Deep Learning Theory and Applications

Autoencoders Part 2

Yale

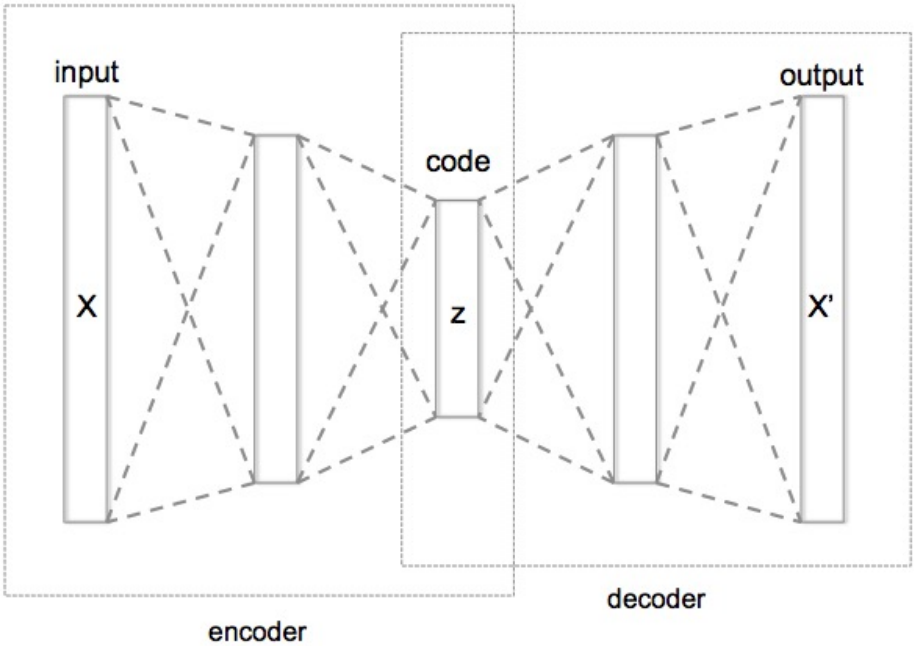
CPSC/AMTH 663

Outline

1. Non-linear dimensionality reduction review
2. Autoencoders
3. Sparse Autoencoders
4. Denoising Autoencoder
5. Contractive Autoencoder
6. SAUCIE
7. AAnet
8. VAE Preview

Autoencoder Applications

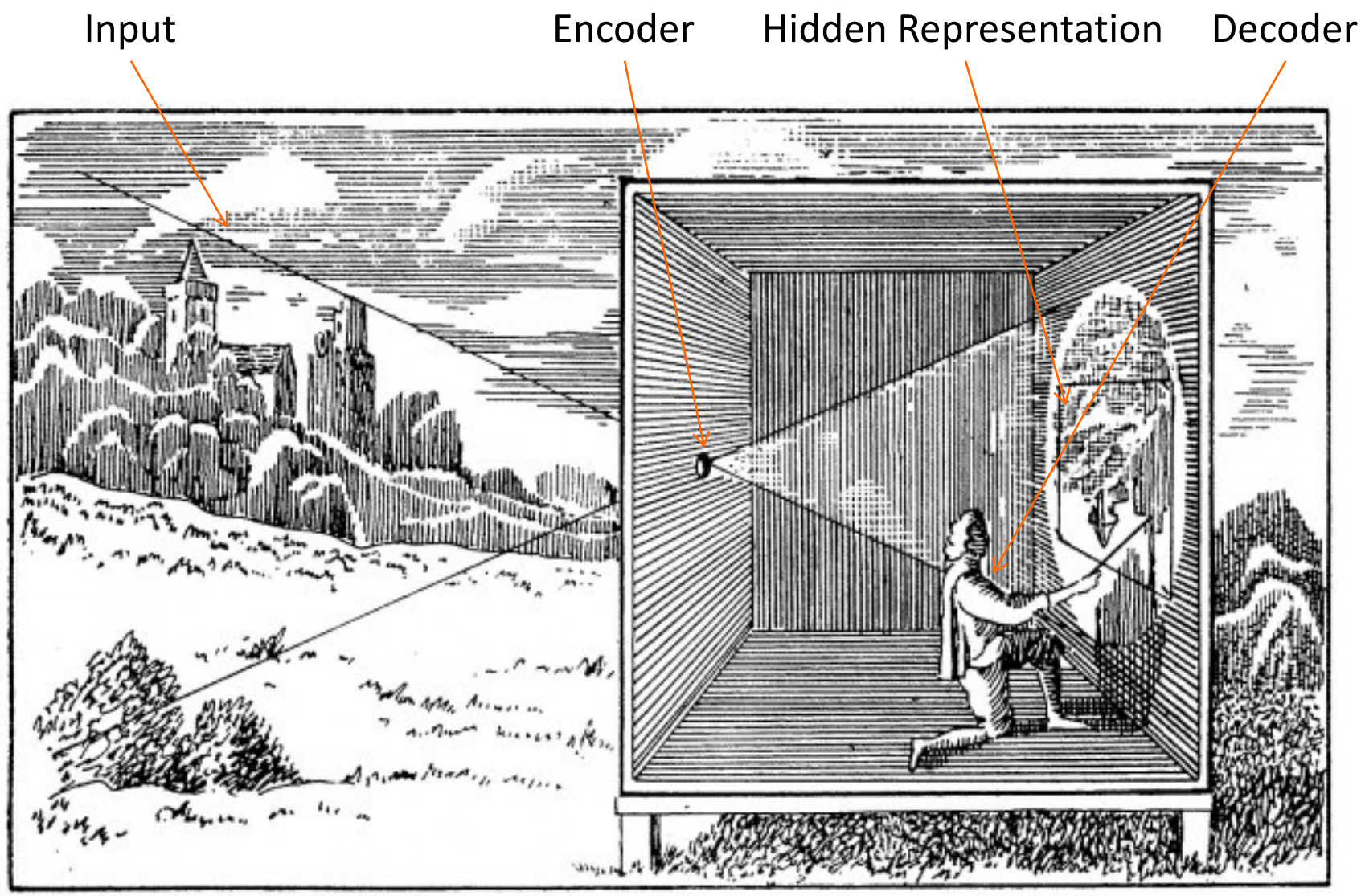
- *Dimensionality reduction*
  - *Including visualization*
- Data denoising
- Also…
  - Feature learning
  - Information retrieval
  - Data compression
  - Data generation



Are Autoencoders useful?

- Can't we just set $g\big(f(x)\big) = x$ everywhere?
  - Not very useful
- Design the AE so it can't learn to copy the input perfectly
  - Restrict the AE to copy only approximately
  - Can prioritize certain aspects of the input
- Examples
  - Restrict the size of the code (i.e. add a bottleneck)
  - Add regularization

Camera Obscura



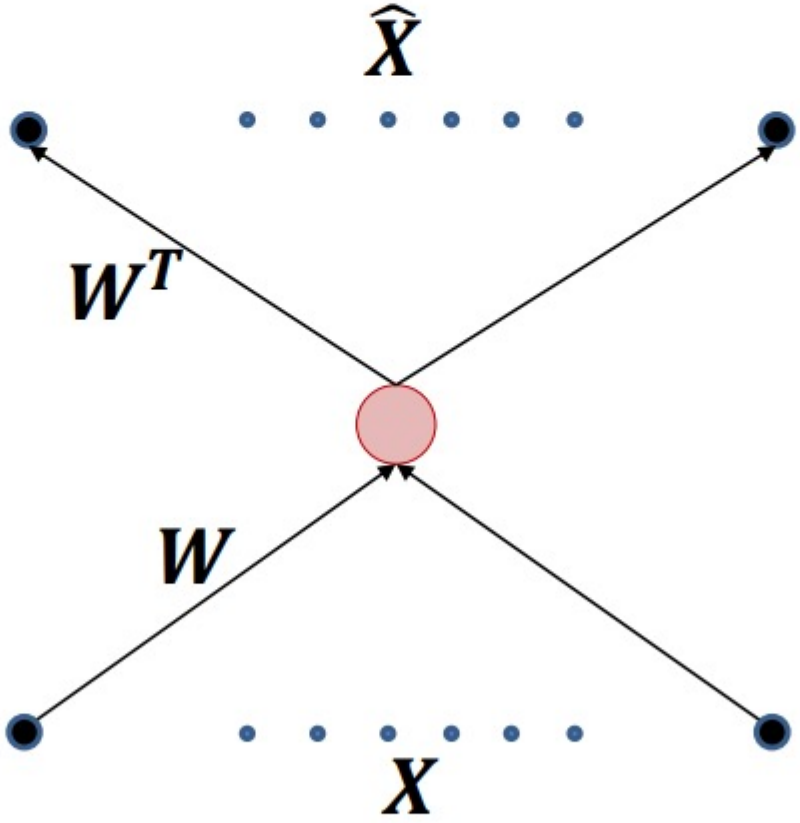
Camera Obscura, the renaissance autoencoder

Undercomplete Autoencoders

- Hope: training the autoencoder will result in z having useful properties
- $\text{Dim}(z) < \text{Dim}(x) \Rightarrow$ ***undercomplete*** autoencoder
  - Forces the AE to capture the most salient features of the training data
  - The AE only approximately copies the input (lossy compression)

Undercomplete Autoencoders

- Special case: Linear decoder, mean squared error loss
- Example: single hidden unit
- What will this learn?

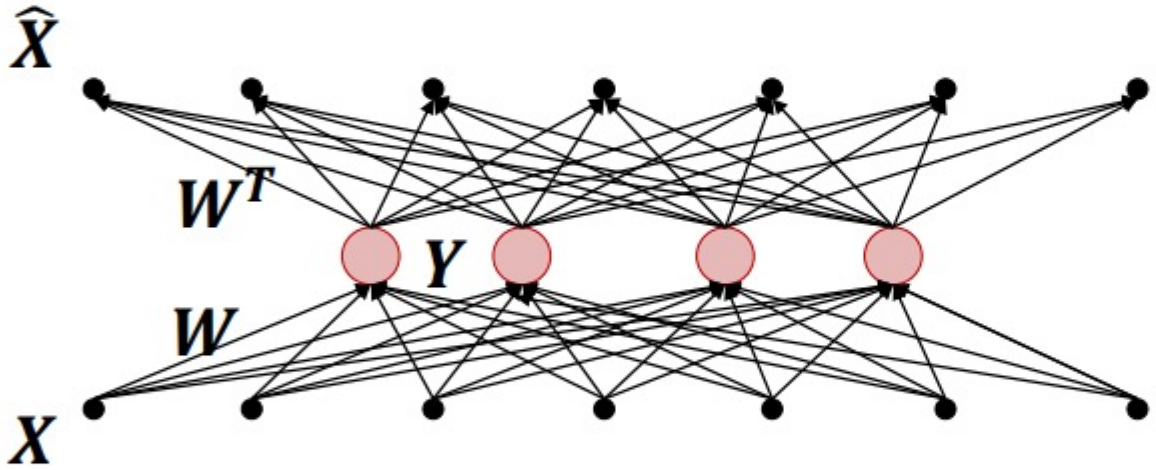


- $\hat{x} = w^T wx$
  - Recall w is a row vector
- Minimize reconstruction error:

$$\hat{w} = \arg\min_{w} \mathbb{E}[\|x - w^T wx\|^2]$$

- Equivalent to PCA!

Undercomplete Autoencoders



- What about more hidden nodes?
- $Y = WX, \hat{X} = W^T Y$
- Find W to minimize $\mathbb{E}[\|X - W^T WX\|^2]$
- Still PCA

Undercomplete Autoencoders

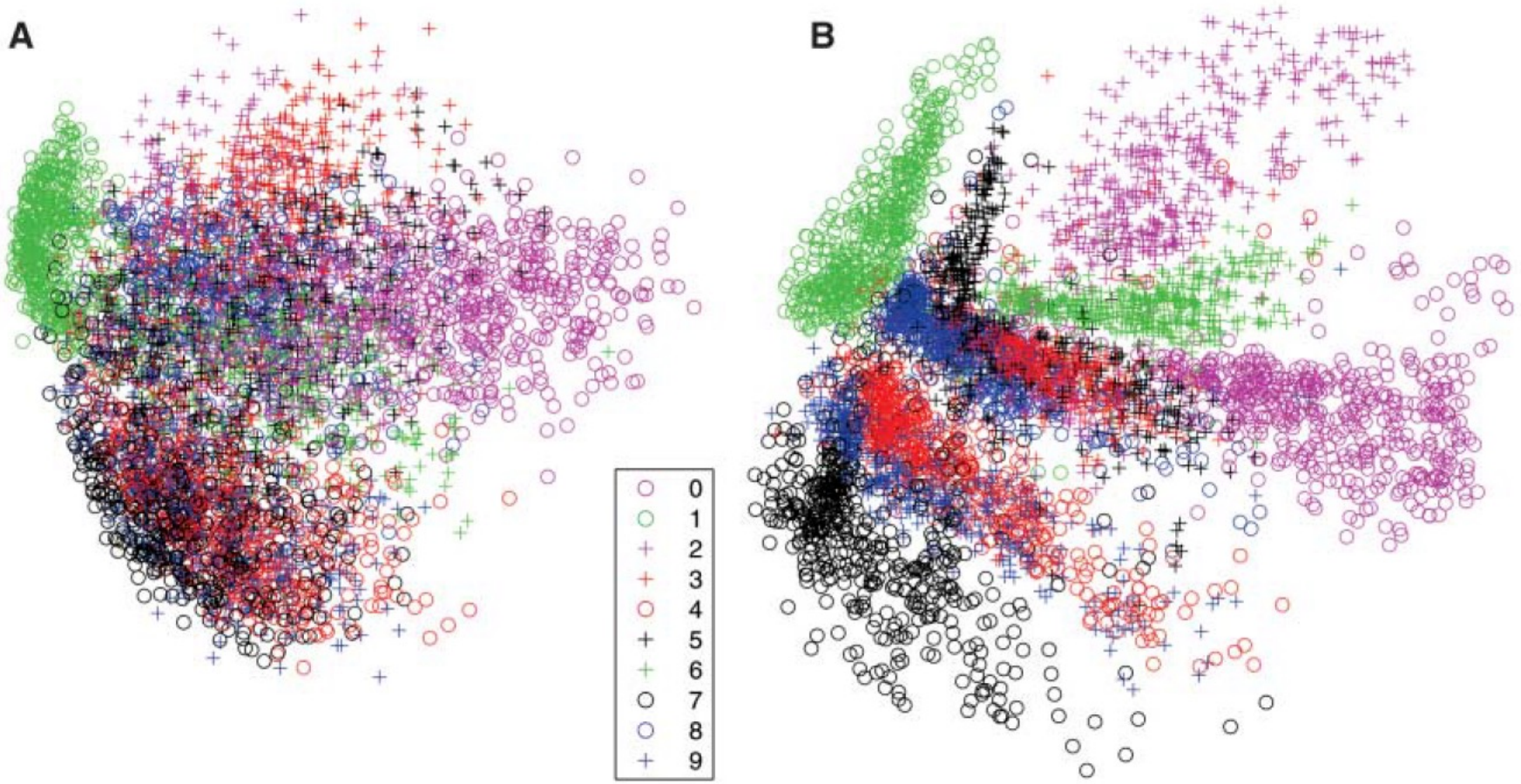
- Special case: Linear decoder, mean squared error loss
  - Undercomplete AE learns to span the same subspace as PCA
  - The AE has learned the principal subspace of the training data
- Nonlinear encoder and decoder functions can give powerful nonlinear generalization of PCA
  - Be careful with capacity
  - Too much capacity ⇒ AE learns to copy the input w/o extracting useful information
- Example: could learn a 1-dimensional code for the entire dataset
  - Map each training example x_i to the integer i
  - Nothing useful is learned in this case

