COMP 6321 Machine Learning

Dimensionality Reduction & Autoencoders

Computer Science & Software Engineering
Concordia University, Fall 2020

"Curse of dimensionality"

high dimensional spaces are *vast* and *hard to fill up!*

Richard Bellman

- When dimensionality of data $\mathbf{x} \in \mathbb{R}^D$ or features $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^M$ is large, problems can quickly arise:
  1. Our intuition about distance, orthogonality, volume is based on experience in 2D or 3D. In high dimensions these **intuitions break down**.
  2. Many spatial data structures **do not scale well** with dimension, for example *k*-d trees (nearest neighbour).
  3. Can be **harder to avoid over-fitting**, especially for models that have parameters per-dimension.

In high dimensions, any two random vectors $\mathbf{u}, \mathbf{v}$ are essentially orthogonal.
In high dimensions, nearly all the volume of a ball is near its surface.
In high dimensions, nearly all the probability density of a Gaussian is away from the origin.

"In a 30-dimensional grocery store, anchovies can be next to fish *and* next to pizza toppings." – Geoff Hinton

Ways to reduce dimension

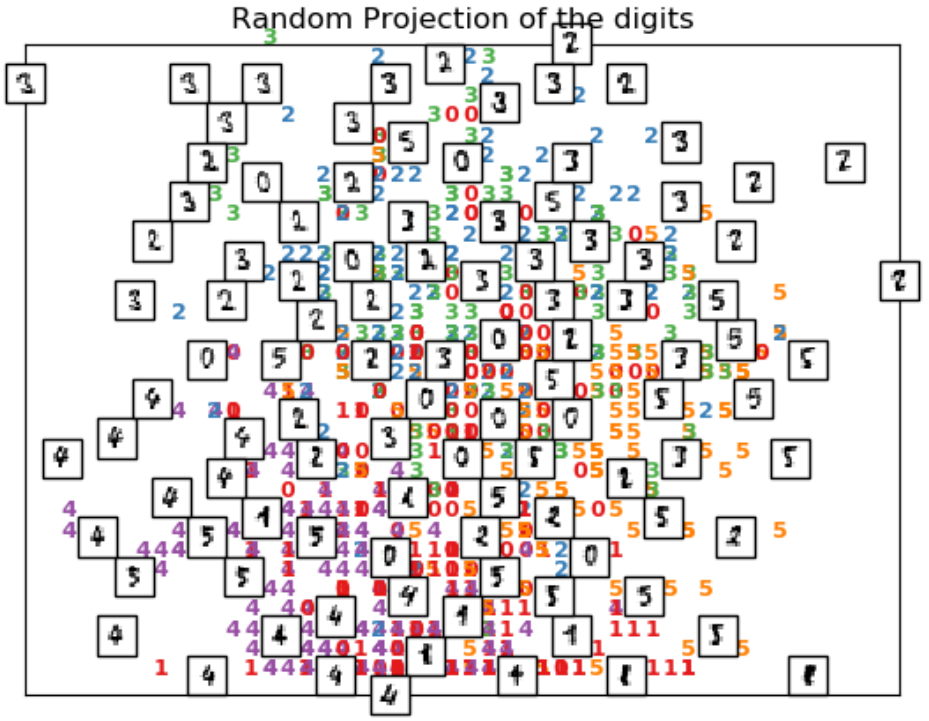
- Perform feature selection, *i.e.* only the top *K* most informative features to be used for a task.
- Engineer new domain-specific features, and use them to replace some of the original raw features. (Or rely on feature selection to choose.)
- Linear transformation to a subspace
  - Random projection? PCA? Other?
- Non-linear transformation to a manifold
  - Are some mappings more useful than others?

