Autoencoders

Sys-AI

- hi
- shea
- autoencoders

The Problem

- Construct new features from raw features
  - Automatic
  - Complex

- For example, instead of using the raw pixel values of an image, we might want to work with higher-level features like “Is the bottom half darker than the top?” So we’re trying to basically map original data points to new data points that use a different representation.
- Before a lot of newer techniques came around, and still even now, people spend a lot of time hand-engineering useful features. Here, we will try to engineer features automatically, in a way that hopefully beats manual features.
- We would like our features to be complex when needed. They should not just be linear combinations or simple functions of the raw features, as that would limit their expressive power.

The Problem

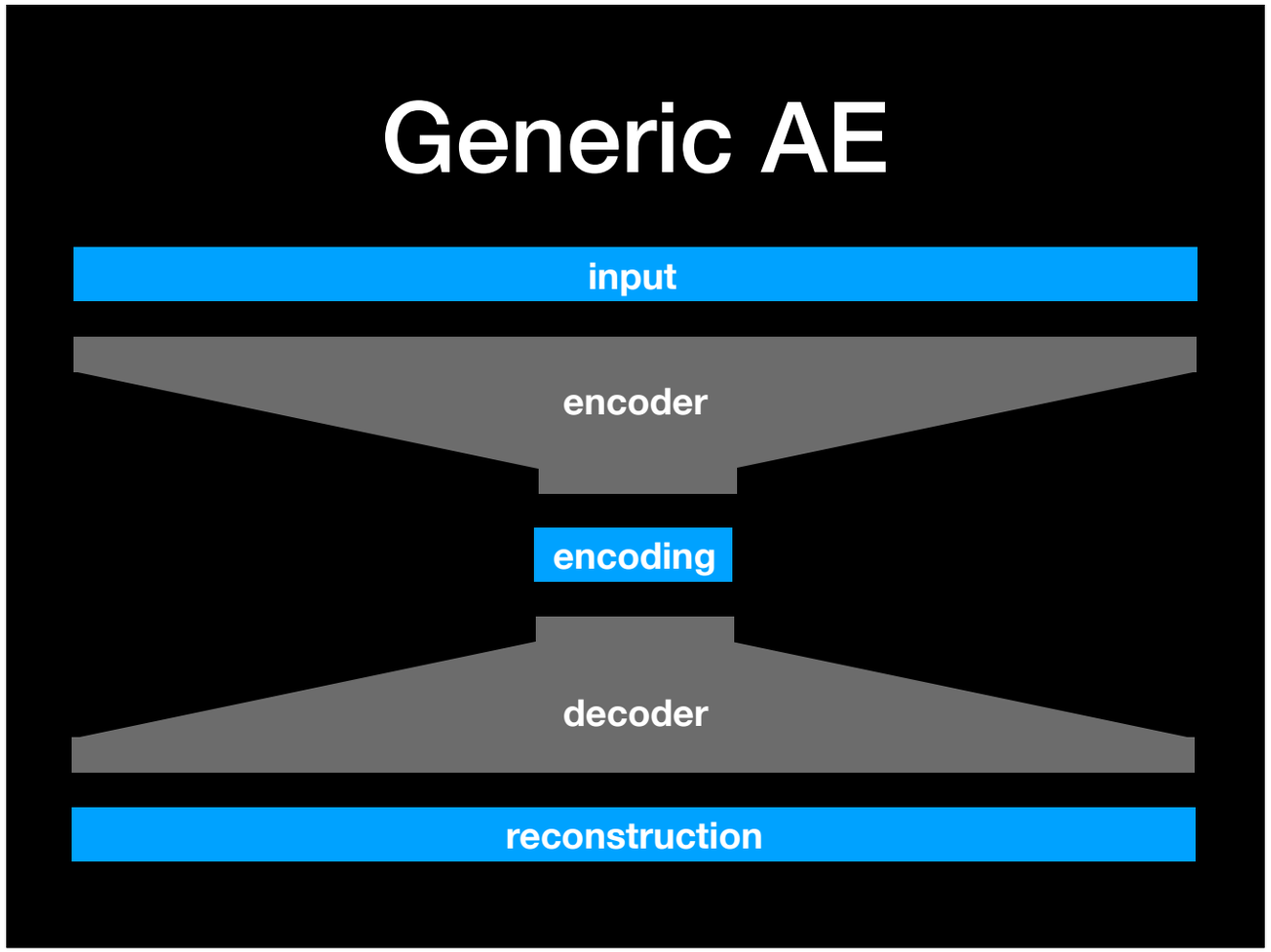
- Dimensionality reduction
- Desirable properties
- Noise tolerance
- Generative model
- Transfer learning

