymbol{m}^{(i)}$. This is, in general, a

**Input:** training dataset $X \in \mathbb{R}^{n \times d}$, number of features to select $k$, decoder network $f_\theta(\cdot)$, number of epochs $B$, learning rate $\lambda$, initial temp. $T_0$, final temp. $T_B$.

Initialize the parameters of the concrete autoencoder.
**for** $b \in \{1 \ldots B\}$ **do**
    Adjust temp. of Concrete dist. $T = T_0(T_B/T_0)^{b/B}$
    **for** $i \in \{1 \ldots k\}$ **do**
        Sample $m^{(i)} \sim \text{Concrete}(\alpha^{(i)}, T)$
        Let $u^{(i)} = X \cdot m^{(i)}$
    **end for**
    Define $U$ to be the matrix $\in \mathbb{R}^{n \times k}$ that results from horizontally concatenating the $u^{(1)} \ldots u^{(k)}$.
    Let the loss $L$ be defined as $\|f_\theta(U) - X\|_F$
    Compute gradient of the loss w.r.t. $\theta$ using backpropagation and w.r.t each $\alpha^{(i)}$ using reparametrization trick.
    Update $\theta \leftarrow \theta - \lambda \nabla_\theta L$, and $\alpha^{(i)} \leftarrow \alpha^{(i)} - \lambda \nabla_{\alpha^{(i)}} L$
**end for**

**Return:** decoder $f_\theta(\cdot)$ and Concrete params. $\alpha^{(i)}$

(a)          (b)

*Figure 2.* **Concrete autoencoder architecture and pseudocode.** (a) The architecture of a concrete autoencoder consists of a single encoding layer, shown in brown, and arbitrary decoding layers (e.g. a deep feedforward neural network), shown in teal. The encoder has one neuron for each feature to be selected. During the training phase, the $i^{\text{th}}$ neuron $u^{(i)}$ takes the value $x^\top m^{(i)}$, $m^{(i)} \sim \text{Concrete}(\alpha^{(i)}, \text{T})$. During test time, these weights are fixed and the element with the highest value in $\alpha^{(i)}$ is selected by the corresponding $i^{\text{th}}$ hidden neuron. The architecture of the *decoder* remains the same during train and test time, namely that $\hat{x} = f_\theta(u)$, where $u$ is the vector consisting of each $u^{(i)}$. (b) Here, we show pseudocode for the concrete autoencoder algorithm, see Appendix B for more details.

weighted linear combination of the input features, but notice that when $T \to 0$, each node in the concrete selector layer outputs exactly one of the input features. After the network is trained, during test time, we thus replace the concrete selector layer with a discrete $\arg\max$ layer in which the output of the $i^{\text{th}}$ neuron is $x_{\arg\max_j \alpha_j^{(i)}}$.

We randomly initialize $\alpha_i$ to small positive values, to encourage the selector layer to stochastically explore different linear combinations of input features. However, as the network is trained, the values of $\alpha_i$ become more sparse, as the network becomes more confident in particular choices of input features, reducing the stochasticity in selected features. The concrete autoencoder architecture is shown in Fig. 2(a) and the pseudocode for training in Fig. 2(b).

### 3.2. Annealing Schedule

The temperature of Concrete random variables in the concrete selector layer has a significant affect on the output of the nodes. If the temperature is held high, the concrete selector layer continuously outputs a linear combination of features. On the contrary, if the temperature is held low, the concrete selector layer is not able to explore different combinations of features and converges to a poor local minimum. Neither fixed temperature allows the concrete selector layer to converge to informative features.

Instead, we propose a simple annealing schedule that sets

the temperature for all of the concrete variables, initially beginning with a high temperature $T_0$ and gradually decaying the temperature until a final temperate $T_B$ at each epoch according to a first-order exponential decay: $T(b) = T_0(T_B/T_0)^{b/B}$ where $T(b)$ is the temperature at epoch number $b$, and $B$ is the total number of epochs. We compare various methods for setting the temperature of the concrete selector nodes in Fig. 3. We find that this annealing schedule allows the concrete selector layer to effectively stochastically explore combinations of features in the initial phases of training, while in the later stages of training, the lowered temperature allows the the network to converge to informative individual features.

## 4. Experiments

In this section, we carry out experiments to compare the performance of concrete autoencoders to other feature subset selections on standard public datasets. For all of the experiments, we use Adam optimizer with a learning rate of $10^{-3}$. The initial temperature of the concrete autoencoder $T_0$ was set to 10 and the final temperature $T_B$ to 0.01. We trained the concrete autoencoder until until the mean of the concrete samples exceeded 0.99. For UDFS and AEFS, we swept values of each regularization hyperparameter and report the results with optimal hyperparameters according to mean squared error for reconstruction for each method.

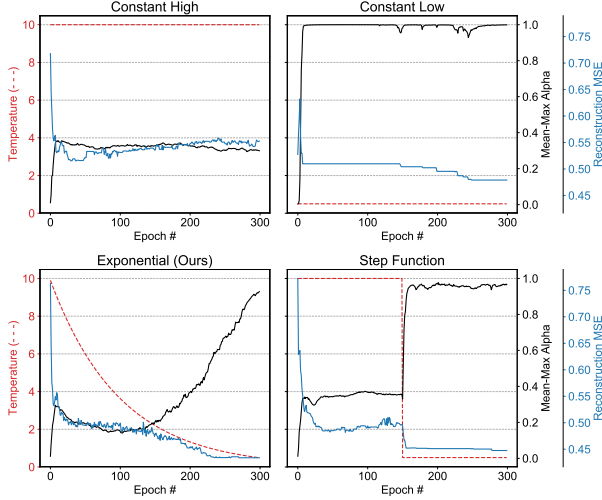Furthermore, since the reconstruction $f_\theta(\cdot)$ can overfit to

