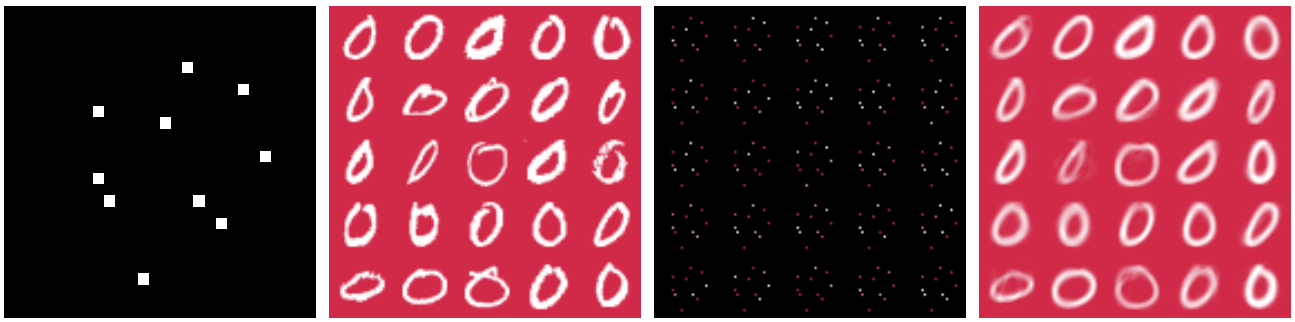
Appendices

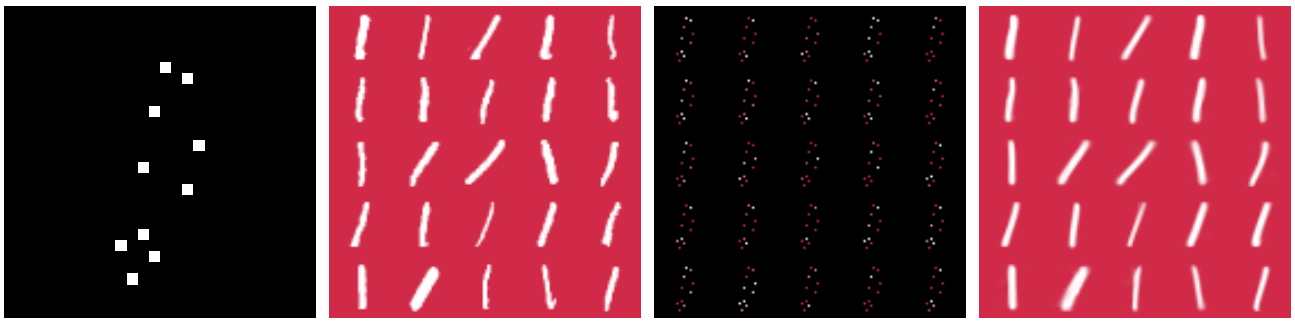
A. Selected Features for Single Classes in MNIST

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.



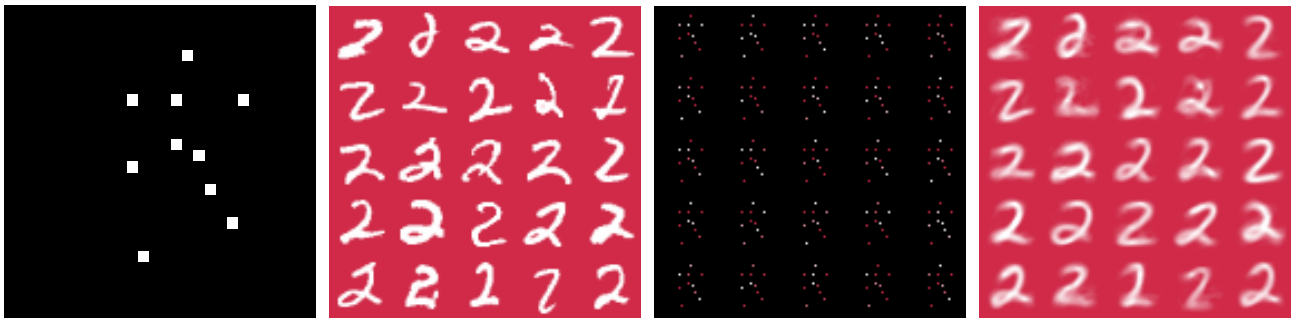
(a) (b) (c) (d)

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. Additional Related Works

In this appendix, we include additional literature that is relevant to concrete autoencoders, but that which we did not have space to include in the main text.

In Chandra & Sharma (2015), the authors propose a complicated method for performing discrete feature selection with autoencoders. However, their method is multi-stage and cannot be optimized in an end-to-end manner. The method we propose instead, using the concrete random variables, shares the same machinery as Louizos et al. (2017), who use concrete random variables to create sparse networks. However, the motivation in that work is to reduce the number of parameters in the network, and not perform feature selection. The idea of using continuous relaxations of random variables for feature selection was also explored by (Yamada et al., 2018). They observed the benefits of using concrete random variables for exploring different features. However, they were unable to get feature selection with concrete random variables to converge, as they did not introduce an annealing schedule as we did in our method.

In Section 4.4, we apply concrete autoencoders to gene expression inference. This allows us to impute the expression of all of the genes from a a measurement of a small number of them. This bears some resemblance to gene imputation methods (Van Dijk et al., 2018; Li & Li, 2018). However, these methods impute the gene information for single cells by utilizing measurements from other cells. They do not directly perform gene selection.