- We might want to construct new features for the purpose of dimensionality reduction. Instead of, for example, working with data points in 250,000-dimensional space, as in the case of 500 by 500 pixel images, we may want to work in a lower-dimensional space. This improves efficiency, because models may be slow with large numbers of features. This may also produce better results, because compressing the data fundamentally requires understanding its underlying structure. By making the number of dimensions very small, we can make data visualizations.
- We might want to enforce some property that we desire, such as sparsity of the features, invariance to slight changes to the raw features, etc.
- We might want to get features that ignore any noise that was present in the raw features.
- We might want to create something that learns the distribution of the data and can generate new data points.
- We might want to train a component that can be used in many machine learning tasks with similar structure, instead of always starting from scratch.

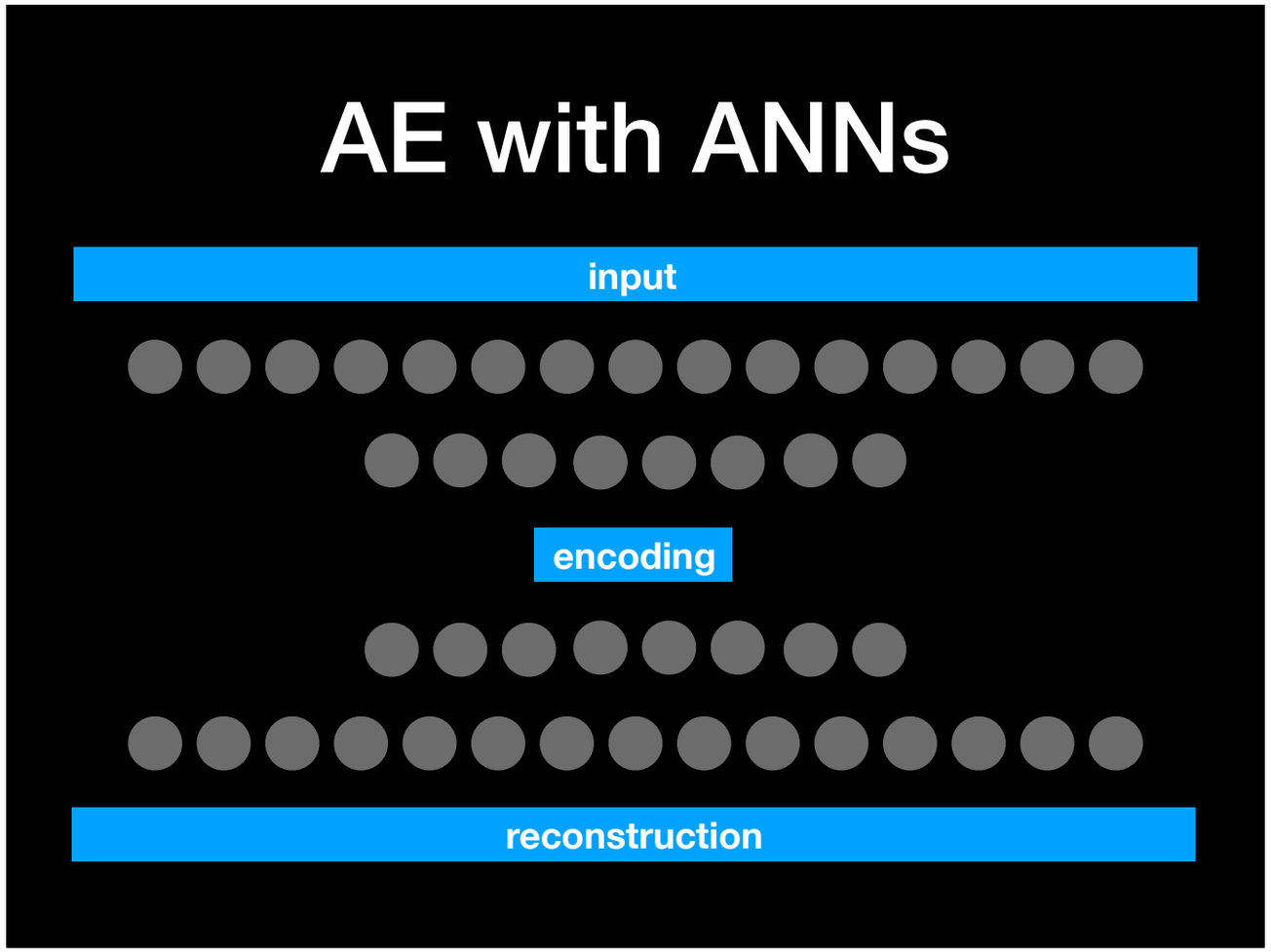
Metrics

- Quality of reconstruction of original data point
  - Sum of per-instance MSEs
- Dimensions, sparsity, noise tolerance, similarity/novelty of generated data points, etc.

- We hope that our autoencoder is representing the original data points well. To test that, we use it to make a reconstruction of the original data point. This reconstruction is lossy because the representation is imperfect— and when doing dimensionality reduction, because the representation has strictly lower information content than the original data point. So the reconstruction will not be exactly right, but we can assess how close it is such as by computing the mean square error between the reconstruction and the original.