Dimensionality Reduction

- Random projections have applications in signal processing, sparse coding, but tend to make classification harder

784-dimensional space

map



2-dimensional space

Image: https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html

Dimensionality Reduction

- Principle component analysis useful for isolating the strongest variations in the data, discovers some structure

784-dimensional space

map

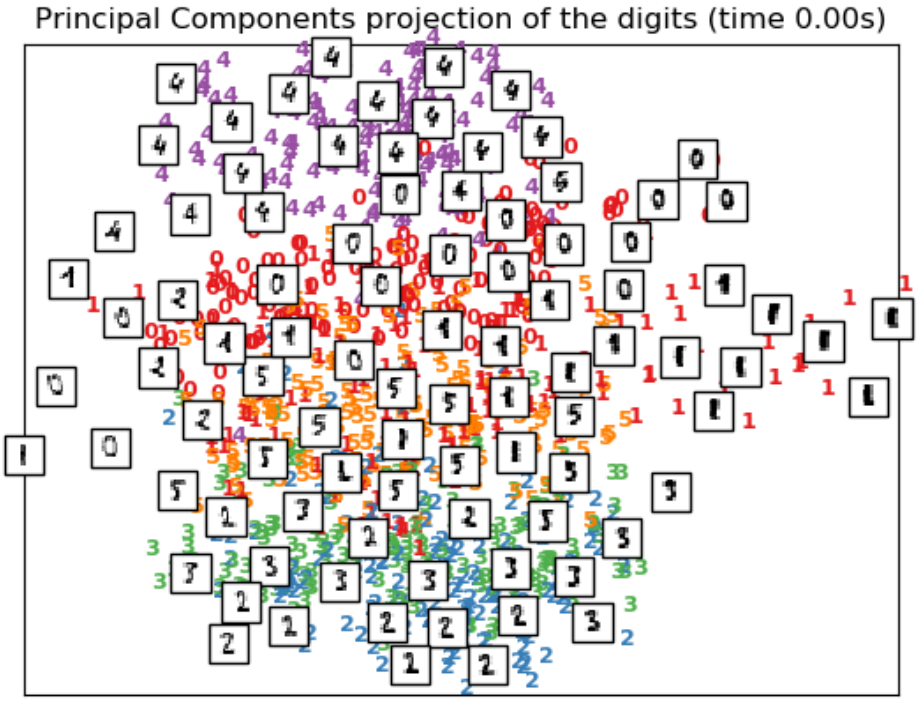


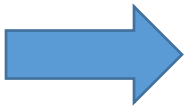
Image: https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html

Dimensionality Reduction

- Stochastic neighbour embedding (SNE, t-SNE) useful for visualizing clusters or topologies.

Note: SNE does *not* learn an explicit map from data space to embedding space, so you can't ask the embedding of a test point. Useless for classification!