Regularized Autoencoders

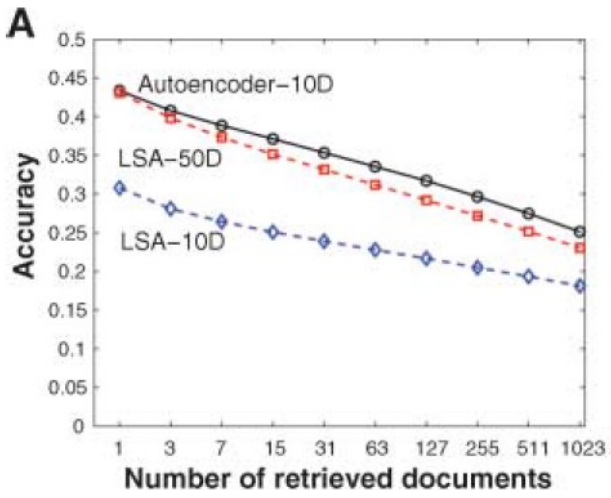
- Undercomplete AEs can fail to learn anything useful if the capacity is too large
- Same problem if the hidden code is dimension equal to or greater than the input dimension
  - Latter case gives an ***overcomplete*** AE
  - Even linear encoder and decoder wouldn't be useful

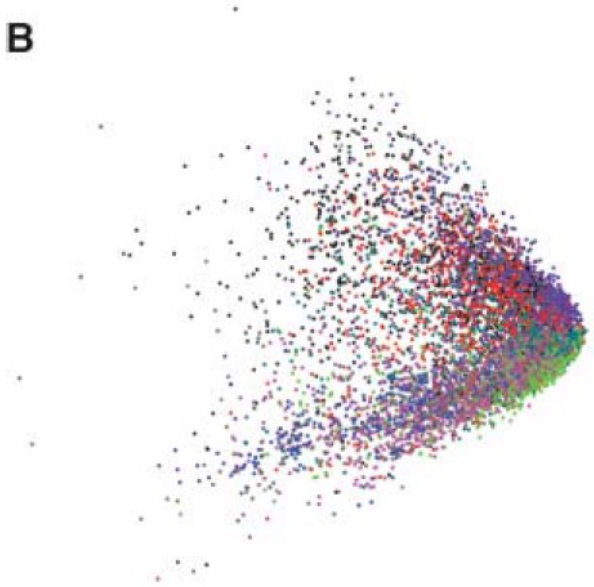
- Regularized AEs modify the loss function to encourage desirable properties
  - E.g., sparsity, robustness to noise
  - Regularized nonlinear and overcomplete AE can still learn something useful about the data distribution

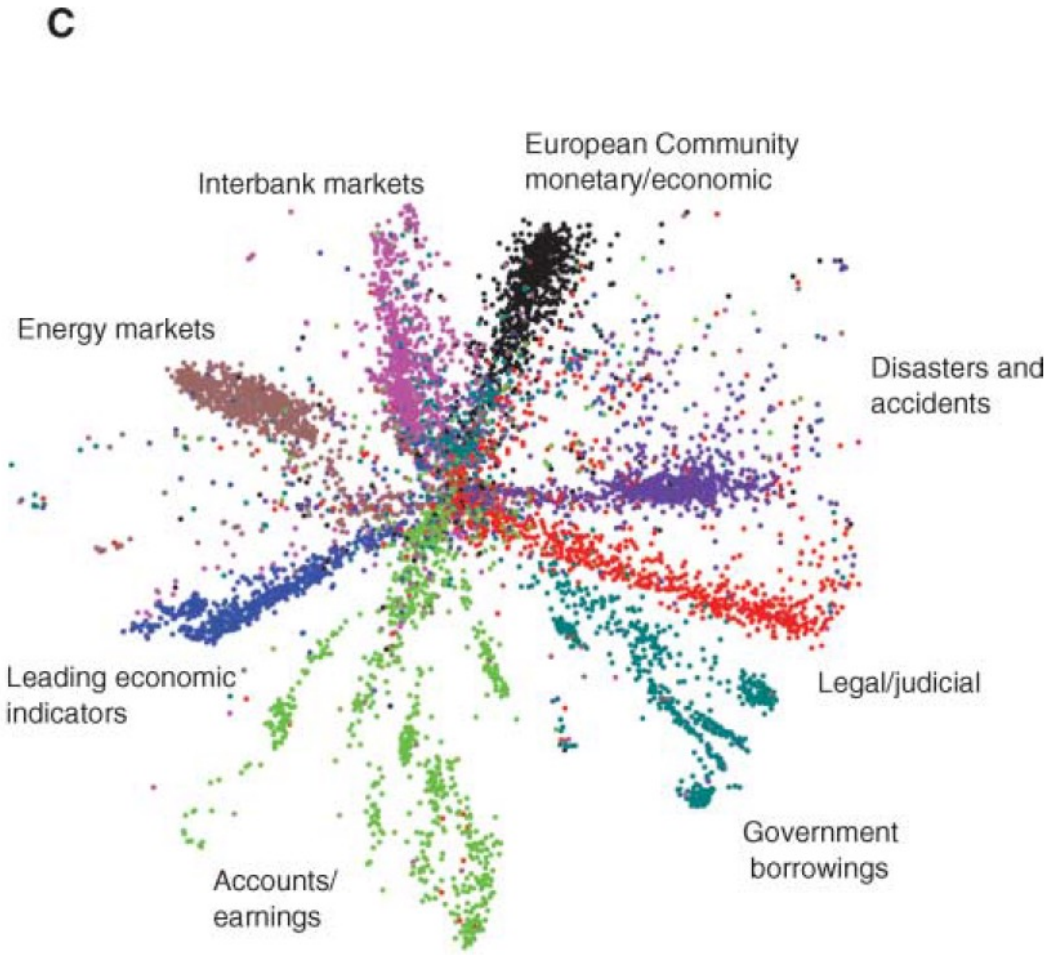
AE MNIST



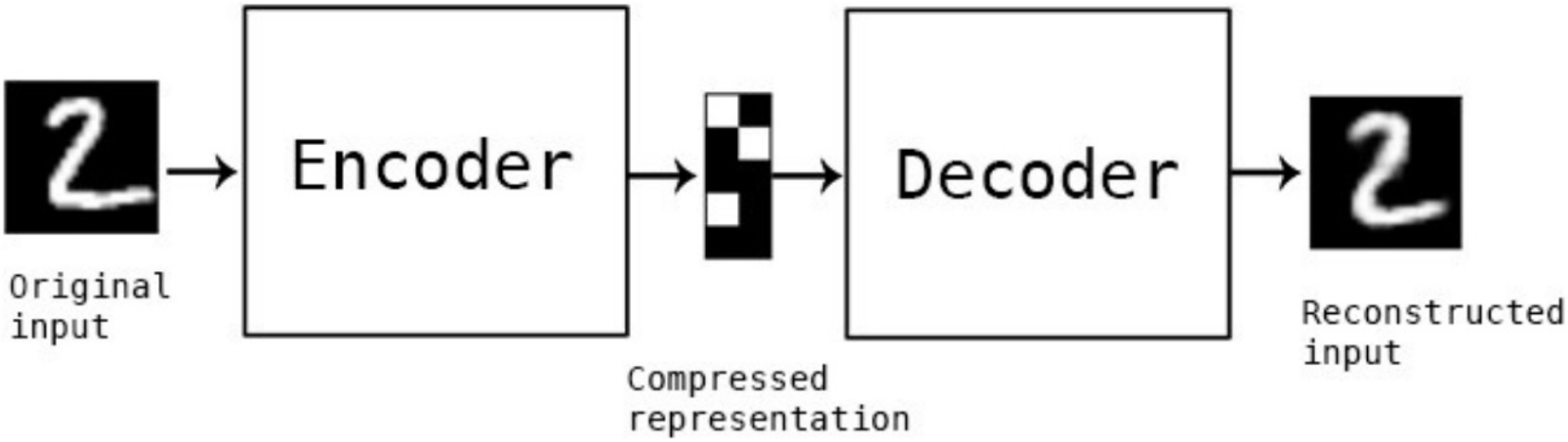
Document Embedding







Autoencoding MNIST



Autoencoding MNIST

- Single hidden layer with 32 nodes, fully connected layers
- Cross-entropy loss per pixel, ReLU activations
- Train for 50 epochs
  - Reaches a stable train/test loss of 0.11
  - Top = original, bottom = reconstructed digits

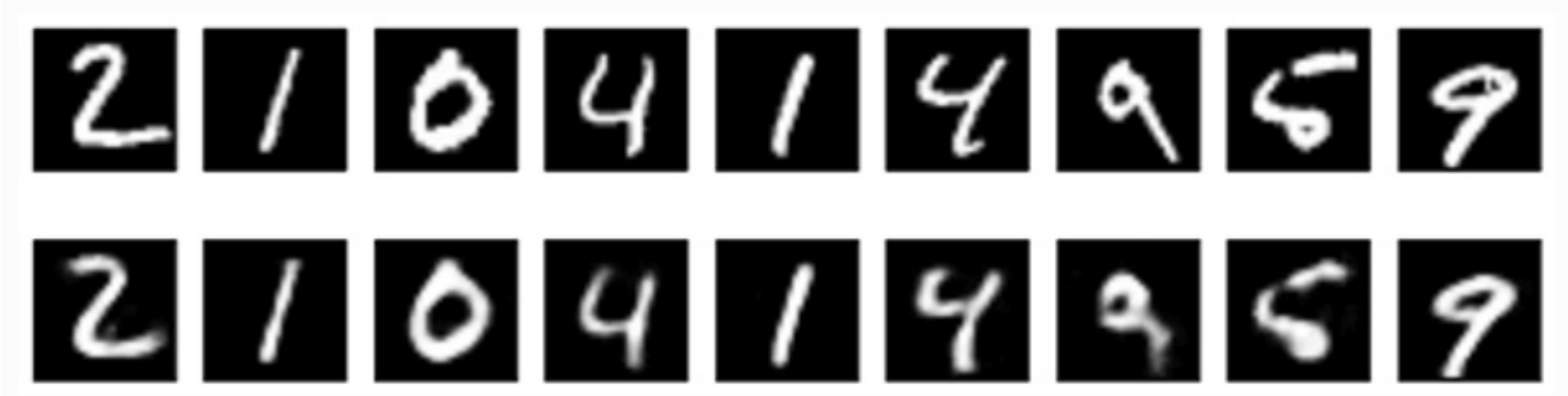
Autoencoding MNIST

- Single hidden layer with 32 nodes, fully connected layers
- Cross-entropy loss per pixel, ReLU activations
- Add L1 sparsity penalty to hidden layer
- Train for 100 epochs
  - Train loss 0.11, test loss 0.10
  - Top = original, bottom = reconstructed digits