*Figure 3.* **Annealing schedules for the concrete autoencoder.** Here, we show the effect of different annealing schedules on a concrete autoencoder trained on the MNIST dataset with $k = 20$ selected features. At each epoch, we plot the temperature in red, average of the largest value in each concrete sample $\boldsymbol{m}^{(i)}$ in black, as well the reconstruction error (using linear regression with the top $k = 20$ features on validation data), shown in blue. If the temperature is kept high, the concrete samples do not converge to individual features, and the reconstruction error remains large (top left). If the temperature is kept low, the samples immediately converge to poor features, and the error remains large (top right). If the temperature is exponentially decayed (the annealing schedule we use), the samples converge to informative features, and the reconstruction error reaches a suitable minimum (bottom left). Finally, if the temperature is dropped abruptly, the samples converge, but the error is suboptimal (bottom right).

patterns particular to the training set, we divide each dataset randomly into train, validation, and test datasets according to a 72-8-20 split[2]. We use the training set to learn the parameters of the concrete autoencoders, the validation set to select optimal hyperparameters, and the test set to evaluate generalization performance, which we report below.

We compare concrete autoencoders to many of the unsupervised feature selection methods mentioned in Related Works including UDFS, MCFS, and AEFS. We also include principal feature analysis (PFA), proposed by Lu et al. (2007), which is a popular method for selecting discrete features based on PCA, as well as a spectral method, the Laplacian score (He et al., 2006). Where available, we made use of `scikit-feature` implementation of each method (Li et al., 2016). In our experiments, we also include, as upper

---

[2]For the MNIST, MNIST-Fashion, and Epileptic datasets, we only used 6000, 6000 and 8000 samples respectively to train and validate the model (using a 90-10 split), because of long runtime of the UDFS algorithm. The remaining samples were used for the test set.

bounds on performance, dimensionality reduction methods that are not restricted to choosing individual features. In experiments with linear decoders, we use PCA and in experiments with non-linear decoders, we use equivalent autoencoders. The methods, because they allow $k$ *combinations* of features, bound the performance of any feature selection techniqe.

We evaluate these methods on a number of datasets (the sizes of the datasets can be found in Table 1):

**MNIST and MNIST-Fashion** consist of 28-by-28 grayscale images of hand-written digits and clothing items, respectively. We choose these datasets because they are widely known in the machine learning community. Although these are image datasets, the objects in each image are centered, which means we can meaningfully treat each 784 pixels in the image as a separate feature.

**ISOLET** consists of preprocessed speech data of people speaking the names of the letters in the English alphabet. This dataset is widely used as a benchmark in the feature selection literature. Each feature is one of the 617 quantities produced as a result of preprocessing, including spectral coefficients and sonorant features.

**COIL-20** consists of centered grayscale images of 20 objects. Images of the objects were taken at pose intervals of 5 degrees amounting to 72 images for each object. During preprocessing, the images were resized to produce 20-by-20 images, with each feature being one of the 400 pixels.

**Smartphone Dataset for Human Activity Recognition** consists of sensor data collected from a smartphone mounted on subjects while they performed several activities such as walking upstairs, standing and laying. Each feature represents one of the 561 raw or processed quantities from the sensors on the phone.

**Mice Protein Dataset** consists of protein expression levels measured in the cortex of normal and trisomic mice who had been exposed to different experimental conditions. Each feature is the expression level of one protein.

**GEO Dataset** consists of gene expression profiles measured from a large number of samples through a microarray-based platform. Each of the 10,463 features represents the expression level of one gene.

To evaluate the various feature selection methods, we examine two metrics, both reported on a hold-out test set:

**Reconstruction error**: We extract the $k$ selected features. We pass the resulting matrix $X_S$ through the reconstruction function $f_\theta$ that we have trained. We measure the Frobenius norm between the original and reconstructed test matrices $\|f_\theta(X_S) - X\|_F$, normalized by the number of features $d$.

**Classification accuracy**: We extract the $k$ features selected
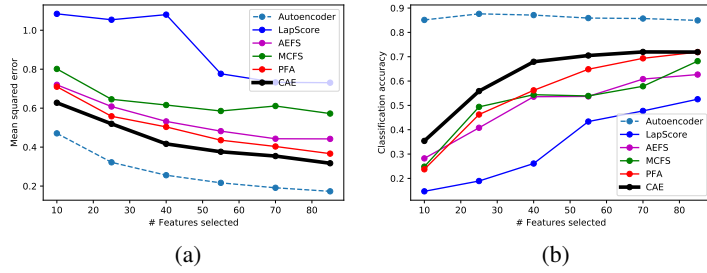
(a)  (b)

*Figure 4.* **Results on the ISOLET dataset using non-linear decoders.** Here, we compare the concrete autoencoder (CAE) to other feature selection methods using a 1-hidden layer neural network as the reconstructor. (a) We find that across all values of $k$ tested, concrete autoencoders have lowest reconstruction errors (b) We find that the features learned by the concrete autoencoder also tend result in higher classification accuracies.

by the method. We then pass the resulting matrix $X_S$ to an extremely randomized trees classifier (Geurts et al., 2006), a variant of random forests that has been used with feature selection methods in prior literature (Drotár et al., 2015). We measure the accuracy between the predicted labels and true labels, which are available for each of the datasets. Note that the labels are only used for training the classifier and not for the feature selection.