- Depending on what we are using autoencoders for, we may consider another aspect of the autoencoder or add an extra term to the loss function. We might want to minimize the number of dimensions, maximize sparsity, maximize noise tolerance, or maximize the similarity and novelty of the generated data points, for example.



- Generally speaking, an autoencoder has two pieces. The encoder maps inputs to an encoding. Here I've illustrated a low-dimensionality encoding, indicated by the narrow box for the encoding vector, but in general the encoding dimensionality equal to or greater than the input dimensionality.
- The second piece is the decoder, which maps encodings to reconstructions of an input that may have produced them.
- The main object of interest is usually the encoding in the middle, but the decoding step is needed because we need to train this thing. We have no notion of "correct" encodings, so we need to see how well the encoding can be used to reconstruct the input in order to assess it.



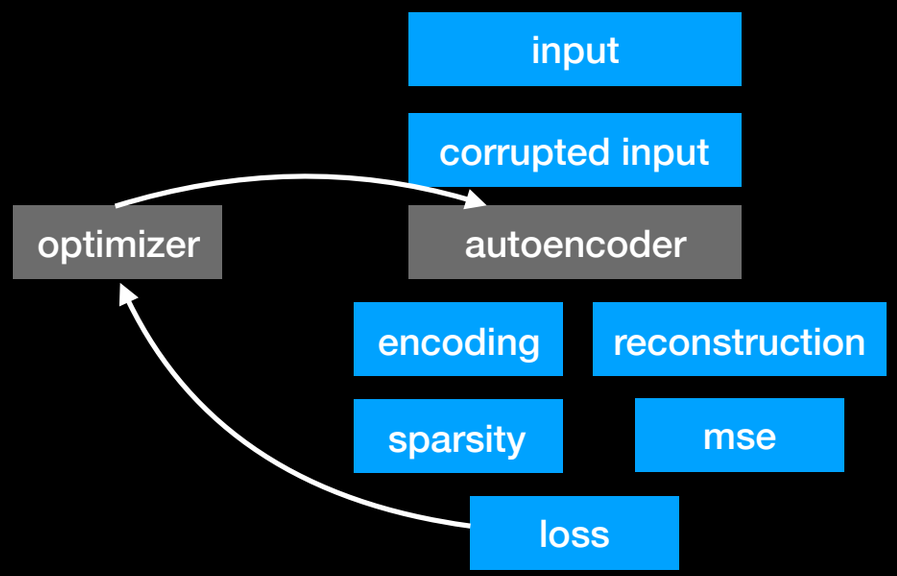
- As we talked about previously, artificial neural networks are good at learning complex representations of data. So typically, we use feedforward artificial neural networks for the encoder and decoder, although in principle anything could be used. The weights in the decoder network are just the inverse of the weights in the encoder network, because it’s a symmetric process.