- Look the same, but sparser code

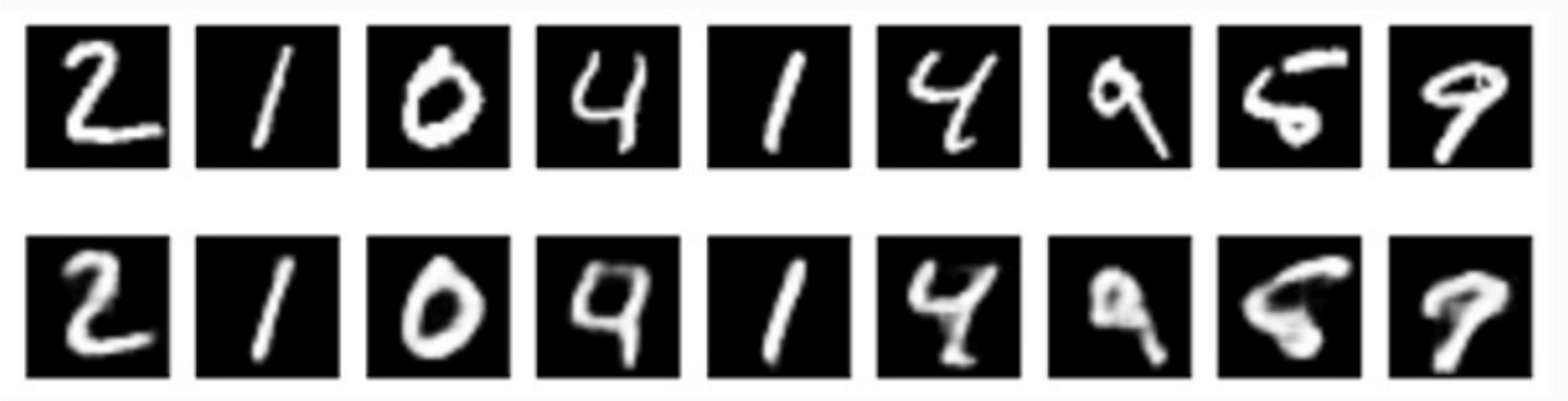
Autoencoding MNIST

- Add two other hidden layers on each side of size 128 & 64
- Cross-entropy loss per pixel, ReLU activations
- Train for 100 epochs
  - Train/test loss ≈ 0.097
  - Top = original, bottom = reconstructed digits



Autoencoding MNIST

- Convolutional autoencoder
- Loss of 0.094
- Top = original, bottom = reconstructed digits



Sparse Autoencoders

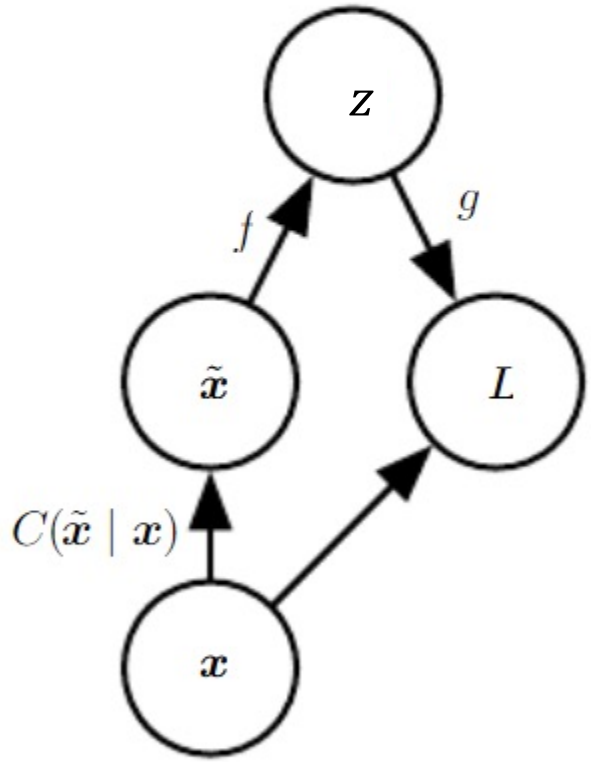
- Commonly used for feature learning for another task
- Include a sparsity penalty $\Omega(z)$ on the code layer z:

$$L\left(x, g\left(f(x)\right)\right) + \Omega(z)$$

  - Recall, encoder output $z = f(x)$, decoder output $g(z)$
- Example: L1 penalty
  - $\Omega(z) = \lambda \sum_i |z_i|$
- To achieve actual zeros, use ReLU activation prior to the code layer

Denoising Autoencoders

- Traditional AE minimizes $L\left(x, g\left(f\left(x\right)\right)\right)$
- ***Denoising autoencoder*** minimizes $L\left(x, g\left(f\left(\tilde{x}\right)\right)\right)$
  - $\tilde{x}$ is a corrupted version of x
  - Denoising AEs learn to undo this corruption
- Corruption process $C(\tilde{x}|x)$ modeled by conditional distribution
- The AE learns a reconstruction distribution $p_{reconstruct}(x|\tilde{x})$ from training pairs $(x, \tilde{x})$



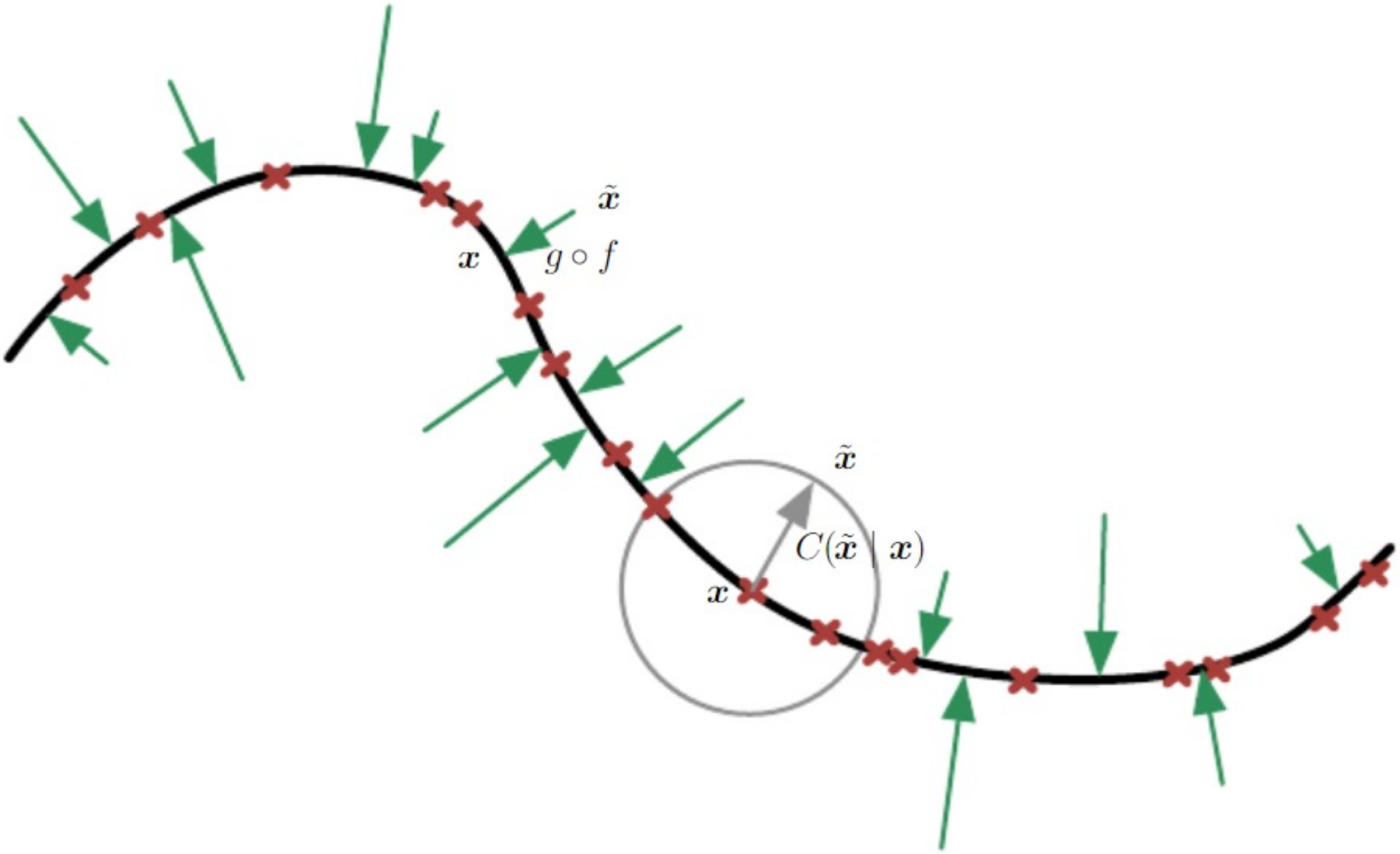
Denoising Autoencoders

The process:

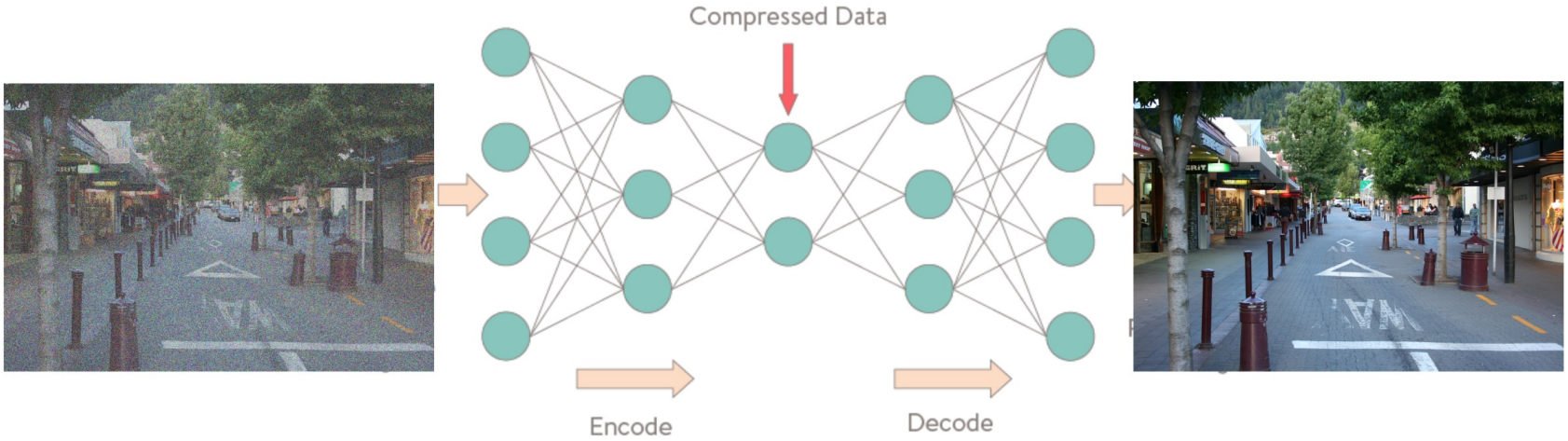
1. Sample training example x from training data
2. Sample corrupted version $\tilde{x}$ from $C(\tilde{x}|x)$
3. Use $(x, \tilde{x})$ as a training example for estimating $p_{reconstruct}(x|\tilde{x}) = p_{decoder}(x|z)$
   - $z = f(\tilde{x})$ the encoder output, $p_{decoder}$ defined by decoder $g(z)$

- Can perform SGD on the negative log-likelihood $-\log p_{decoder}(x|z)$
  - Equivalent to doing SGD on
    $$-\mathbb{E}_{x\sim p_{data}(x)}\mathbb{E}_{\tilde{x}\sim C(\tilde{x}|x)}\log p_{decoder}(x|z = f(\tilde{x}))$$
    - p_{data} is the training distribution

Denoising Autoencoders



Application to Image Denoising



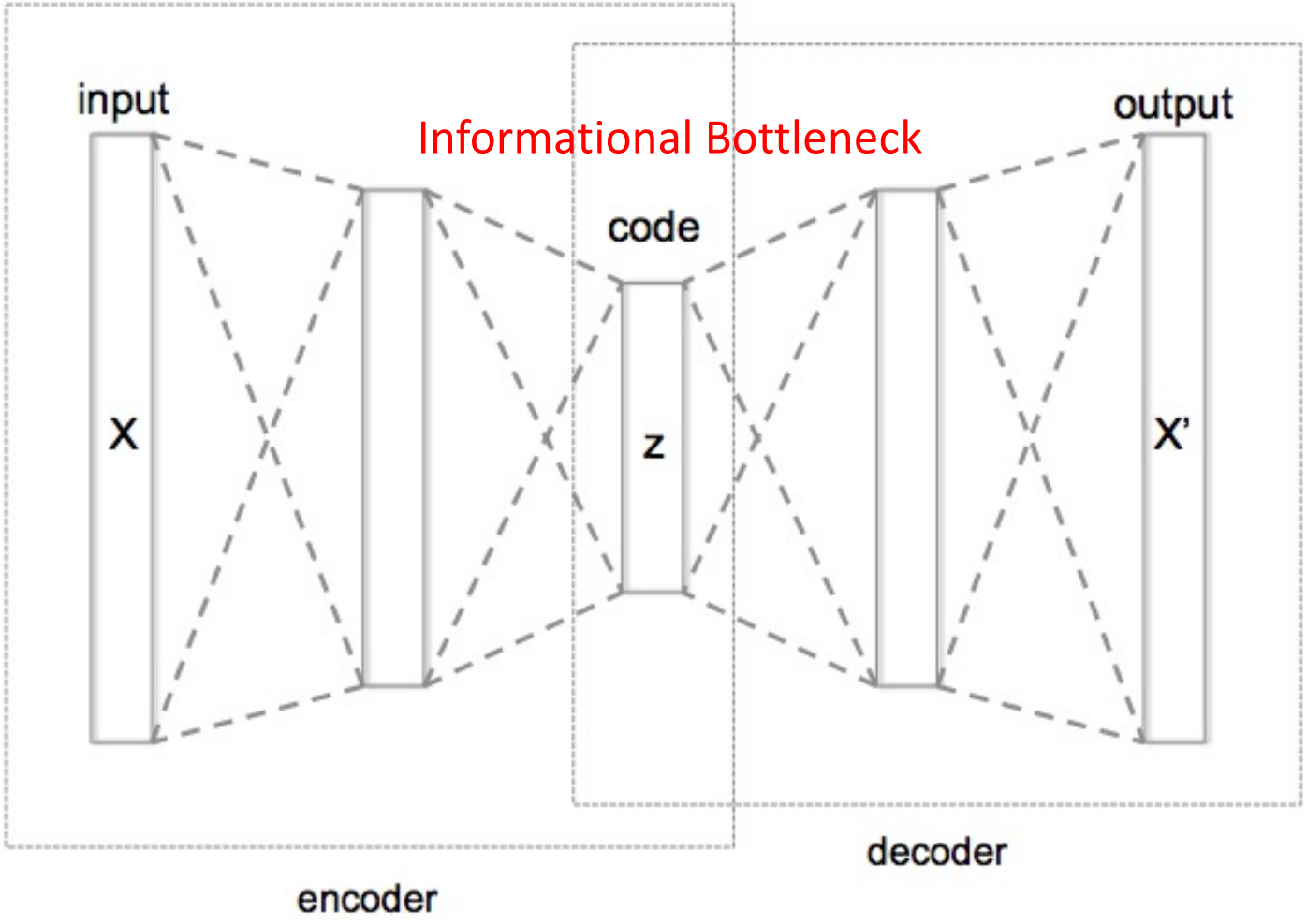
Autoencoding MNIST

- Denoising autoencoder
  - Add Gaussian noise
- Top = noisy original, bottom = reconstructed digits