784-dimensional space



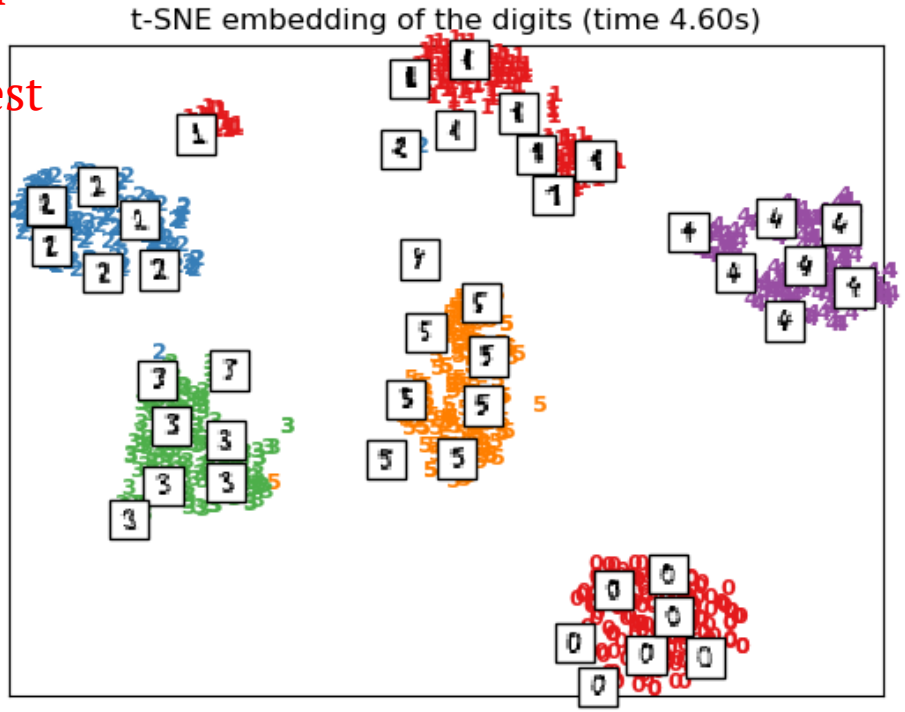
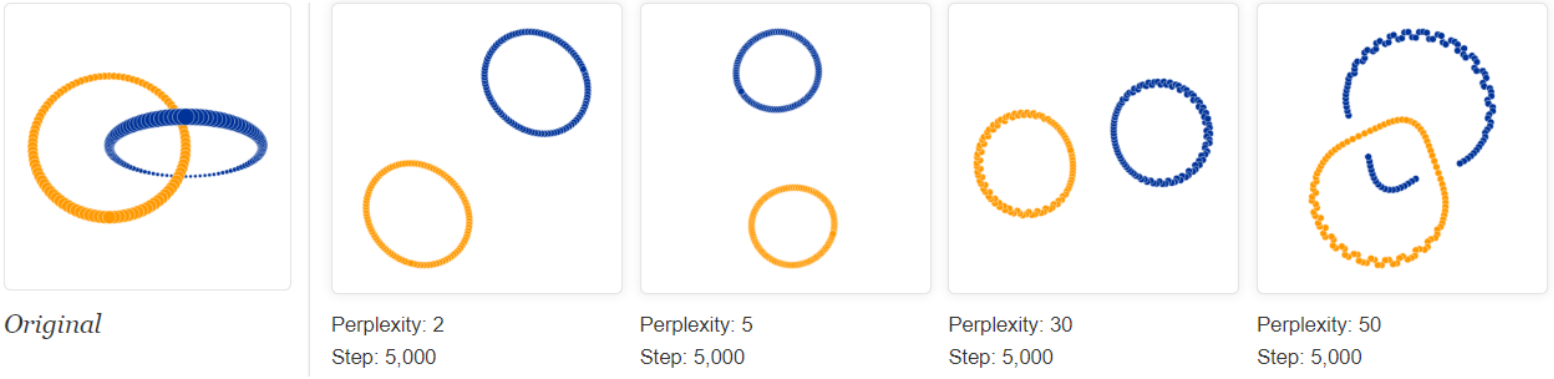
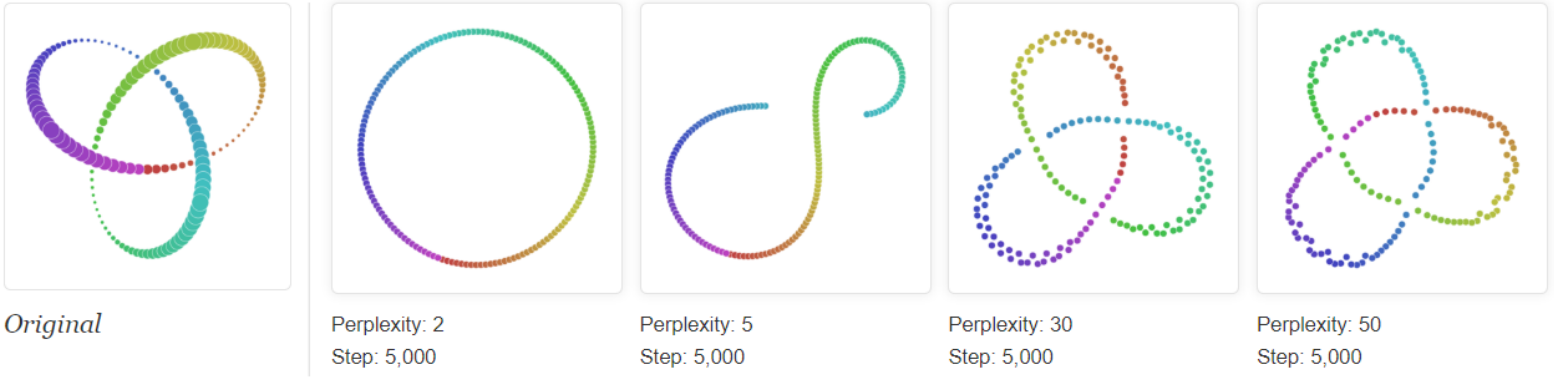