Use Cases

- Dimensionality reduction: Choose encoding dimensions.
- Desirable properties: Add a term to loss function.
- Noise tolerance: Corrupt the inputs with noise, but use the original inputs as targets.
- Generative model: Create a setup similar to generative-adversarial models using the encodings.
- Transfer learning: Use a convolutional layer instead of a fully-connected one.

- To implement dimensionality reduction, we simply need to choose a number of encoding dimensions that is less than the number of input dimensions.
- To implement a desirable property such as sparsity, we can add a term to the loss function that penalizes the divergence of the encodings' average value from a low threshold.
- To implement noise tolerance, we can corrupt the inputs with noise, but assess the reconstruction using the original input, so that the network is forced to reconstruct denoised data from noisy data.
- To implement a generative model, we can use a setup similar to that of generative-adversarial models, where a separate discriminator model tries to discern genuine outputs of the autoencoder from artificially-generated fakes.
- To implement transfer learning, we can use convolutional layers instead of fully-connected ones. After training, we can use these layers to initialize convolutional neural networks for other tasks.

AE Training



- Of course, the devil is in the details, so here is a more complete example of the pipeline for actually training an autoencoder. It includes some optional pieces that are only used for certain purposes.
- We might start up here with our input, perhaps an image of a handwritten digit, a 784-dimensional vector.
- If we want noise tolerance, then we might corrupt the image with some noise. We'll give the autoencoder the corrupted image, but still expect it to spit out an image that looks like the original.
- We'd then feed that into the autoencoder and do the feedforward process to determine the encoding, in the middle layer of the network and the reconstruction, in the last layer of the network.
- As for the encoding, we might assess its sparsity if that's a quality we desire.
- For the reconstructed image, we will typically calculate its mean square error as compared to the original image.
- We'd add those two terms together to get our loss.
- Lastly, an optimization algorithm of our choice would use gradient descent to update the weights of the network slightly.

Key Innovation

- Combine ANNs with feature construction
  - Very versatile
  - Effective
  - Works in unsupervised setting

- The key innovation of autoencoders is to combine artificial neural networks and feature construction.
- The resulting model is very versatile. By changing hyperparameters such as the encoding dimensionality, adding constraints such as sparsity, preprocessing the inputs, etc., autoencoders can solve a variety of problems.
- Autoencoders are effective. The paper lists a number of problems for which autoencoders helped to beat the previous state-of-the-art and it reconstructs MNIST to a high degree of visual fidelity with only 36, instead of 784, dimensions.
- Autoencoders work in the unsupervised setting, and since data is many orders of magnitude cheaper than labels, this means they can give insights in many situations where supervised methods cannot.
- A shallow autoencoder, consisting of just a single hidden layer, is enough to reconstruct MNIST to high visual fidelity. Using convolutional layers instead of fully connected layers can increase efficiency further. And layers tend to shrink as they get deeper. Considered beside the many large-scale models being produced today, autoencoders give pretty good results for their computational footprint.