### 4.1. Concrete Autoencoders (Non-Linear Decoder)

First, we constructed a concrete autoencoder with a non-linear decoder architecture consisting of one hidden layer with $3k/2$ neurons, with $k$ being the number of selected features. We performed a series of experiments with the ISO-LET dataset, which is widely used as a benchmark in prior feature selection literature. We benchmarked each feature selection method (besides UDFS whose run-time was prohibitive) with varying numbers of of features (from $k = 10$ to $k = 85$), measuring the reconstruction error using a 1-hidden-layer neural network as well as classification accuracy. The number of neurons in the hidden layer of the reconstruction network was varied within $[4k/9, 2k/3, k, 3k/2]$, and the network with the highest validation accuracy was selected and measured on the test set.

To control for the performance of the reconstruction network, we trained each reconstruction network for the same number of epochs, 200. For the concrete autoencoder, we did not use the decoder that was learned during training, but re-trained the reconstruction networks from scratch. Our resulting classification accuracies and reconstruction error on each dataset are shown in Fig. 4. We find that the concrete autoencoder consistently outperformed other feature selection methods on the ISOLET dataset.

### 4.2. Concrete Autoencoders (Linear Decoder)

Next, we carried out a series of experiments in which we compared concrete autoencoders with linear decoders to the other methods using linear regression as the reconstruction function. Since linear regression can be trained easily to convergence, this allowed us to isolate the effect of using the concrete selector layer for feature selection, and allowed us to train on a wider variety of datasets with less risk of

overfitting. We selected a fixed number $k = 50$ of features with each method, with the exception of the Mice Protein Dataset, for which we used $k = 10$ due to its small size.

After selecting the features using concrete autoencoder and the other feature selection methods, we trained a standard linear regressor with no regularization to impute the original features. The resulting reconstruction errors on a hold-out test set are shown in Table 1. We also used the selected features to measure classification accuracies, which are shown in Table 2 in Appendix C. On almost all datasets, we found that the concrete autoencoder continued to have the lowest reconstruction error and a high classification accuracy.

### 4.3. Interpreting Related Features

An added benefit of using the concrete selector layer is that it allows the user to not only identify the most informative features for reconstruction, but also identify sets of related features through examination of the learned Concrete parameters $\boldsymbol{\alpha}^{(i)}$. Because the concrete selector layer samples the input features stochastically based on $\boldsymbol{\alpha}^{(i)}$, any of the features with the large values in the vector $\boldsymbol{\alpha}^{(i)}$ may be selected, and are thus likely to be correlated to one another.

In Fig. 5, we show how this can reveal related features by visualizing the top 3 pixels with the highest values in the $\boldsymbol{\alpha}^{(i)}$ vector for each of the 20 concrete selector nodes on the MNIST digits. We notice that the pixels that are selected by each node are spatially close to one another, which agrees with intuitive notions of related features, as neighboring pixel values are likely to be correlated in handwritten digits. These patterns are even more striking when generated for individual classes of digits; in that case, the set of correlated pixels may even suggest the direction of the stroke when the digit was written (see Appendix D for more details). Such analysis may be carried out more generally to find sets of related features, such as sets of related genes in a gene expression dataset.

### 4.4. Case Study: L1000 Gene Expression

We now turn to a large-scale test of the concrete autoencoder: examining whether we can improve *gene expression inference*. Gene expression inference arises from an im-

| Dataset | $(n, d)$ | *PCA* | Lap | AEFS | UDFS | MCFS | PFA | CAE |
|---|---|---|---|---|---|---|---|---|
| MNIST | (10000, 784) | *0.012* | 0.070 | 0.033 | 0.035 | 0.064 | 0.051 | **0.026** |
| MNIST-Fashion | (10000, 784) | *0.012* | 0.128 | 0.047 | 0.133 | 0.096 | 0.043 | **0.041** |
| COIL-20 | (1440, 400) | *0.008* | 0.126 | 0.061 | 0.116 | 0.085 | **0.061** | 0.093 |
| Mice Protein | (1080, 77) | *0.003* | 0.603 | 0.783 | 0.867 | 0.695 | 0.871 | **0.372** |
| ISOLET | (7797, 617) | *0.009* | 0.344 | 0.301 | 0.375 | 0.471 | 0.316 | **0.299** |
| Activity | (5744, 561) | *0.002* | 0.139 | 0.112 | 0.173 | 0.170 | 0.197 | **0.108** |

*Table 1.* **Reconstruction errors of feature selection methods using linear regression reconstruction.** Here, we show the mean-squared errors of the various feature methods on six publicly available datasets. Here Lap refers to the Laplacian score and CAE refers to the concrete autoencoder. For each method, we select $k = 50$ features (except for mice protein, where we use $k = 10$ because the original data is lower dimensional) and use a linear regressor for reconstruction. All reported values are on a hold-out test set. (Lower is better.)
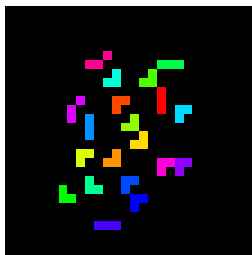


*Figure 5.* **Pixel groups selected by concrete selector nodes on MNIST.** Here, we illustrate the top 3 pixels selected by each of the 20 nodes in the concrete layer when trained on the MNIST dataset. We color each group of 3 pixels with the same color (note that some colors are repeated because of the limited color palette). *cf.* Appendix D, which shows pixel group for classes of digits.

portant problem in molecular biology: characterizing the state of cells in different biological conditions. In particular, the response of cells to diseases, mutations, and drugs is often characterized by the measurement of gene expression patterns (Lamb et al., 2006).