Image: https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html

Stochastic Neighbour Embedding

- Check out https://distill.pub/2016/misread-tsne/ for **awesome** t-SNE demos:





sklearn.manifold.TSNE

class sklearn.manifold. **TSNE** (*n_components=2, perplexity=30.0, early_exaggeration=12.0, learning_rate=200.0, n_iter=1000, n_iter_without_progress=300, min_grad_norm=1e-07, metric='euclidean', init='random', verbose=0, random_state=None, method='barnes_hut', angle=0.5*)

[source]

t-distributed Stochastic Neighbor Embedding.

t-SNE [1] is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.

It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples. For more tips see Laurens van der Maaten's FAQ [2].

Principle Component Analysis (PCA)

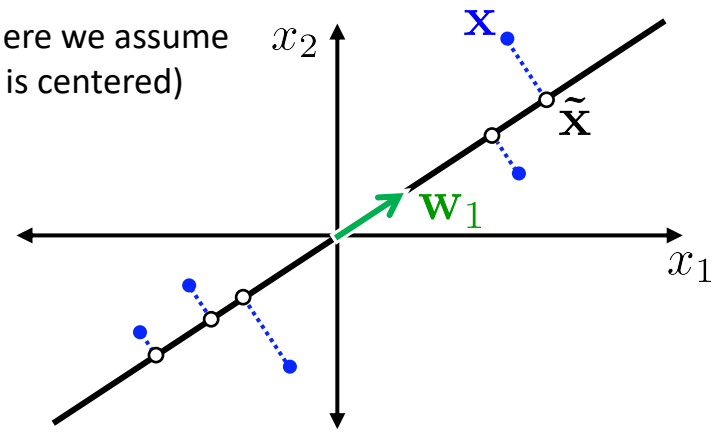
hyperparameter

- **Intuition:** project D-dimensional data to an M-dimensional subspace, in a way that approximates the original data ("preserves most information").
- **Main idea:** given data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with $\mathbf{x}_i \in \mathbb{R}^D$, find points $\{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ with $\mathbf{z}_i \in \mathbb{R}^M$ and an orthonormal basis $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_M \end{bmatrix} \in \mathbb{R}^{D \times M}$ such that the reconstruction error is minimized:

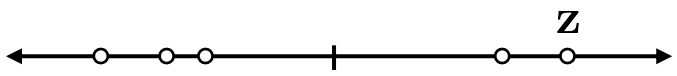
$$\frac{1}{2} \sum_{i=1}^{N} \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2$$

$$\tilde{\mathbf{x}}_i = \mathbf{W}\mathbf{z}_i$$

(Here we assume ***X*** is centered)



(embedding of 1-d subspace in 2-d)



(1-d representations of data)

sklearn.decomposition.PCA

class sklearn.decomposition. **PCA** (*n_components=None*, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*) [source]

Principal component analysis (PCA)

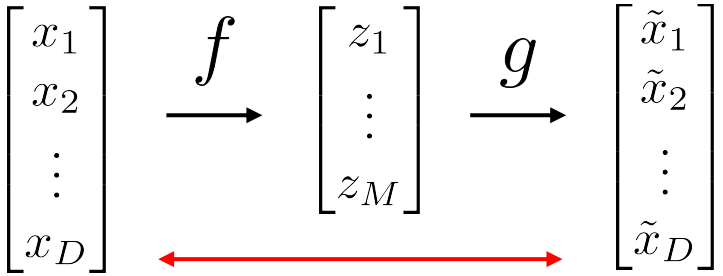
Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD.