Key Results

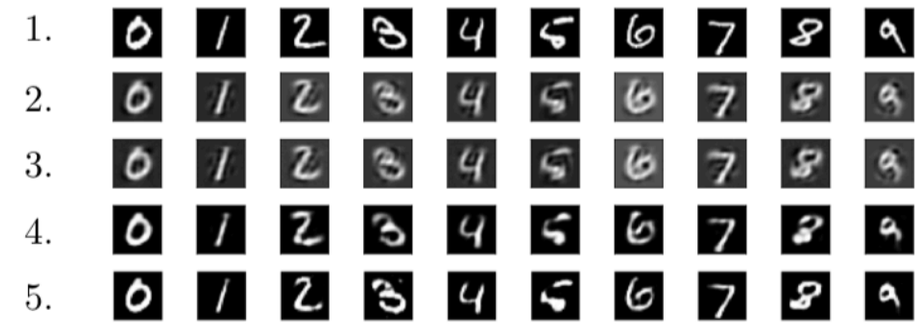
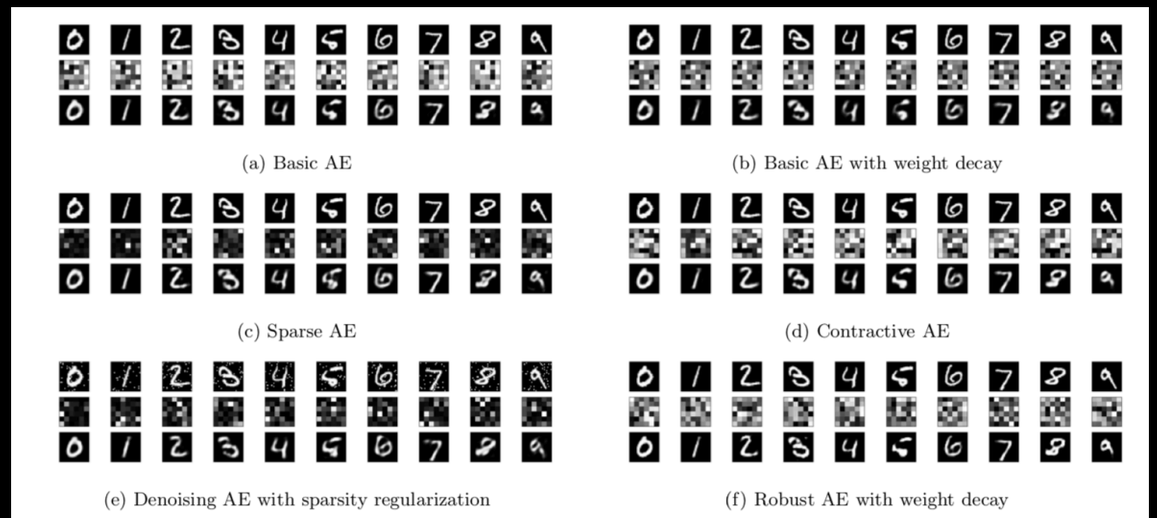


Figure 10: Row 1 shows test samples, second row corresponds to PCA reconstructions, the third one shows those from a linear AE optimizing MSE, row 4 displays reconstructions from a basic AE with tanh activation and cross-entropy as loss function, and last row corresponds to a robust AE.