However, measuring all of the genes expressed in a human cell can be expensive, and thus researchers have looked to computational methods to reduce the cost and time required for gene expression profiling. In particular, researchers from the LINCS Project found that, because gene expression is correlated in different conditions, a set of roughly a thousand carefully-chosen genes can capture most of the gene expression information in the entire human transcriptome (Peck et al., 2006). It is thus possible to use a linear regression model trained on genome-wide gene expression to *infer* the gene expression values of the remaining genes. More recently, Chen et al. (2016) showed that it is possible to leverage the representation power of neural networks to improve the accuracy of gene expression inference in an approach they referred to as D-GEX.

Here, we ask whether it is possible to use concrete autoencoders to determine a good subset of genes, perhaps as an

alternative to the landmark genes, without utilizing any prior biological knowledge of gene networks or gene function. We relied only on a large dataset of gene expression data, from which we aim to select the most informative features.

We used the version of the GEO dataset used in the D-GEX paper, and followed the same preprocessing scheme to obtain a dataset of sample size 112,171 and dimensionality 10,463 genes. We then randomly partitioned the dataset in a manner similar to that performed by Chen et al. (2016): as a result, the training set had 88,807 samples, the validation set had 11,101, and the test set had 12,263. We then considered 3 kinds of reconstruction functions: in the simplest case, we considered multitarget linear regression, and we also implemented neural networks with 1 and 2 hidden layers. See Appendix E for the architecture of the networks.

First, we trained a concrete autoencoder to select 943 using only a linear regression decoder. An analysis of the selected genes showed very little overlap with the landmark genes: only 90 of the CAE-selected 943 genes were among the landmark genes. For consistency with the D-GEX results, we used this same set of 943 genes, selected by a concrete autoencoder with a linear decoder, with all of our reconstruction networks.

We trained each reconstruction networks to impute all of the original 10,463 genes. We measured the reconstruction error on a hold-out test set that was used neither to train the concrete autoencoder nor the reconstruction functions. Our results are summarized in Fig. 6(a), where we plot the mean-squared error of imputation. We show that not only is it possible to use concrete autoencoders to perform gene selection gene expression inference in a differentiable, end-to-end manner on large-scale datasets, doing so improves the performance of the gene expression imputation on a holdout test set of gene expression by around 3% for each architecture, which is significant as the L1000 landmark genes were expert curated using a combination of computational prediction with domain knowledge, and is a very strong benchmark and is widely used in genomics.
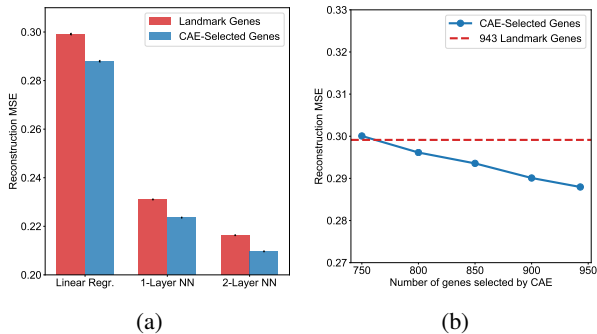
Figure 6. **Imputation errors of concrete autoencoders and landmark genes.** Here, we show the mean-squared error of the imputation task using both the 943 landmark genes (red) and the 943 genes selected by the concrete autoencoder (blue) on the test set. The task is to impute the expression of all 10,463 genes. We observe about a 3% reduction (note that y-axis begins at 0.20) of the reconstruction error when using the genes selected by the concrete autoencoders (CAE) across all architectures. These are results averaged over three trials. Standard deviation bars are shown but were very low, as the final imputations were very similar across all trials. (b) We train the CAE with different numbers of selected features, and calculate the MSE using linear regression on the test set. We find that we can achieve a similar MSE to the landmark genes using only around 750 genes, a 20% reduction in the number of genes measured.

Next, we investigated whether it would be possible to obtain similar accuracies as to the landmark genes while using a smaller set of CAE-selected genes. We trained concrete autoencoders from scratch using $k = 750, 800, 850, 900, 943$, using the same architecture described in Appendix E and using a linear regression decoder. We found that using linear regression as the reconstruction function, we could obtain reconstruction MSEs about as low as the landmark genes, using only 750 genes, which represents roughly a $20\%$ reduction in the number of genes measured, potentially saving substantial experimental costs. These results are illustrated in Fig. 6(b).

## 5. Discussion

In this paper, we have proposed a new method for differentiable, end-to-end feature selection via backpropagation. At its core, the concrete autoencoder uses Concrete random variables and the reparametrization trick to allow gradients to flow through a layer that stochastically selects discrete input features. The stochasticity of the concrete autoencoder allows it to efficiently explore and converge to a subset of input features of specified size that minimizes a particular loss, as described in Section 3. The learned parameters can be further probed to allow the anlayst to interpret related features, as demonstrated in Section 4.3. This makes con-

crete autoencoders different from many competing methods, which rely on regularization to encourage sparse feature selection.

We show via experiments on a variety of public datasets that concrete autoencoders effectively minimize the reconstruction error and maximize classification accuracy using selected features. In the six public datasets that we tested concrete autoencoders, we found that concrete autoencoders outperformed many different complex feature selection methods. This remains the case even when we reduced the decoder layer to be a single linear layer, showing that the concrete selector node is useful even when selecting input features that minimize the loss when using a linear regression as the reconstruction function.

Because the concrete autoencoder is an adaptation of the standard autoencoder, it scales easily to datasets with many samples or high dimensionality. We demonstrated this in section 4.4 using a gene expression dataset with more than 100,000 samples and 10,000 features, where the features selected by the concrete autoencoder outperformed the state-of-the-art gene subset. Furthermore, because of its general formulation, the concrete autoencoder can be easily extended in many ways. For example, it possible to use concrete autoencoders in a supervised manner – to select a set of features that minimize a cross-entropy loss, for example, rather than a reconstruction loss. More details and examples of this approach are provided in Appendix F. Another possible extension is to attach different costs to selecting different features, for example if certain features represent tests or assays that are much more expensive than others. Such as cost may be incorporated into the loss function and allow the analyst to trade off cost for accuracy.