Further reading

- Goodfellow et al., chapter 14
- Hinton and Salakhutdinov Science 2006
- Coifman and Lafon Applied and Computational Harmonic Analysis 2006
- Belkin & Niyogi, Neural Computation 2003
- Alain & Bengio, JMLR 2014

Autoencoder



[Hinton, Salakhutdinov, Science 2006]

Autoencoder (AE)

- Neural network trained to copy its input x to its output
- Hidden layer z describes a ***code*** to represent the input
- Two parts
  - Encoder: $\mathrm{z} = f(x)$
  - Decoder: $x' = g(z)$
- Goal: minimize $L\left(x, g\left(f(x)\right)\right)$
  - L penalizes $g\left(f(x)\right)$ for being dissimilar from x
  - Example: mean squared error

Autoencoder and Identity

- Autoencoders could learn the identity without proper regularization/design
- Design the AE so it cannot learn to copy the input perfectly
  - Restrict the AE to copy only approximately
  - Can prioritize certain aspects of the input
- Examples
  - Restrict the size of the code (i.e. add a bottleneck)
  - Add regularization
  - Or alter penalty function to learn invariances

The Autoencoder compromise

Two opposing forces:

1. Learn a representation z of x s.t. x can be approximately recovered from z
   - x is drawn from the training data ⇒ the AE learns to reconstruct only probable inputs
2. Satisfy the constraint or regularization penalty
   - E.g. architectural constraint or regularization term
   - Less sensitivity to the input is preferred

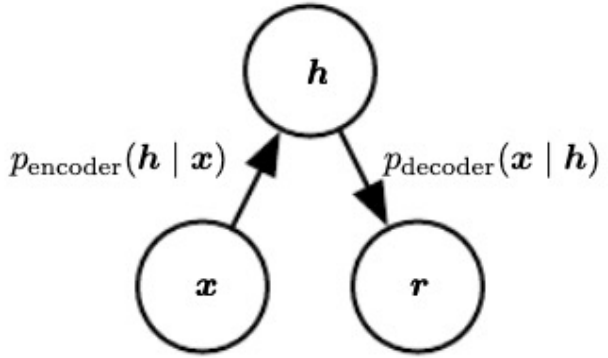
- Neither force alone is useful
- Together they force z to capture information about the structure of the data-generating distribution

Restricting bottleneck

- $\text{Dim}(z) < \text{Dim}(x) \Rightarrow$ ***undercomplete*** autoencoder
  - Forces the AE to capture the most salient features of the training data
  - The AE only approximately copies the input (lossy compression)
- Performs non-linear dimensionality reduction
- Restrict to 2-3 nodes for visualization
  - Like tSNE/PHATE/diffusion maps etc

Stochastic view

- Autoencoders can be thought of as latent variable models if the encoder is stochastic



- Then the autoencoder training actually corresponds to minimizing negative log likelihood

$$-\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$$

Regularizations

- In this view regularizations can be thought of as adding a prior to the log likelihood estimate
- This corresponds to maximum-a-priori inference

$$\boldsymbol{\theta}_{\mathrm{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

- Here the regularization is providing something akin to

$$\log p(\boldsymbol{\theta})$$

- However not all regularizations are log of the likelihood term and can take the form of more generalized penalties

Regularizations

- Regularized AEs modify the loss function to encourage desirable properties
- Examples include:
  - Sparsity
  - Robustness to noise
  - Restrictions on gradient sensitivity

Sparse Autoencoders

- Commonly used for feature learning for another task
- Include a sparsity penalty $\Omega(z)$ on the code layer z:

$$L\left(x, g\left(f(x)\right)\right) + \Omega(z)$$

  - Recall, encoder output $z = f(x)$, decoder output $g(z)$
- This can force the autoencoder to select features and can be useful for further classification tasks

Sparse Autoencoders

- Sparse autoencoders can also be interpreted as a maximum likelihood generative model with latent variables *h*

$$\log p_{\text{model}}(\boldsymbol{x}) = \log \sum_{\boldsymbol{h}} p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x}).$$

- Joint probability

$$\log p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x}) = \log p_{\text{model}}(\boldsymbol{h}) + \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$$

- Example: L1 penalty

$$p_{\text{model}}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}$$

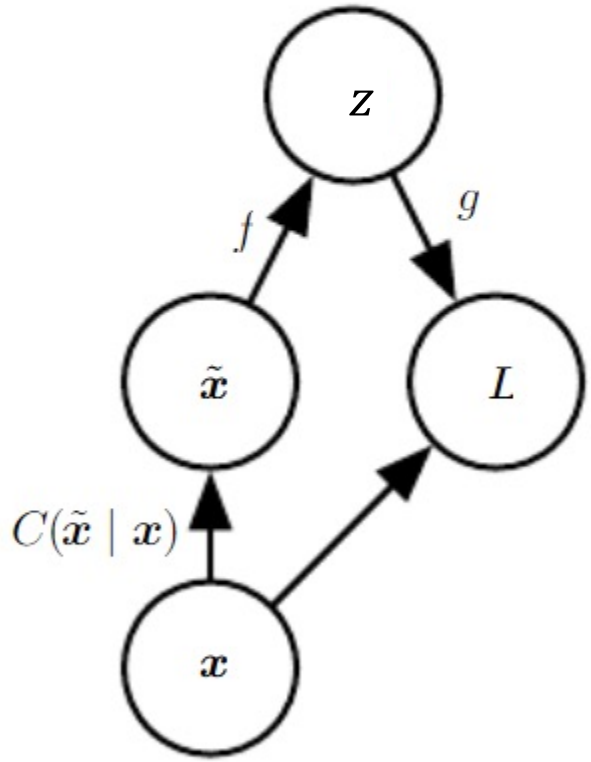
  - Equivalent to:

$$\Omega(\boldsymbol{h}) = \lambda \sum_i |h_i|,$$

- Penalty finds latent variables that explain output variables, i.e., has a connection to generative models

Denoising Autoencoders

- Traditional AE minimizes $L\left(x, g\left(f\left(x\right)\right)\right)$
- ***Denoising autoencoder*** minimizes $L\left(x, g\left(f\left(\tilde{x}\right)\right)\right)$
  - $\tilde{x}$ is a corrupted version of x
  - Denoising AEs learn to undo this corruption
- Corruption process $C(\tilde{x}|x)$ modeled by conditional distribution
- The AE learns a reconstruction distribution $p_{reconstruct}(x|\tilde{x})$ from training pairs $(x, \tilde{x})$



Denoising Autoencoders

The process:

1. Sample training example x from training data
2. Sample corrupted version $\tilde{x}$ from $C(\tilde{x}|x)$
3. Use $(x, \tilde{x})$ as a training example for estimating $p_{reconstruct}(x|\tilde{x}) = p_{decoder}(x|z)$
   - $z = f(\tilde{x})$ the encoder output, $p_{decoder}$ defined by decoder $g(z)$

- Can perform SGD on the negative log-likelihood $-\log p_{decoder}(x|z)$
  - Equivalent to doing SGD on
    $$-\mathbb{E}_{x\sim p_{data}(x)}\mathbb{E}_{\tilde{x}\sim C(\tilde{x}|x)}\log p_{decoder}(x|z = f(\tilde{x}))$$
    - p_{data} is the training distribution

Denoising Autoencoder

Add noise to the input while training and teach the autoencoder to remove noise
[Bengio ‘09]

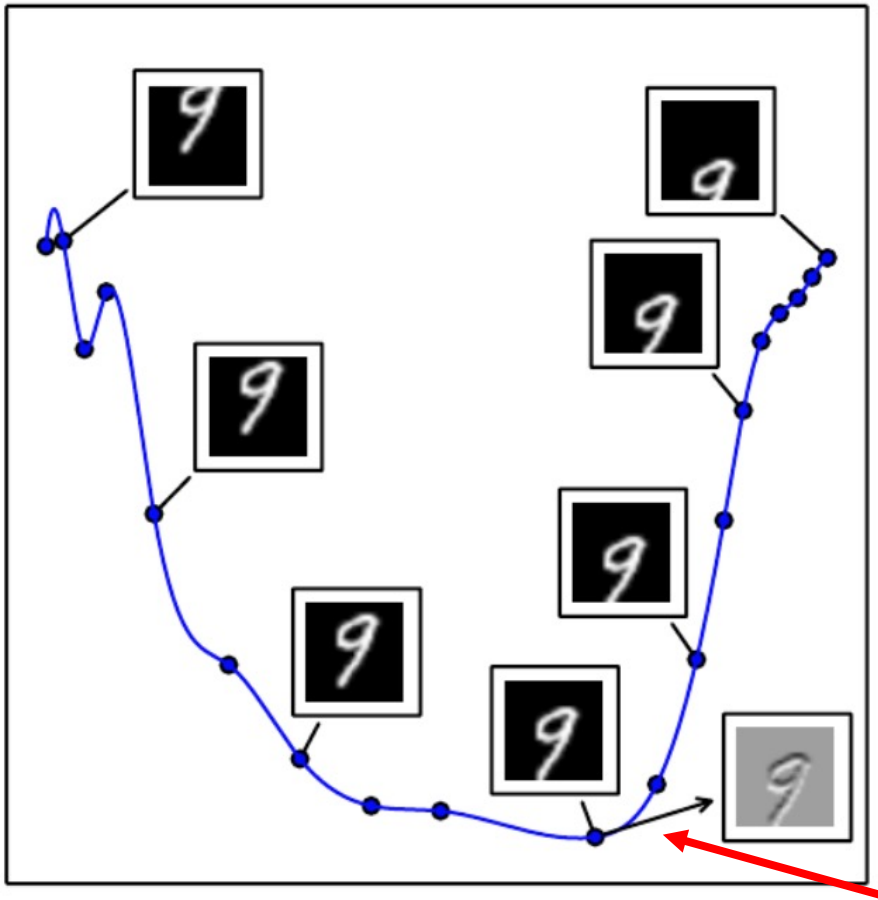
Binary Codes

Information retrieval: find entries in a database that resemble a query entry

- Train an AE to produce a low-dimensional *binary* code
  - Can store database entries in a hash table
  - Return all entries with the same binary code
- Do this with saturated sigmoids on the final layer of the encoder
  - Add additive noise to the sigmoid nonlinearity during training
  - The AE fights the noise until saturation occurs
  - Code must be binarized
- Can enhance binarization (preview: SAUCIE)

Manifold learning

- Tangent plane: plane indicating the local directions of variation allowed on the manifold



Contractive Autoencoders [Rifai 2011]

- Add a regularizer on the code $z = f(x)$
  - Encourages the derivatives of f to be small:

$$\Omega(z) = \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2$$

  - $\Omega(z)$ is the squared Frobenius norm (sum of squared elements) of the (Jacobian) matrix of partial derivatives
- Connection to denoising AEs
  - For small Gaussian noise, the denoising reconstruction error is equivalent to a contractive penalty on $g\big(f(x)\big)$
  - I.e., denoising AEs make the reconstruction resist small, finite perturbations of the input
  - Contractive AEs make the reconstruction resist infinitesimal perturbations of the input

Contractive Autoencoders

- Why the name?
- A contractive AE maps a neighborhood of input points to a smaller neighborhood of output points
  - It contracts the input neighborhood to a smaller output neighborhood
- But this contraction is only local
  - All perturbations of x are mapped near to $f(x)$
- Contraction may not occur globally
  - Possible to map two points x and x' to points $f(x)$ and $f(x')$ that are farther apart than the original points

Contractive Autoencoders

- Why the name?
- A more formal approach:
- Can view the Jacobian matrix J at a point x as approximating $f(x)$ as a linear operator
- A linear operator is contractive if $\|Jx\| \leq 1$ for all x s.t. $\|x\| = 1$
  - I.e. J is contractive if it shrinks the unit sphere
  - Contractive AEs penalize the Frobenius norm of a local linear approximation of $f(x)$ at every training point x to encourage a contraction

Contractive Autoencoders

The two forces in a contractive AE:

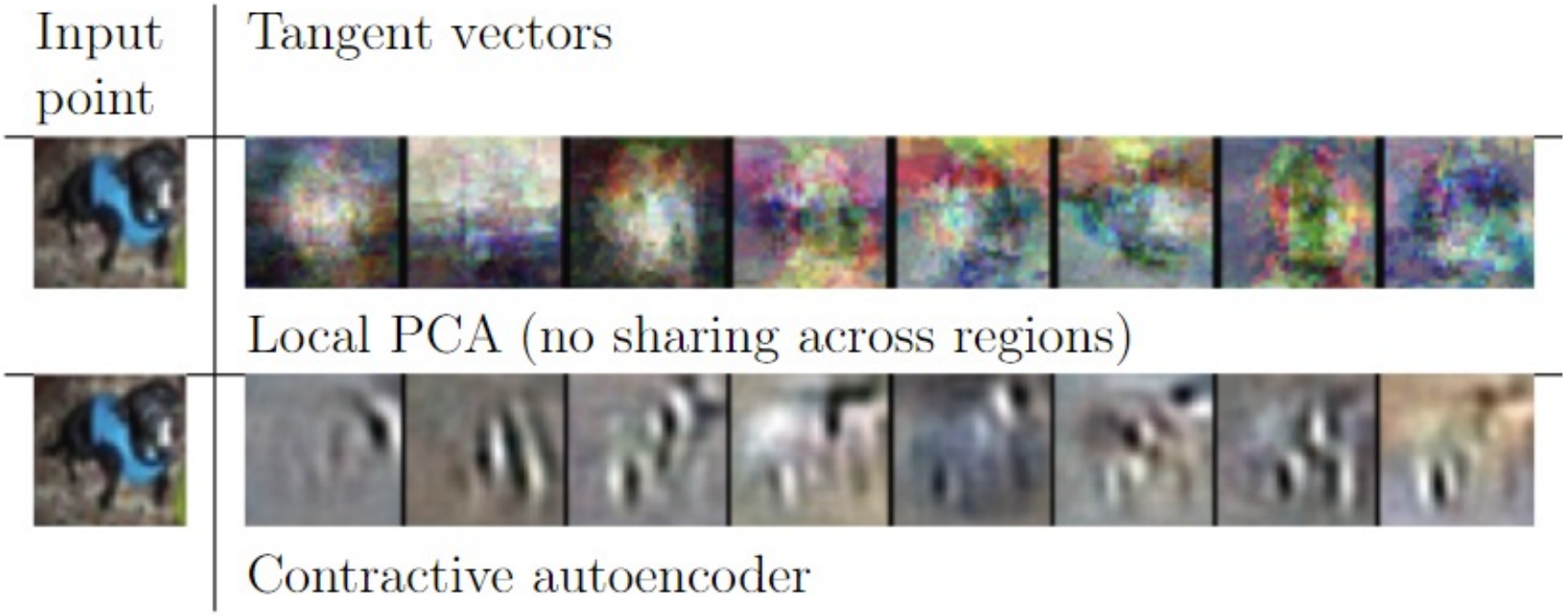
- Reconstruction error and the contractive penalty $\Omega(z)$
  - Reconstruction error alone would encourage the identity function
  - Contractive penalty alone would encourage constant features wrt x
- The compromise yields an AE with mostly tiny derivatives $\frac{\partial f(x)}{\partial x}$
  - Only a small number of directions may have significant derivatives

Contractive Autoencoders

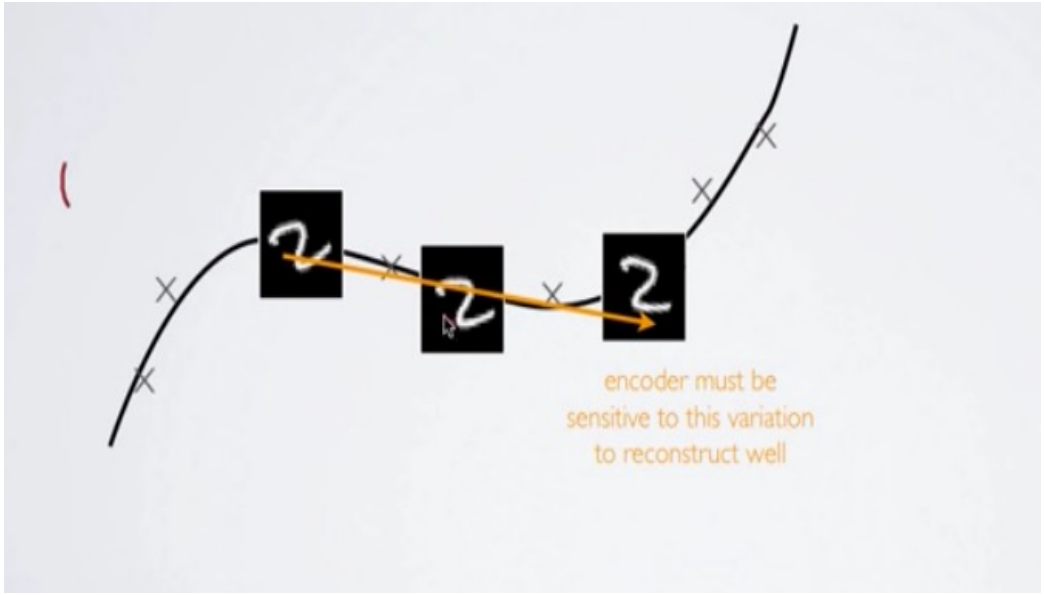
- Goal of a contractive AE is to learn the manifold structure
- Directions x with large Jx rapidly change z
  - Likely to approximate the directions (i.e. tangent planes) of the manifold
- Experimentally, training contractive AE results in most singular values of J being less than 1 (i.e. contractive)
- Directions with the largest singular values interpreted as the tangent directions

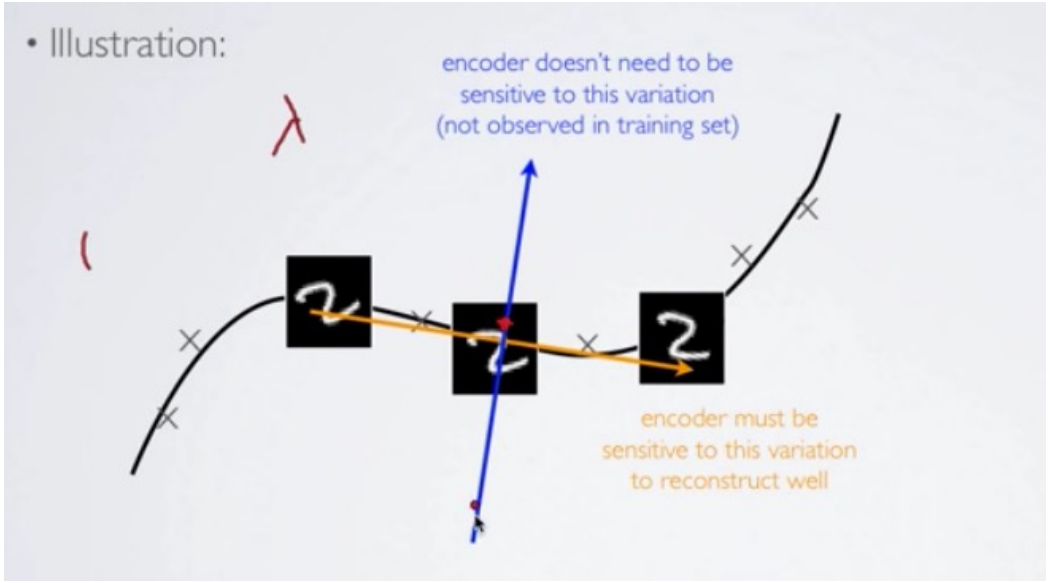
Contractive Autoencoders

- Experimentally obtained tangent vectors
  - Correspond to meaningful transformations



Contractive Autoencoders





Contractive Autoencoders

- Practical issue: deep contractive AEs expensive to compute
- One strategy:
  - Separately train a series of single-layer autoencoders to reconstruct the previous autoencoder's hidden layer
  - Compose these together to form a deep AE
- Practical issue: Can obtain useless results
  - E.g., encoder multiply by ϵ and decoder divide by ϵ
    - Driving ϵ to zero satisfies the contractive penalty
- Strategy: tie the weights of f and g together
  - E.g., set the weight matrix of g to be the transpose of the weight matrix of f

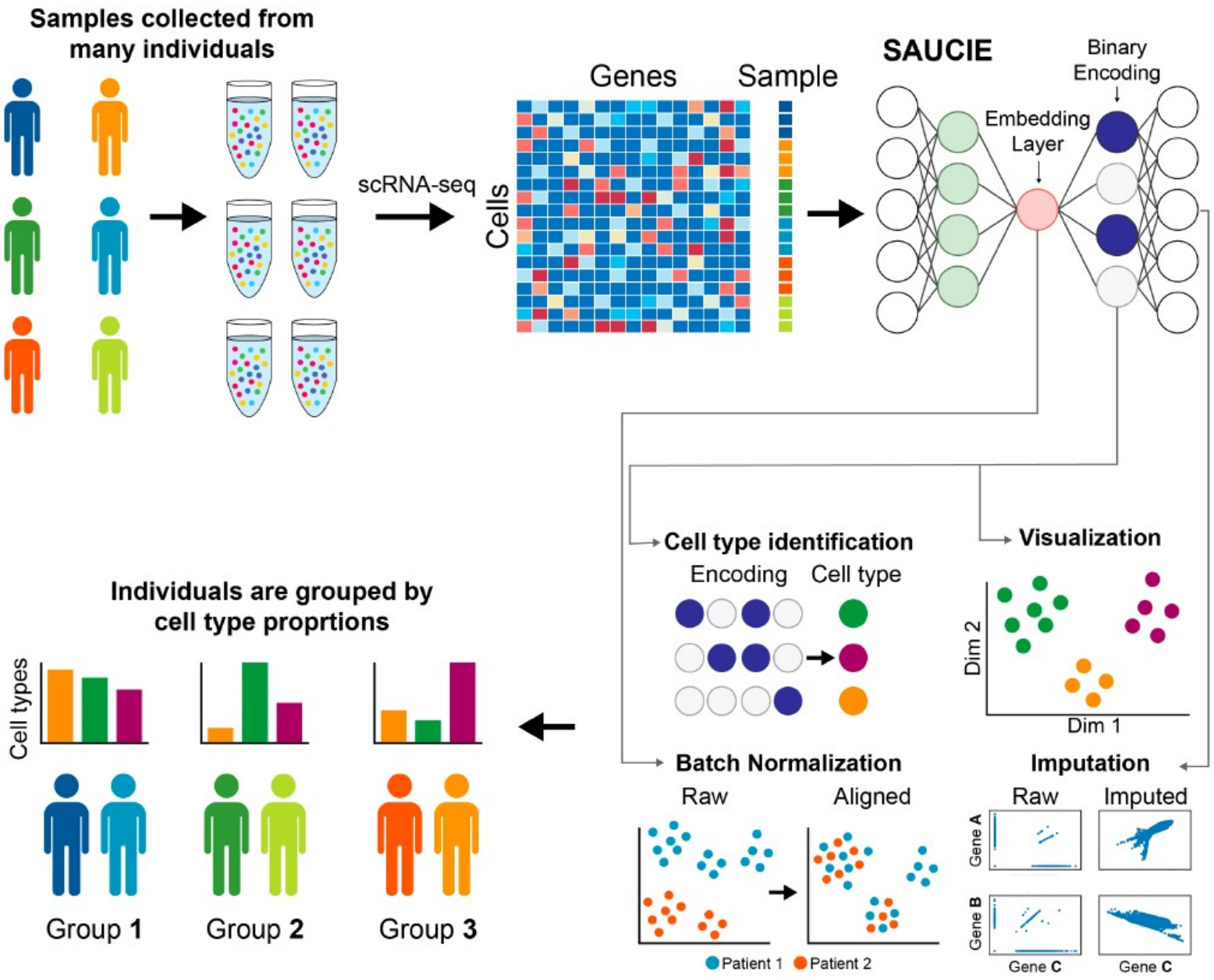
SAUCIE

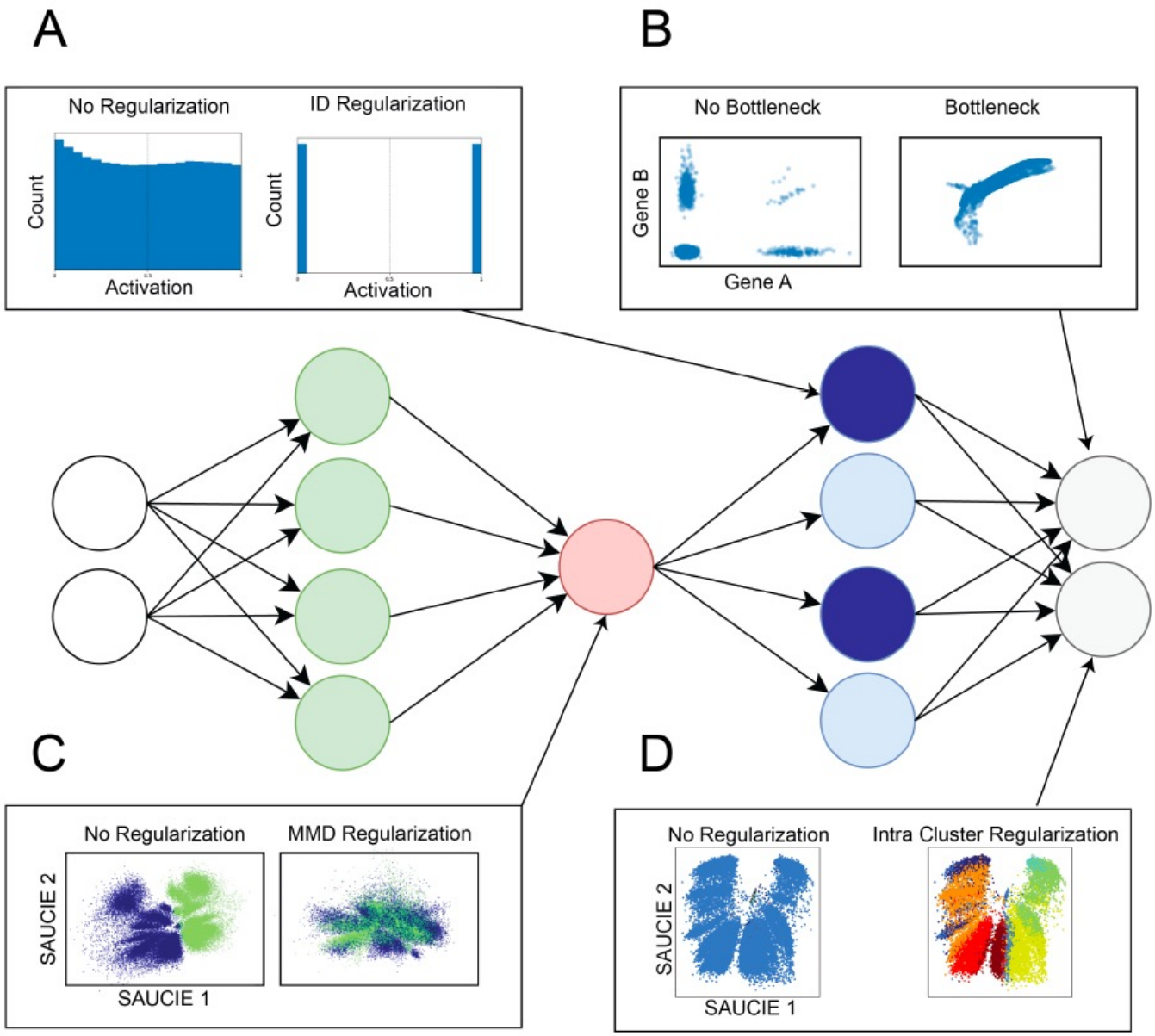
Sparse Autoencoder for Clustering Imputation and Embedding