Key Results



(a) Basic AE

(b) Basic AE with weight decay

(c) Sparse AE

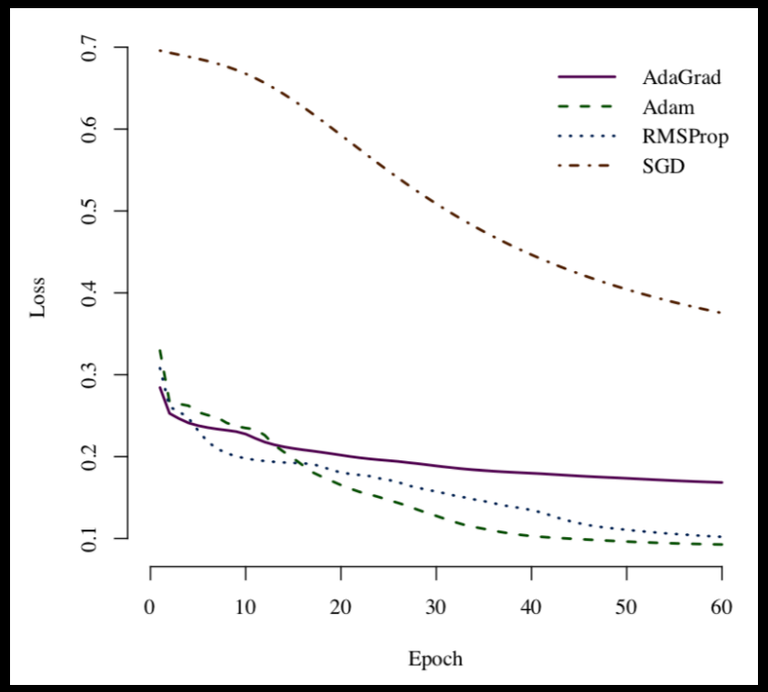
(d) Contractive AE

(e) Denoising AE with sparsity regularization

(f) Robust AE with weight decay

Figure 17: Reconstructing test samples with different AE models. First row of each figure shows test samples, second row shows activations of the encoding layer and third row displays reconstructions. Encoded values range from -1 (black) to 1 (white).

Key Results

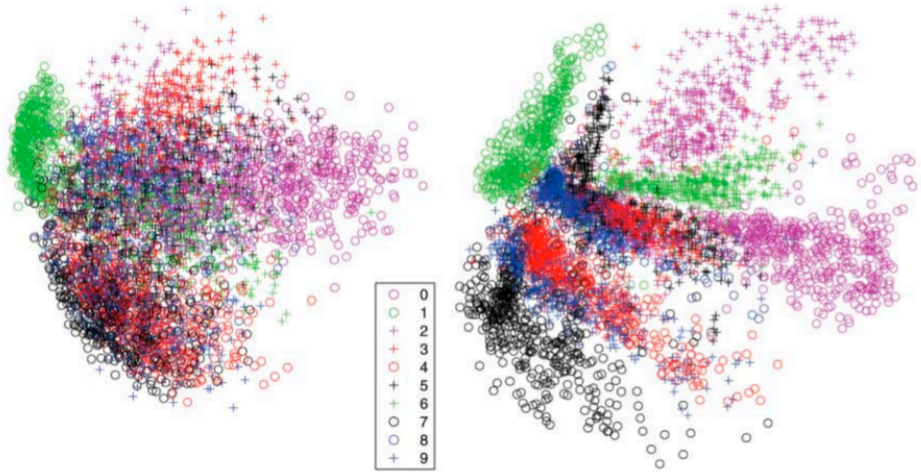

AdaGrad
Adam
RMSProp
SGD
Loss
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50
60
Epoch

Key Results

- Semi-supervised text classification with AEs outperforms fully-supervised methods
- Anomaly detection in spacecraft telemetry with AEs beats PCA and Kernel PCA
- Semantic hashing with AEs beats TF-IDF

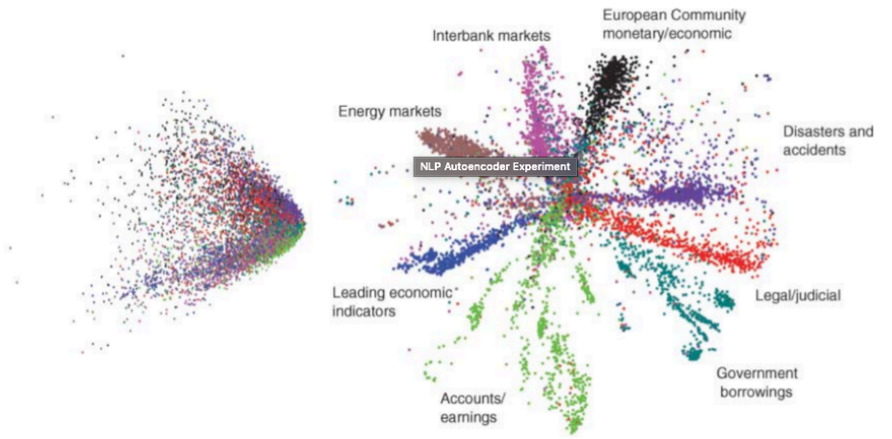
- The paper lists some key results involving autoencoders; here I have reproduced some of the more impressive ones.
- One result finds that semi-supervised text classification using autoencoders outperforms even fully-supervised methods.
- Another result finds that anomaly detection in spacecraft telemetry with autoencoders beats both PCA and Kernel PCA.
- Another results finds that semantic hashing with autoencoders yields better document lookup performance than the common TF-IDF technique.