Advantages of the concrete autoencoder include its generality and ease of use. Implementing the architecture in popular machine learning frameworks requires only modifying a few lines of code from a standard autoencoder. Furthermore, the runtime of the concrete autoencoder is similar to that of the standard autoencoder and improves with hardware acceleration and parallelization techniques commonplace in deep learning. The only additional hyperparameters of the concrete autoencoder are the initial and final temperatures used in the annealing schedule. We find that the default values used in this paper work well for a variety of datasets.

Concrete autoencoders, like the other feature selection methods we compared with in this paper, do not provide $p$-values or statistical significance quantification. Features discovered through concrete autoencoders should be validated through hypothesis testing or additional analysis using relevant domain knowledge. We believe that the concrete autoencoder can be of particular use in simplifying assays and experiments that measure a large number of related quantities, such as medical lab tests and genotype sequencing.

## Acknowledgments

## References

Amaldi, E. and Kann, V. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science*, 209(1-2):237–260, 1998.

Battiti, R. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, 5(4):537–550, 1994.

Bock, C., Farlik, M., and Sheffield, N. C. Multi-omics of single cells: strategies and applications. *Trends in biotechnology*, 34(8):605–608, 2016.

Cai, D., Zhang, C., and He, X. Unsupervised feature selection for multi-cluster data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 333–342. ACM, 2010.

Chen, J., Song, L., Wainwright, M., and Jordan, M. Learning to explain: An information-theoretic perspective on model interpretation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 883–892. PMLR, 10–15 Jul 2018.

Chen, Y., Li, Y., Narayan, R., Subramanian, A., and Xie, X. Gene expression inference with deep learning. *Bioinformatics*, 32(12):1832–1839, 2016.

Drotár, P., Gazda, J., and Smékal, Z. An experimental comparison of feature selection methods on two-class biomedical datasets. *Computers in biology and medicine*, 66:1–10, 2015.

Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern classification*. John Wiley & Sons, 2012.

Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

Goldberg, D. E. and Holland, J. H. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

Gumbel, E. J. *Statistical theory of extreme values and some practical applications: a series of lectures*. Number 33. US Govt. Print. Office, 1954.

Han, K., Li, C., and Shi, X. Autoencoder feature selector. *arXiv preprint arXiv:1710.08310*, 2017.

He, X., Cai, D., and Niyogi, P. Laplacian score for feature selection. In *Advances in neural information processing systems*, pp. 507–514, 2006.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *science*, 313 (5786):504–507, 2006.

Hotelling, H. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

Huang, S., Chaudhary, K., and Garmire, L. X. More is better: recent progress in multi-omics data integration methods. *Frontiers in genetics*, 8:84, 2017.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kohavi, R. and John, G. H. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, 1997.

Lamb, J., Crawford, E. D., Peck, D., Modell, J. W., Blat, I. C., Wrobel, M. J., Lerner, J., Brunet, J.-P., Subramanian, A., Ross, K. N., et al. The connectivity map: using gene-expression signatures to connect small molecules, genes, and disease. *science*, 313(5795):1929–1935, 2006.

Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., and Liu, H. Feature selection: A data perspective. *arXiv preprint arXiv:1601.07996*, 2016.

Lu, Y., Cohen, I., Zhou, X. S., and Tian, Q. Feature selection using principal feature analysis. In *Proceedings of the 15th ACM international conference on Multimedia*, pp. 301–304. ACM, 2007.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Peck, D., Crawford, E. D., Ross, K. N., Stegmaier, K., Golub, T. R., and Lamb, J. A method for high-throughput gene expression signature analysis. *Genome biology*, 7 (7):R61, 2006.

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

Wang, G., Song, Q., Sun, H., Zhang, X., Xu, B., and Zhou, Y. A feature subset selection algorithm automatic recommendation method. *Journal of Artificial Intelligence Research*, 47:1–34, 2013.

Yang, Y., Shen, H. T., Ma, Z., Huang, Z., and Zhou, X. l2,
1-norm regularized discriminative feature selection for
unsupervised learning. 2011.

# Appendices

## A. Selected Features for Single Classes in MNIST

Here, we show additional examples of using the concrete autoencoder on subsets of the MNIST data that consist of a single digit. Here, we select $k = 10$ features for each subset of data.
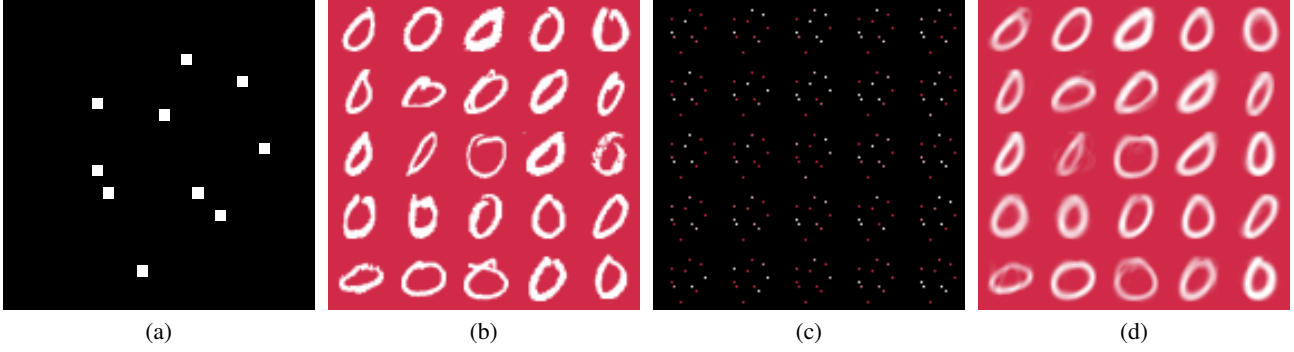


(a)      (b)      (c)      (d)

*Figure 7.* Here, we show the results of using concrete autoencoders to select the $k = 10$ most informative pixels of images of the digit 0 in the MNIST dataset. Compare with Fig. 1 in the main paper for more information.