(Amodio, van Dijk, Srinivasan et al. 2019, Nature Methods)

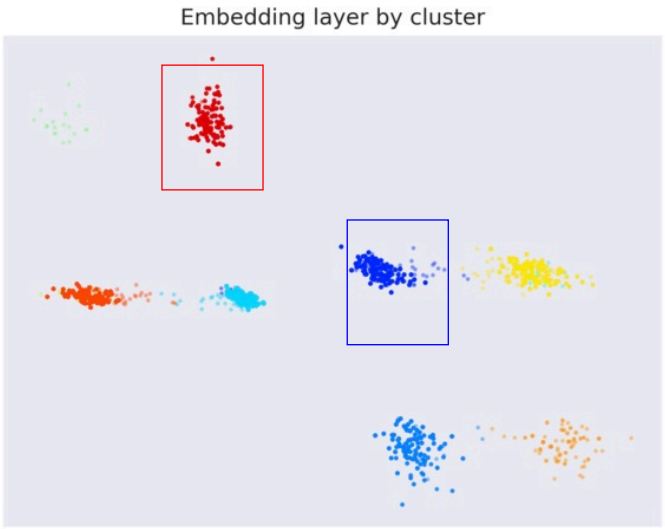
Main idea: **Let a neural network merge and find emergent patterns in multisample data**

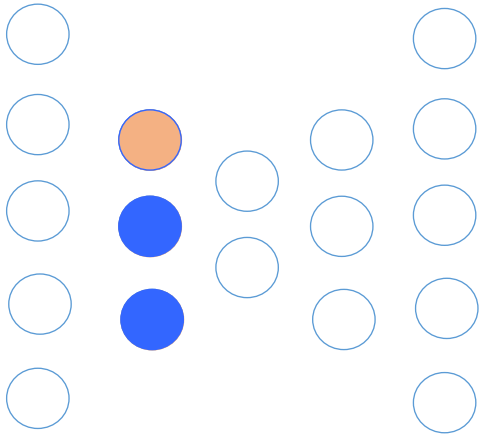
Matthew Amodio
David van Dijk
Krishnan Srinivasan
Kevin Moon
Guy Wolf


Samples collected from many individuals
scRNA-seq
Genes
Sample
Cells
SAUCIE
Binary Encoding
Embedding Layer
Cell type identification
Encoding
Cell type
Visualization
Dim 2
Dim 1
Batch Normalization
Raw
Aligned
Patient 1
Patient 2
Imputation
Raw
Imputed
Gene A
Gene B
Gene C
Gene C
Individuals are grouped by cell type proprtions
Cell types
Group 1
Group 2
Group 3


A
No Regularization
ID Regularization
Count
Activation
Count
Activation
B
No Bottleneck
Bottleneck
Gene B
Gene A
C
No Regularization
MMD Regularization
SAUCIE 2
SAUCIE 1
D
No Regularization
Intra Cluster Regularization
SAUCIE 2
SAUCIE 1

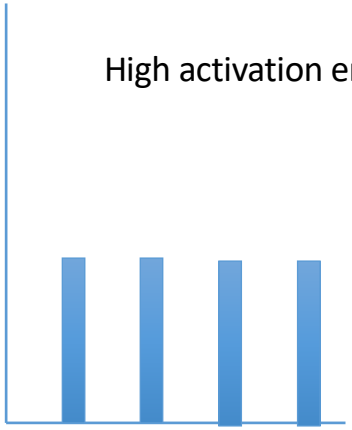
Representation for Clustering

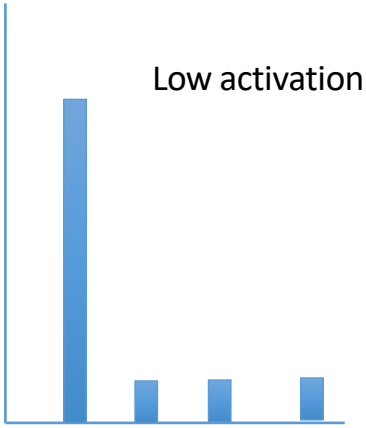




We want: codes for subspaces
Binary activation, good spacing

Clustering: Information Dimension Regularization





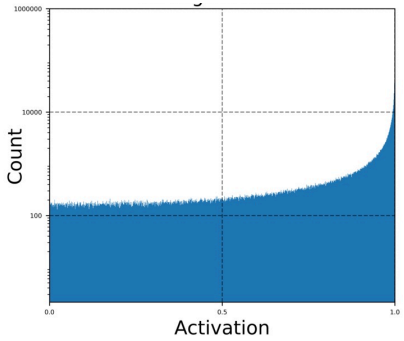
Penalty:

$$-\sum a_i \log(a_i)$$

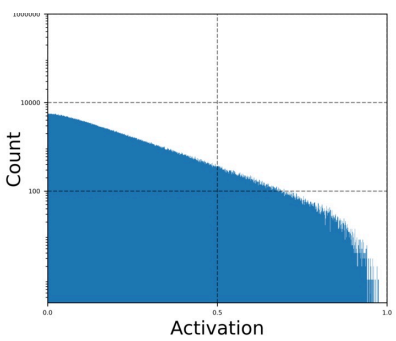
$$a_i = \frac{A_i}{\sum A_i}$$

Information Dimension (ID) Regularization

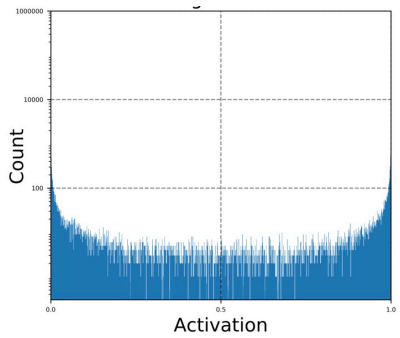
No regularization



L1 regularization

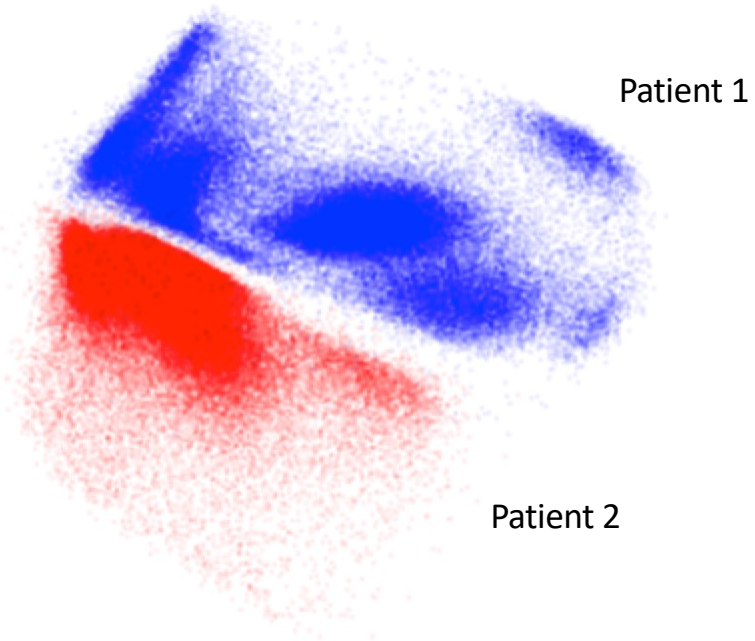


ID regularization

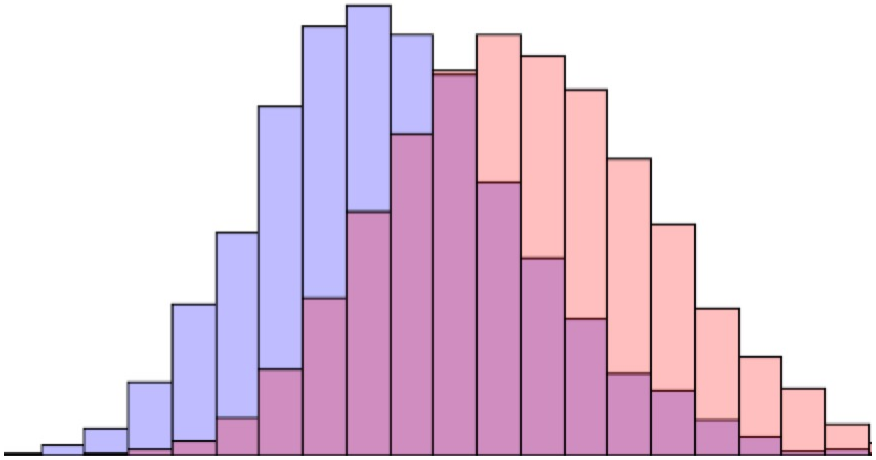


ID regularization enforces a binary-like code

Batch Effects: Manifold Alignment

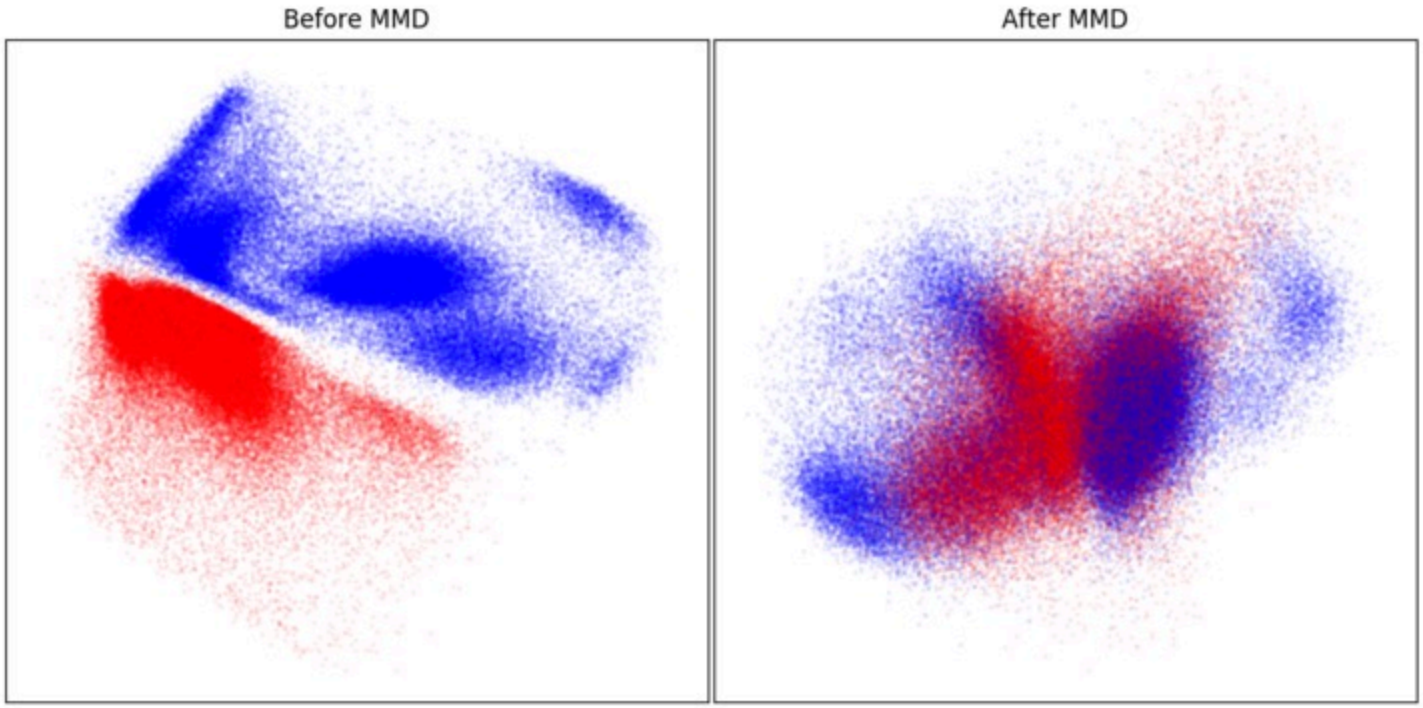


Mean Maximal Discrepancy

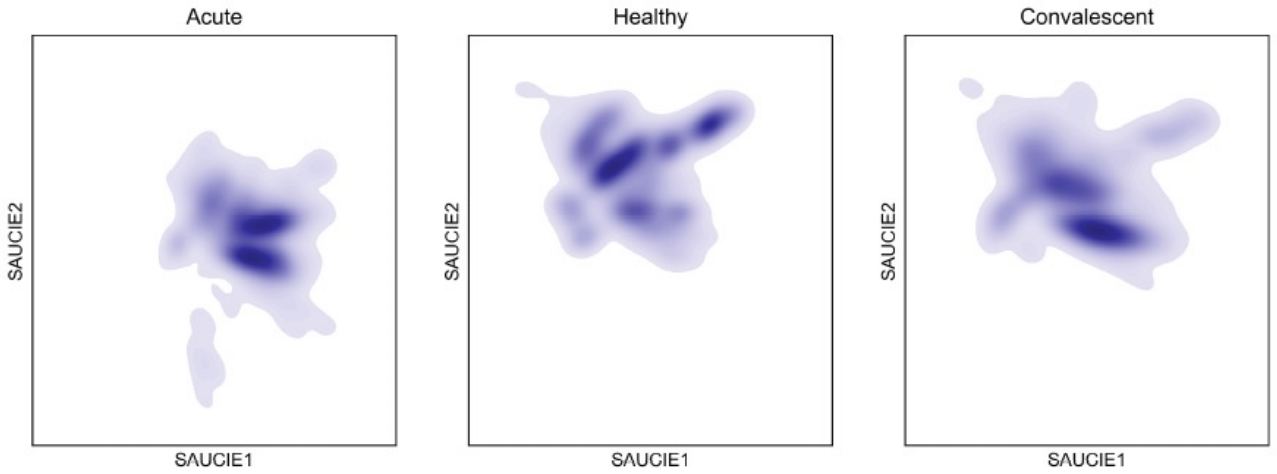


$$MMD(p,q) = \frac{1}{m^2}\Sigma_{i,j\in m}K(p_i,p_j) - \frac{2}{mn}\cdot\Sigma_{i,j}K(p_i,q_j) + \frac{1}{n^2}\Sigma_{i,j\in n}K(q_i,q_j)$$

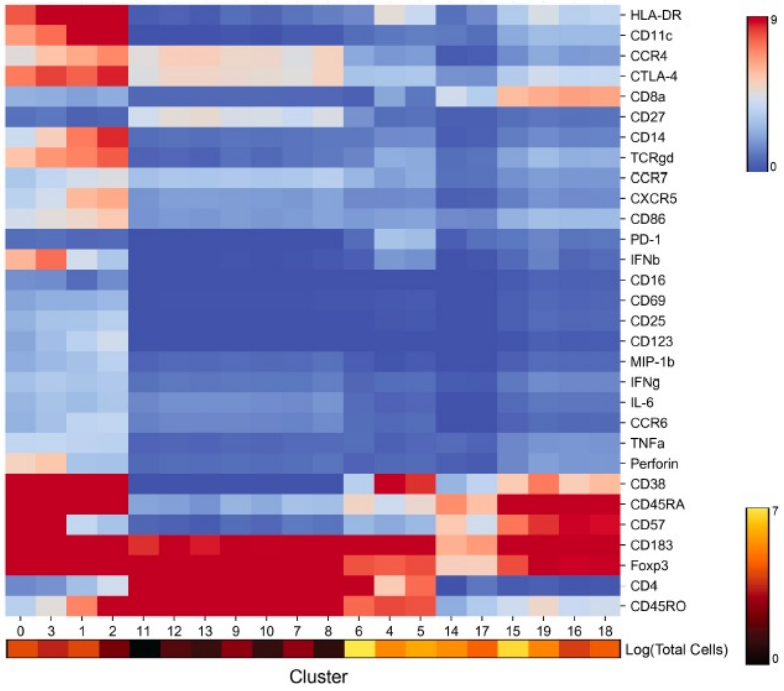
[Dziugaite, Roy, Ghahramani, UAI 2015]


Before MMD
After MMD

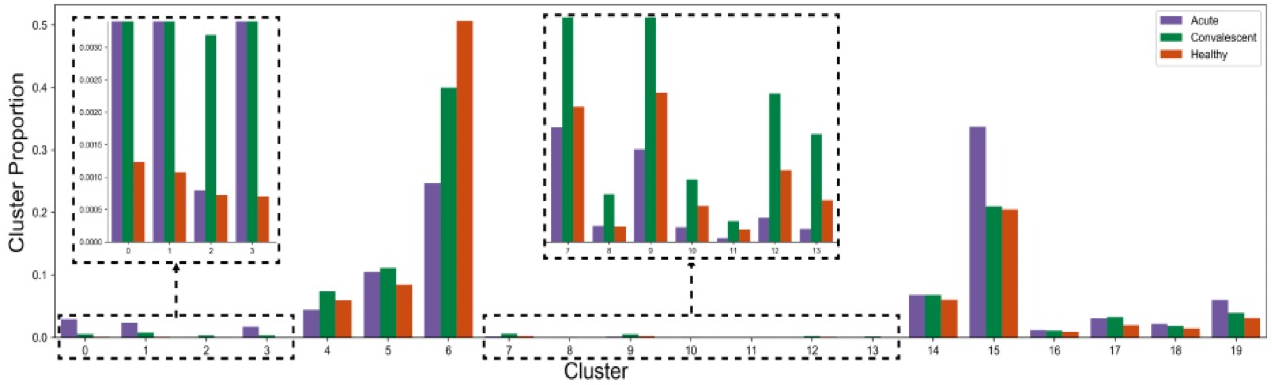
Cellular Manifolds of Dengue Patients



Cell Clusters: 180 Samples Combined

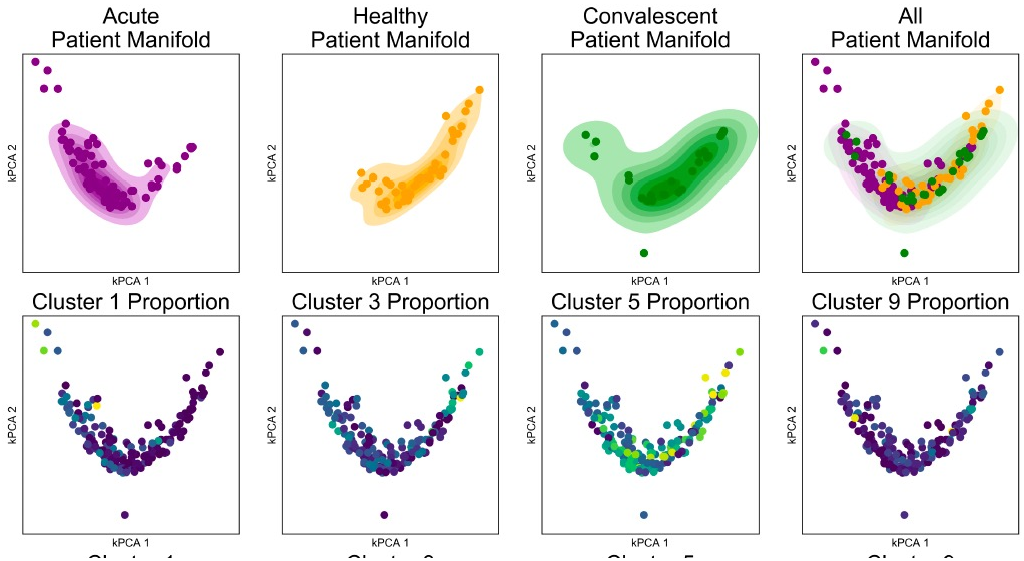


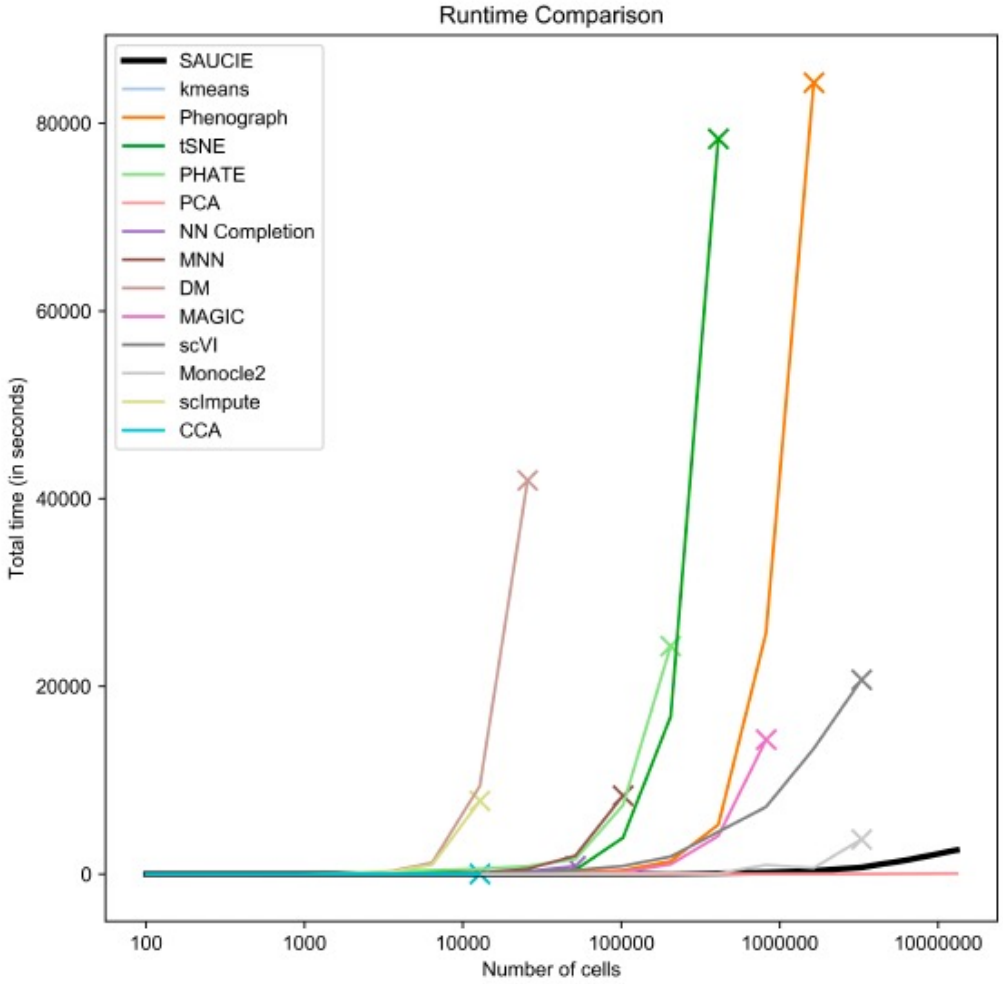
Patient Cluster Signatures



MMD distance can be derived between patient cluster proportions and embedded

Patient manifolds show coherent organization

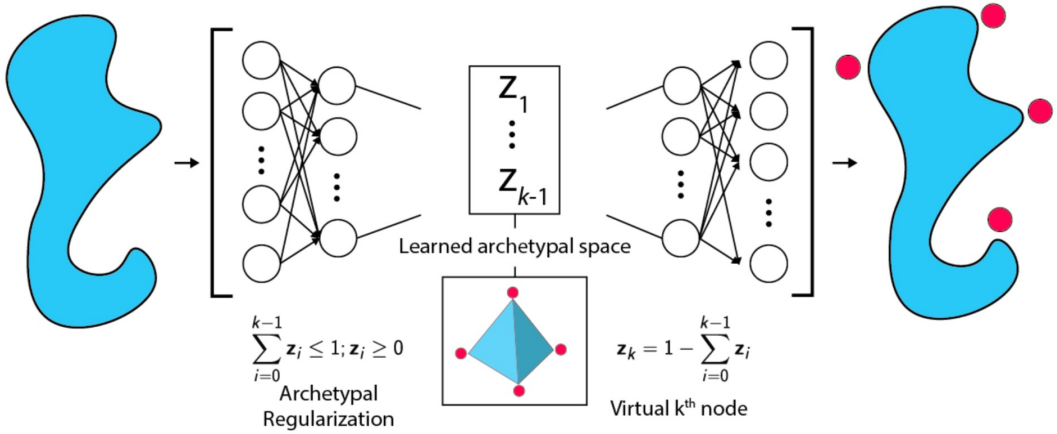



Runtime Comparison
SAUCIE
kmeans
Phenograph
tSNE
PHATE
PCA
NN Completion
MNN
DM
MAGIC
scVI
Monocle2
scImpute
CCA
Total time (in seconds)
0
20000
40000
60000
80000
100
1000
10000
100000
1000000
10000000
Number of cells