Methods

fit (self, X[, y])	Fit the model with X.
fit_transform (self, X[, y])	Fit the model with X and apply the dimensionality reduction on X.
get_covariance (self)	Compute data covariance with the generative model.
get_params (self[, deep])	Get parameters for this estimator.
get_precision (self)	Compute data precision matrix with the generative model.
inverse_transform (self, X)	Transform data back to its original space.

Autoencoders (AEs)

- With PCA, once we have orthonormal $\mathbf{W}$ we can "encode" the data by $\mathbf{z}_i = \mathbf{W}^T \mathbf{x}_i$
- This means PCA searches for a $\mathbf{W}$ that provides a good (and linear) "self encoding" $\tilde{\mathbf{x}}_i = \mathbf{W}\mathbf{W}^T \mathbf{x}_i$
- **Autoencoders** generalize this to arbitrary encoder and decoder functions:



1. encode: $\mathbf{z}_i = f(\mathbf{x}_i, \mathbf{w})$
2. decode: $\tilde{\mathbf{x}}_i = g(\mathbf{z}_i, \boldsymbol{\omega})$
3. loss:

$$\ell(\mathbf{w}, \boldsymbol{\omega}) = \frac{1}{2} \sum_{i=1}^{N} \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2$$

Autoencoders (AEs)

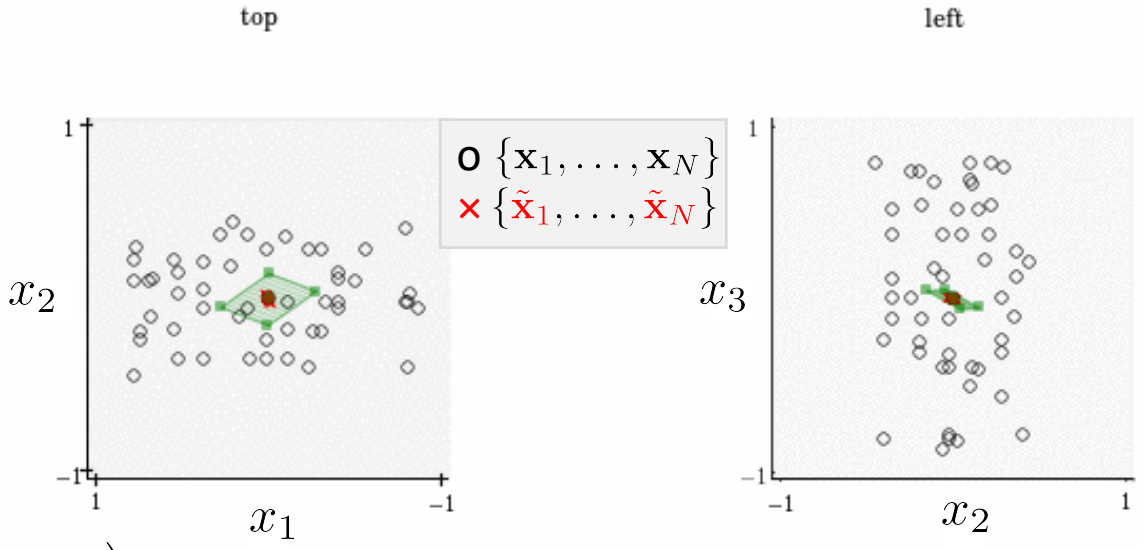
- Unsupervised “representation learning” technique
- Learns a non-linear encoder and decoder function
  - Early autoencoders were stochastic neural networks where the encoder and decoder shared parameters
  - Encoder and decoder can be any neural network, e.g.