(a)      (b)      (c)      (d)

*Figure 8.* Here, we show the results of using concrete autoencoders to select the $k = 10$ most informative pixels of images of the digit 0 in the MNIST dataset. Compare with Fig. 1 in the main paper for more information.



(a)      (b)      (c)      (d)

*Figure 9.* Here, we show the results of using concrete autoencoders to select the $k = 10$ most informative pixels of images of the digit 0 in the MNIST dataset. Compare with Fig. 1 in the main paper for more information.
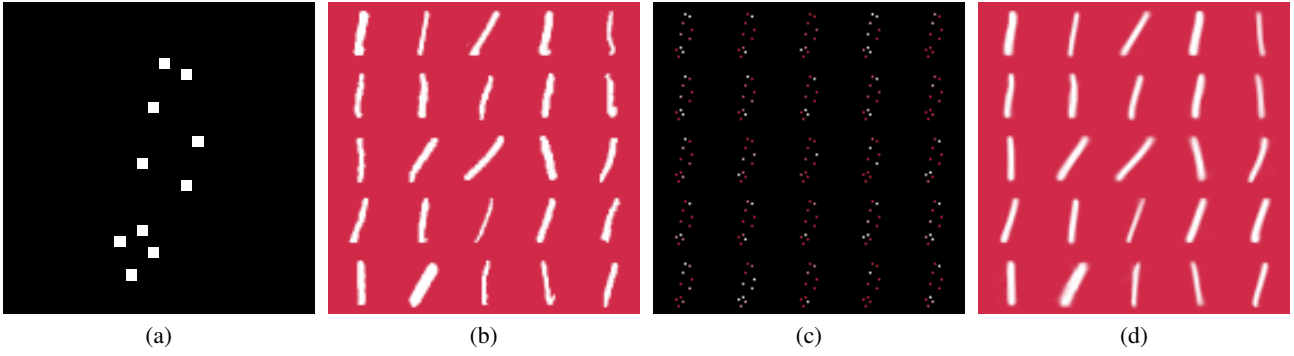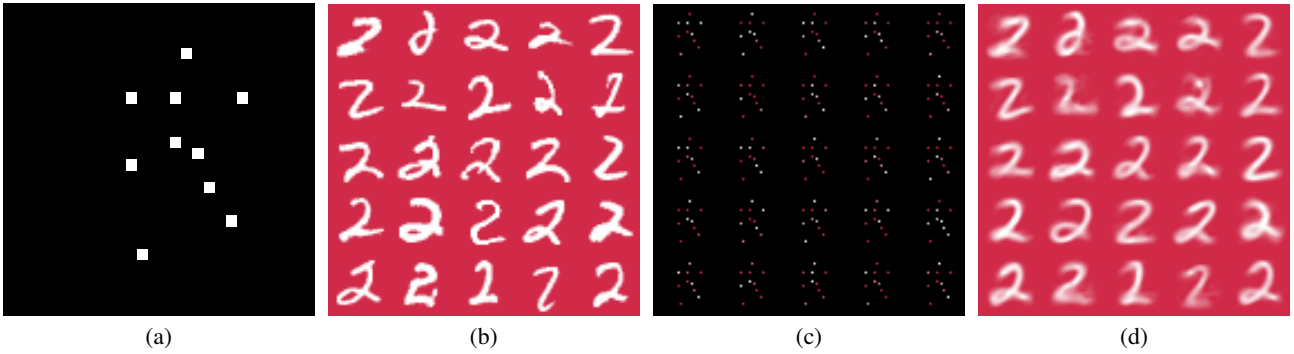
## B. Pseudocode for a Concrete Autoencoder

Here, we have the pseudocode for training the concrete autoencoder in more detail. We also describe how to use a trained concrete autoencoder for feature selection on new data, as well as how to use the concrete autoencoder for imputation.

---

*Algorithm 1.* Training a Concrete Autoencoder

---

**Input:** training dataset $X \in \mathbb{R}^{n \times d}$, number of features to select $k$, decoder network $f_\theta(\cdot)$, number of epochs $B$, learning rate $\lambda$, initial temp $T_0$, final temp $T_B$.