C. 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 λ, initial temp T_0, final temp T_B.

for $i \in \{1 \dots k\}$ **do**
  Initialize a d-dimensional vector of parameters $\boldsymbol{\alpha}^{(i)}$ with small positive values.
end for
Initialize the parameters θ of the reconstruction function in a standard way for neural networks.
for $b \in \{1 \dots B\}$ **do**
  Let $T = T_0(T_B/T_0)^{b/B}$
  for $i \in \{1 \dots k\}$ **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 $X_S^{(1)} \dots X_S^{(k)}$.
  Let the loss L be defined as $\|f_\theta(X_S) - X\|_F$
  Compute the gradient of the loss w.r.t. θ 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 $i \in \{1 \dots k\}$ **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: $f_\theta(\hat{\boldsymbol{x}})$

D. 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.

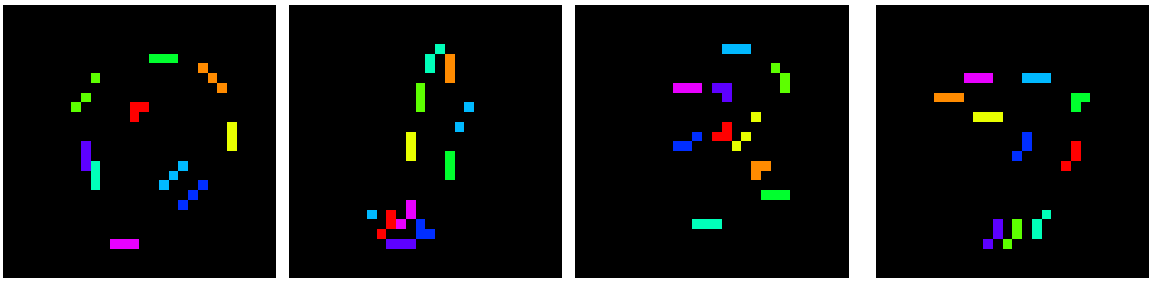
After selecting the features using concrete autoencoder and the other feature selection methods, we trained a standard linear regressor with no regularization to impute the original features. The resulting reconstruction errors on a hold-out test set are shown in Table 1 in the main text. We also used 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.)

E. 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

F. 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.

G. Supervised Concrete Autoencoders and Other Extensions

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.

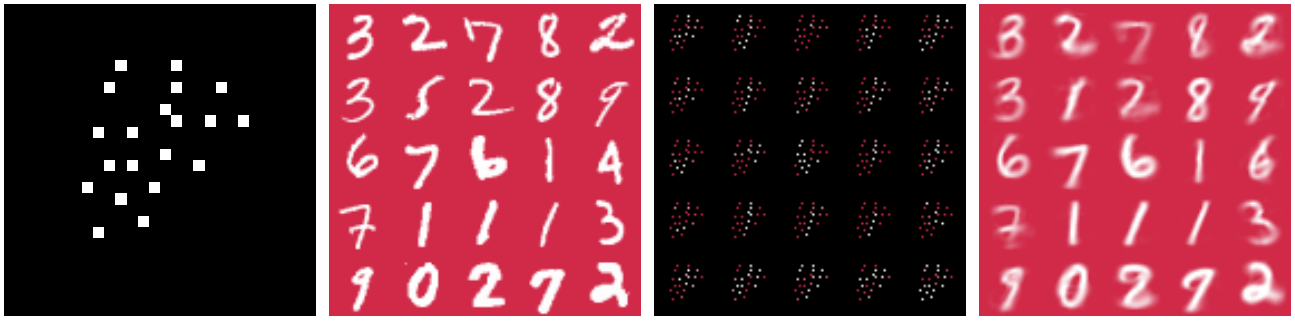
Algorithm 4. Training a Concrete Autoencoder with Supervised Labels

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 λ, initial temperature T_0, final temperature T_B.

for $i \in \{1 \dots k\}$ **do**
  Initialize a d-dimensional vector of parameters $\boldsymbol{\alpha}^{(i)}$ with small positive values.
end for
Initialize the parameters θ of the reconstruction function in a standard way for neural networks.
for $b \in \{1 \dots B\}$ **do**
  Let $T = T_0(T_B/T_0)^{b/B}$
  for $i \in \{1 \dots k\}$ **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 $X_S^{(1)} \dots X_S^{(k)}$.
  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. θ 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 classifier 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. Note that we trained a separate 1-hidden layer neural network that reconstructs the images using only the 20 selected pixels because the decoder of the concrete autoencoder outputs class probabilities.

In the supervised regime, concrete autoencoders can be compared with other deep learning-based feature selection methods,

such as those that utilize knockoffs (Lu et al., 2018; Romano et al., 2018). We carried out empirical comparisons to Deep Knockoffs. We found that on the MNIST dataset (with $k = 20$), while knockoffs were able to recover relevant features with low false discovery rates, concrete autoencoder consistently achieved lower test reconstruction MSE (11% lower RMSE).

Besides supervised learning, the concrete autoencoder can be extended in different ways. One 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. Another possible extension is to add a KL objective that measures the deviation of the concrete node from the uniform distribution (to measure how sparse it is), which may work as a method of regularization.