AAnet

Archetypal Analysis network

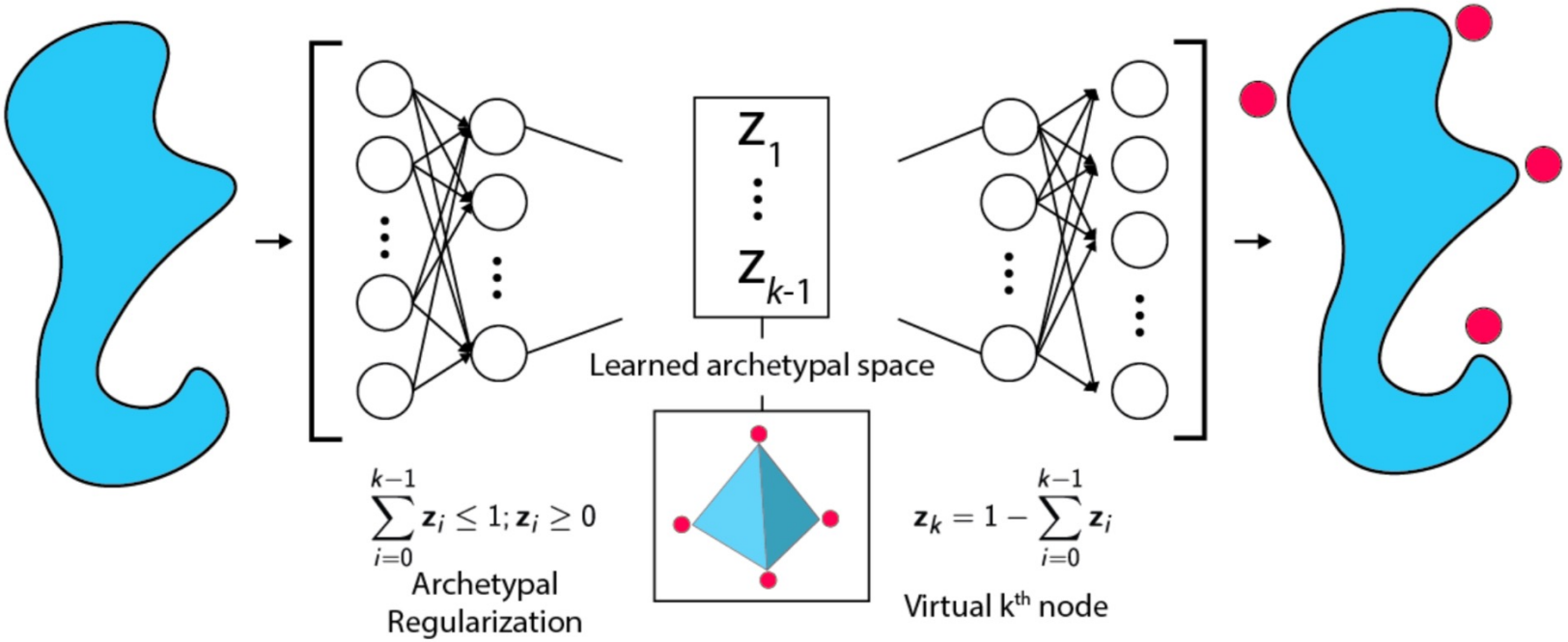
(D. van Dijk*, D. Burkhardt*, et al., IEEE Big 2019)



Main idea: **Archetypal analysis on non-linear spaces using neural networks**

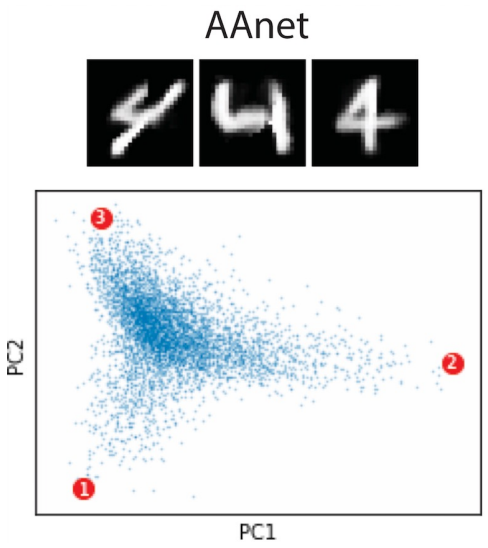
David van Dijk
Dan Burkhardt
Alex Tong
Guy Wolf
Smita Krishnaswamy

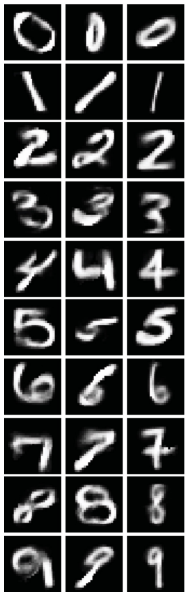
Learn non-linear transformation to fit data into convex polytope



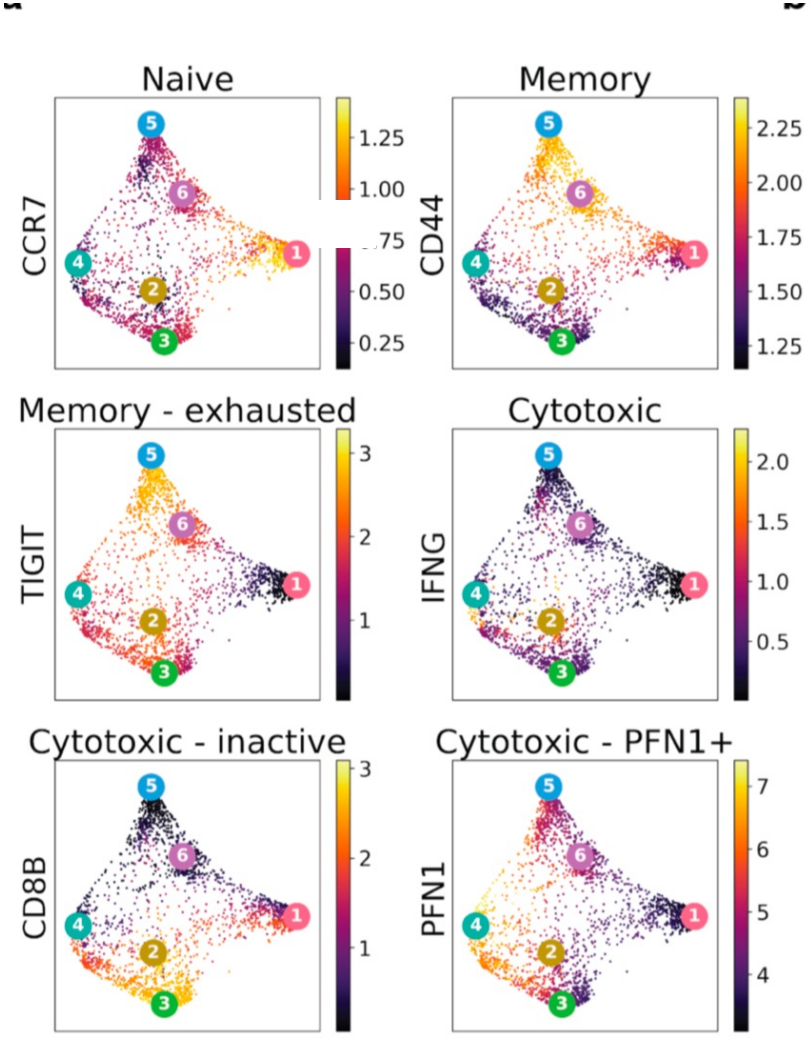
Learns each datapoint as an archetypal mixture with mixing coefficients z_i

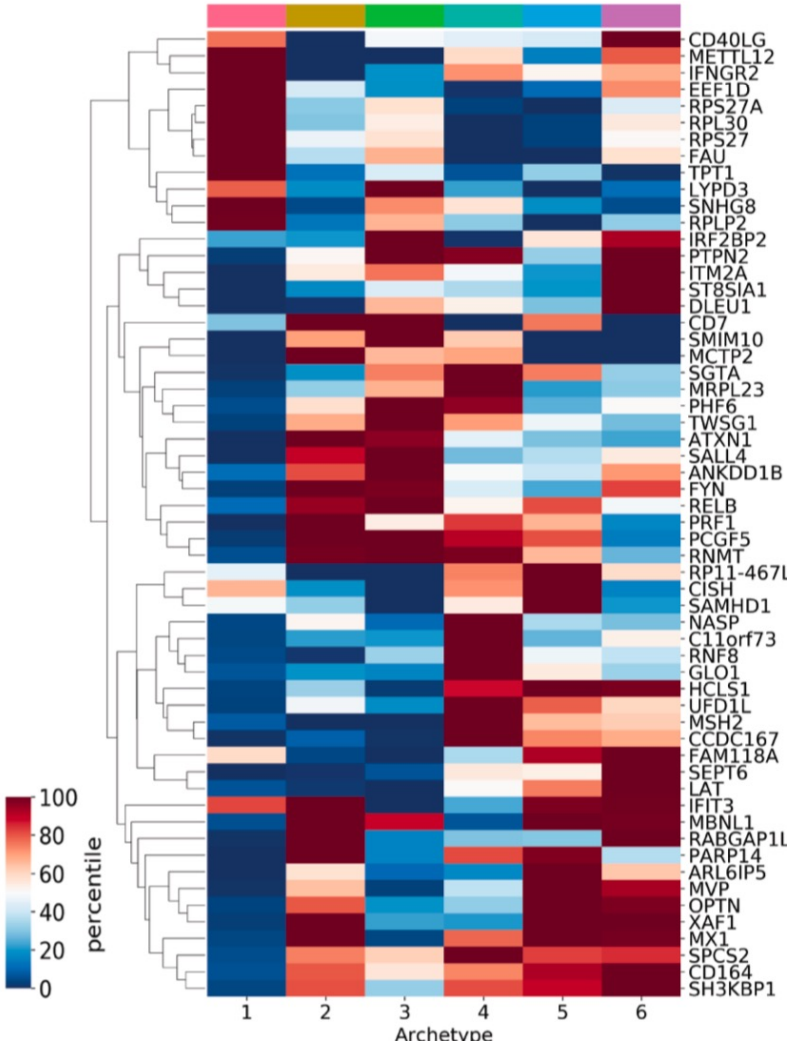
AAnet finds non-linear archetypes



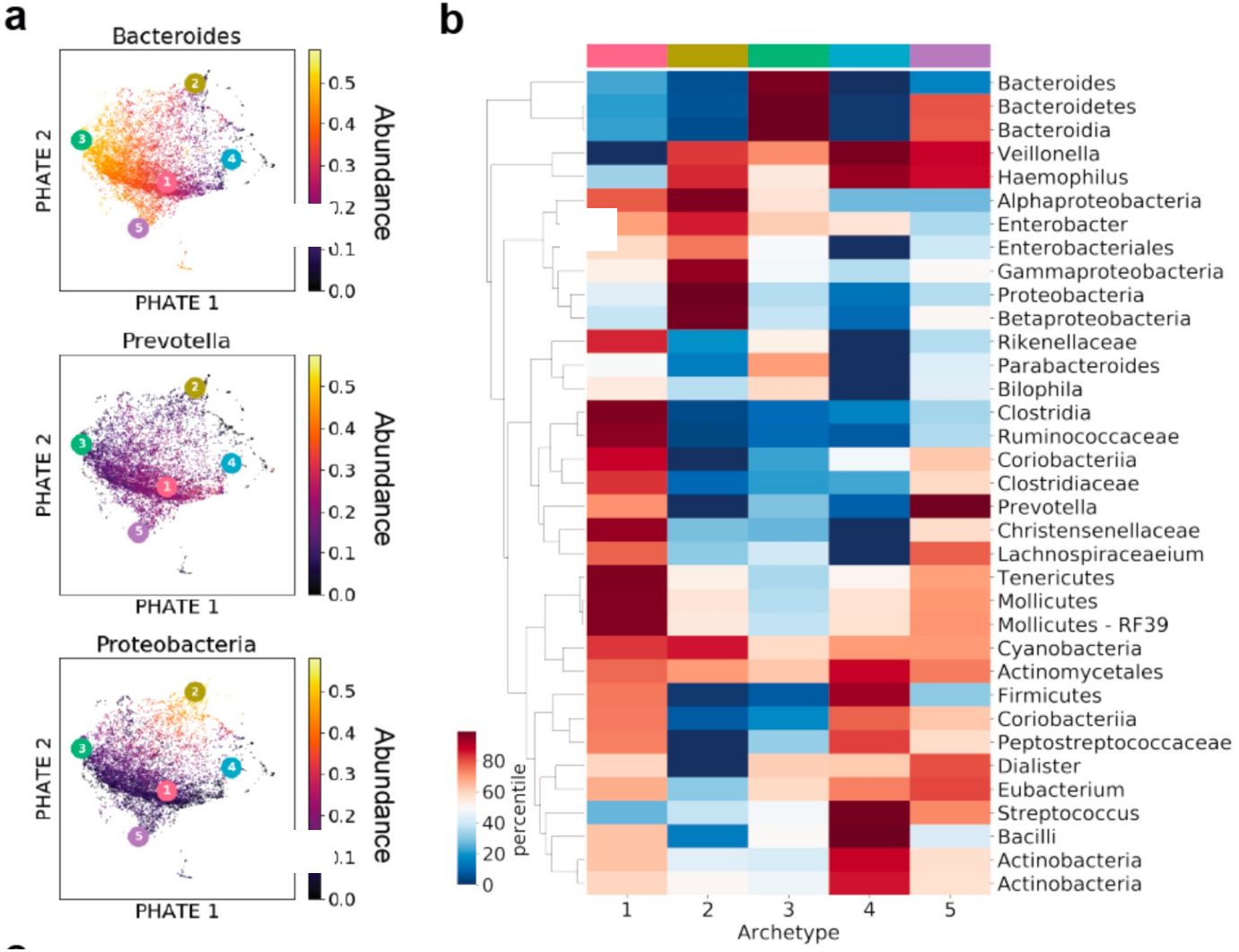


AAnet describes TIL state space





AAnet on gut microbiome samples



Unsupervised Pretraining

- Greedy layer-wise unsupervised pretraining
  - Enabled deep networks for the first time (without special architectures like convolution or recurrence)
  - A representation learned for one task (unsupervised learning) can be sometimes useful for another task (supervised learning with the same input domain)
- Approach: Train a single layer at a time to learn a representation
  - E.g., use a single layer autoencoder
  - Each layer is pretrained to take the output of the previous layer and produce a new representation that is simpler
  - Finally, train the entire network to perform the task

Further reading

- Goodfellow et al., chapters 14 and 15
- Amodio et al. Nature Methods 2019
- van Dijk et a. IEEE Big Data 2019
- https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
- https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
- Kingma and Welling Foundations and Trends in ML 2019