**for** $i \in \{1 \ldots k\}$ **do**
    Initialize a $d$-dimensional vector of parameters $\boldsymbol{\alpha}^{(i)}$ with small positive values.
**end for**
Initialize the parameters $\theta$ of the reconstruction function in a standard way for neural networks.
**for** $b \in \{1 \ldots B\}$ **do**
    Let $T = T_0(T_B/T_0)^{b/B}$
    **for** $i \in \{1 \ldots k\}$ **do**
        Sample $\boldsymbol{m}^{(i)} \sim \text{Concrete}(\boldsymbol{\alpha}^{(i)}, T)$
        Let $X_S^{(i)} = X \cdot \boldsymbol{m}^{(i)}$
    **end for**
    Define $X_S$ to be the matrix $\in \mathbb{R}^{n \times k}$ that results from horizontally concatenating the $X_S^{(1)} \ldots X_S^{(k)}$.
    Let the loss $L$ be defined as $\|f_\theta(X_S) - X\|_F$
    Compute the gradient of the loss w.r.t. $\theta$ using backpropagation and w.r.t each $\boldsymbol{\alpha}^{(i)}$ using the reparametrization trick.
    Update the parameters $\theta \leftarrow \theta - \lambda \nabla_\theta L$, and $\boldsymbol{\alpha}^{(i)} \leftarrow \boldsymbol{\alpha}^{(i)} - \lambda \nabla_{\boldsymbol{\alpha}^{(i)}} L$
**end for**

**Return:** trained reconstruction function $f_\theta(\cdot)$ and trained Concrete parameters $\boldsymbol{\alpha}^{(i)}$

---

---

*Algorithm 2.* Using a Trained Concrete Autoencoder for Feature Selection

---

**Input:** test sample $\boldsymbol{x} \in \mathbb{R}^d$, trained Concrete parameters $\boldsymbol{\alpha}^{(i)}$

**for** $i \in \{1 \ldots k\}$ **do**
    Let $m^{(i)} = \arg\max_j(\boldsymbol{\alpha}_j^{(i)})$, where $j$ indexes the elements of the sample vector
    Let $\boldsymbol{x}_S^{(i)} = \boldsymbol{x}_{m^{(i)}}$
**end for**

**Return:** $\boldsymbol{x}_S$

---

---

*Algorithm 3.* Using a Trained Concrete Autoencoder for Imputation

---

**Input:** test sample with subset of features $\hat{\boldsymbol{x}} \in \mathbb{R}^k$, trained reconstruction function $f_\theta(\cdot)$

**Return:** $f_\theta(\hat{\boldsymbol{x}})$

---

## C. Classification Accuracies for Feature Selection Methods with Linear Reconstruction

We carried out a series of experiments in which we compared concrete autoencoders with linear decoders to the other feature selection methods using linear regression as the reconstruction function. We selected $k = 50$ of features with each method.

After selecting the features using concrete autoencoder and the other feature selection methods, we trained a standard linear regressor with no regularization to impute the original features. The resulting reconstruction errors on a hold-out test set are shown in Table 1 in the main text. We also used the selected features to measure classification accuracies, which are shown in Table 2 here. Generally, we find that the concrete autoencoder continues to have the lowest reconstruction error and a high (but not always the highest) classification accuracy.

| Dataset | $(n, d)$ | *PCA* | Lap | AEFS | UDFS | MCFS | PFA | CAE |
|---------|----------|-------|-----|------|------|------|-----|-----|
| MNIST | (10000, 784) | *0.925* | 0.646 | 0.690 | 0.892 | 0.807 | 0.852 | **0.906** |
| MNIST-Fashion | (10000, 784) | *0.825* | 0.517 | 0.580 | 0.547 | 0.513 | **0.683** | 0.677 |
| COIL-20 | (1440, 400) | *0.996* | 0.389 | 0.580 | 0.556 | 0.635 | **0.642** | 0.586 |
| Mice Protein | (1080, 77) | *0.721* | 0.134 | 0.125 | **0.139** | **0.139** | 0.130 | 0.134 |
| ISOLET | (7797, 617) | *0.895* | 0.407 | 0.576 | 0.455 | 0.522 | 0.622 | **0.685** |
| Activity | (5744, 561) | *0.796* | 0.280 | 0.240 | 0.287 | 0.295 | 0.364 | **0.420** |

*Table 2.* **Classification accuracies of feature selection methods.** Here, we show the classification accuracies of the various feature methods on six publicly available datasets. Here CAE refers to the concrete autoencoder. For each method, we select $k = 50$ features (except for mice protein dataset, for which we use $k = 10$) and use a neural network with 1 hidden layer for reconstruction. All reported values are on the test set. The classifier used here was a Extremely Randomized Trees classifier (a variant of Random Forests) with the number of trees being 50. (Higher is better.)

# D. Examples of Feature Groups in MNIST Digits

Here, we show examples of feature groups that were selected by the concrete autoencoder on single classes of digits in the MNIST dataset (see Section 4.3 in the main text for more details). The patterns in the case of single classes of digits are even more striking; here, the set of correlated pixels can be used to infer the direction of the stroke when the digit was written, as correlated pixels are more likely to be part of the same stroke. For example, consider Fig. 10(b), in which the pixel groups shown for the digit '1'. We note that the pixel groups tend to form vertical subsets, as the writing stroke connected those sets of pixels together.
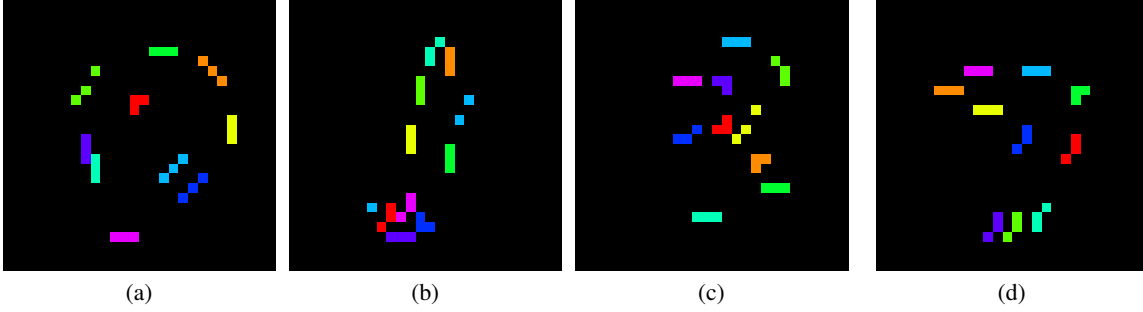