Key Results



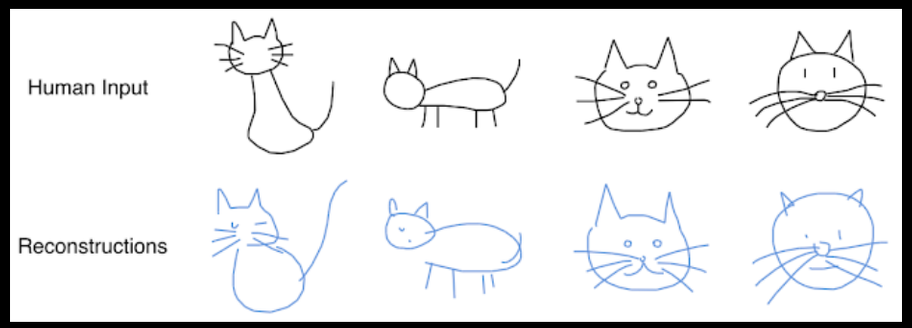
A comparison of separability of 2-dimensional codes generated by an autoencoder (right) and PCA (left) on the MNIST dataset

Key Results

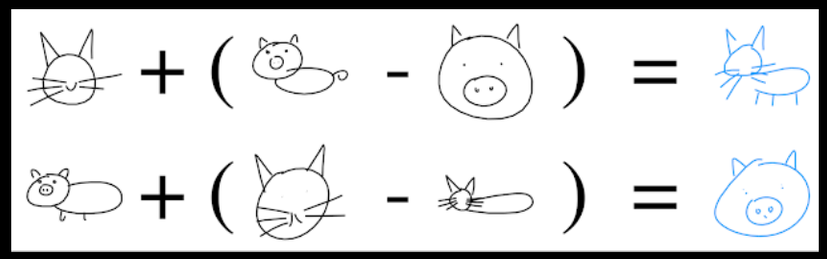


A comparison of separability of 2-dimensional codes generated by an autoencoder (right) and PCA (left) on news stories

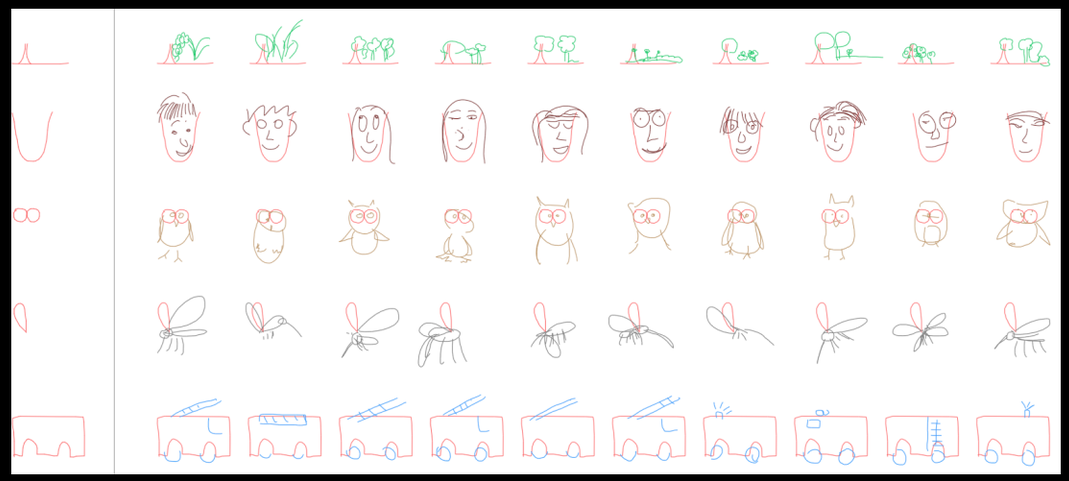
Key Results



Key Results



Key Results



Limitations

- Features are not always interpretable
- Slow-running compared to alternative techniques
- Design is difficult
- Not well justified theoretically

Long-Term Impact

- Yes
  - Unsupervised
  - Can learn very nontrivial structure