(a)  (b)  (c)  (d)

*Figure 10.* Here, we show the results of using concrete autoencoders to select the $k = 10$ most informative pixels groups of images of the digit 0 in the MNIST dataset. Compare with Fig. 5 in the main paper. Each panel is a different digit: (a) the digit 0, (a) the digit 1, (a) the digit 2, (a) the digit 7

# E. Architecture for GEO Dataset Experiments

For the GEO dataset, we trained three reconstruction networks to perform the imputation from both the landmark and CAE-selected genes. In each case, the hidden layers consisted of 9000 neurons and dropout rate was set to $0.1$. The initial learning rate was $0.001$ and decayed after each epoch by multiplication with $0.95$. The number of epochs was set to 100 for training the reconstruction networks. We used batch sizes of 256.

The concrete autoencoder was trained with a linear decoder network for 5000 epochs. For this dataset, we set the initial temperature to be 10, and the final temperature to be 0.01.

# F. Supervised Concrete Autoencoders

Concrete autoencoders can be easily adapted to the supervised setting by replacing the reconstruction neural network in the decoder with a neural network classifier. The pseudocode, shown below, is quite similar to training the standard concrete autoencoder.

---

*Algorithm 4.* Training a Concrete Autoencoder

---

**Input:** training features $X \in \mathbb{R}^{n \times d}$, training labels $\boldsymbol{y}$, number of features to select $k$, classifier function $f_\theta(\cdot)$, number of epochs $B$, learning rate $\lambda$, initial temperature $T_0$, final temperature $T_B$.

**for** $i \in \{1 \ldots k\}$ **do**
     Initialize a $d$-dimensional vector of parameters $\boldsymbol{\alpha}^{(i)}$ with small positive values.
**end for**
Initialize the parameters $\theta$ of the reconstruction function in a standard way for neural networks.
**for** $b \in \{1 \ldots B\}$ **do**
     Let $T = T_0(T_B/T_0)^{b/B}$
     **for** $i \in \{1 \ldots k\}$ **do**
         Sample $\boldsymbol{m}^{(i)} \sim \text{Concrete}(\boldsymbol{\alpha}^{(i)}, T)$
         Let $X_S^{(i)} = X \cdot \boldsymbol{m}^{(i)}$
     **end for**
     Define $X_S$ to be the matrix $\in \mathbb{R}^{n \times k}$ that results from horizontally concatenating the $X_S^{(1)} \ldots X_S^{(k)}$.
     Let $L$ be the cross entropy loss between the true labels $\boldsymbol{y}$ and the logits $f_\theta(X_S)$
     Compute the gradient of the loss w.r.t. $\theta$ using backpropagation and w.r.t each $\boldsymbol{\alpha}^{(i)}$ using the reparametrization trick.
     Update the parameters $\theta \leftarrow \theta - \lambda \nabla_\theta L$, and $\boldsymbol{\alpha}^{(i)} \leftarrow \boldsymbol{\alpha}^{(i)} - \lambda \nabla_{\boldsymbol{\alpha}^{(i)}} L$
**end for**

**Return:** trained reconstruction function $f_\theta(\cdot)$ and trained Concrete parameters $\boldsymbol{\alpha}^{(i)}$

---

We trained a concrete autoencoder in this supervised manner on the MNIST digits, some representative images are shown in Fig. 11. Generally, we found the imputation quality to be not as good as when the objective function is directly reconstruction error.
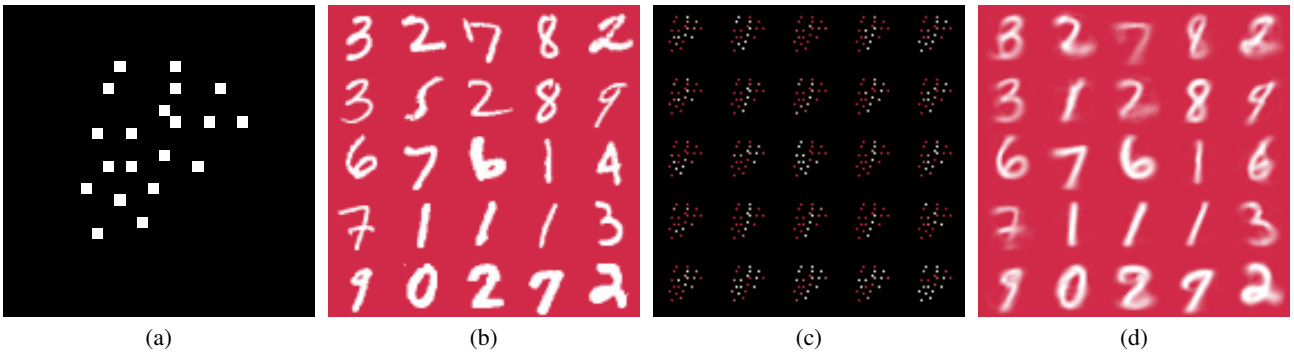


(a)       (b)       (c)       (d)

*Figure 11.* **Demonstrating concrete autoencoders on the MNIST dataset.** Here, we show the results of using concrete autoencoders to select in an supervised manner the $k = 20$ most informative pixels of images in the MNIST dataset. (a) The 20 selected features (out of the 784 pixels) on the MNIST dataset are shown in white. (b) A sample of input images in MNIST dataset with the top 2 rows being training images and the bottom 3 rows being test images. (c) The same input images with only the selected features shown as white dots. (d). The reconstructed versions of the images, using only the 20 selected pixels, shows that generally the digit is identified correctly.