$$\mathbf{z} = \tanh(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \quad \tilde{\mathbf{x}} = \tanh(\mathbf{W}_2\mathbf{z} + \mathbf{b}_2)$$

  - Choice of *regularization* has major influence on **z**
- Orthonormality not usually enforced (unlike PCA)
- Reconstruction loss still implicitly prefers encodings that “preserve information”
- Can be used to decrease or *increase* dimensionality

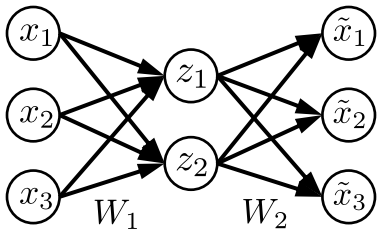
Training a 3-2-3 *tanh* autoencoder w/ SGD

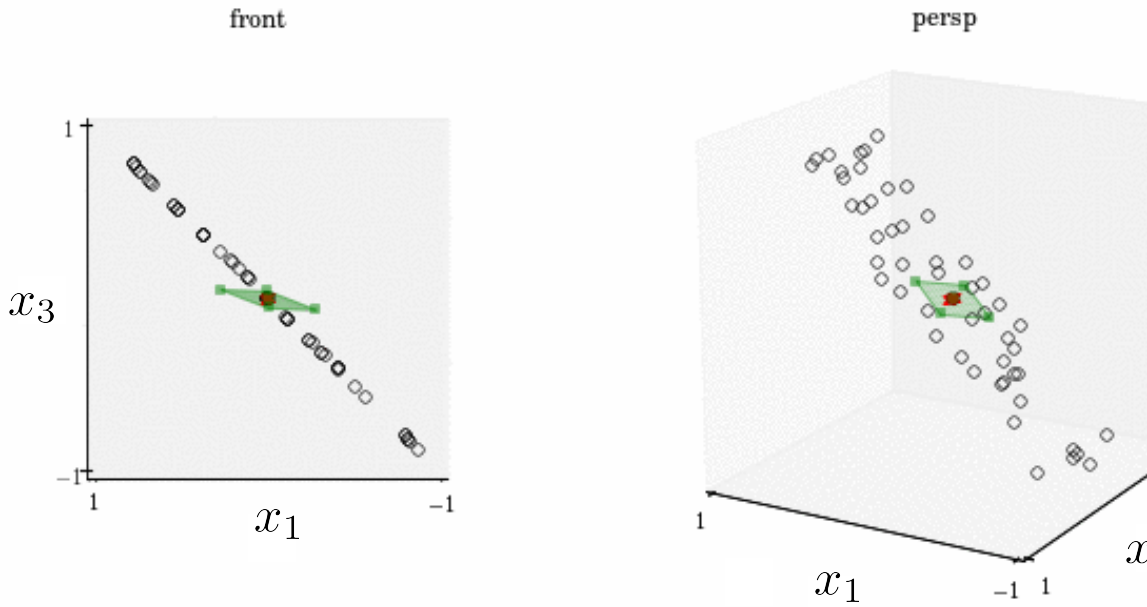
$$\mathbf{z} = f(\mathbf{x}, \mathbf{w})$$
$$\tilde{\mathbf{x}} = g(\mathbf{z}, \boldsymbol{\omega})$$
$$\ell = \tfrac{1}{2} \left\| \mathbf{x} - \tilde{\mathbf{x}} \right\|^2$$



$$f(\mathbf{x}, \mathbf{w}) = \tanh\left(\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} w_7 \\ w_8 \end{bmatrix} \right)$$

$$g(\mathbf{z}, \boldsymbol{\omega}) = \begin{bmatrix} \omega_1 & \omega_4 \\ \omega_2 & \omega_5 \\ \omega_3 & \omega_6 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} \omega_7 \\ \omega_8 \\ \omega_9 \end{bmatrix}$$





Example of using regularization to influence the encoding

Rejected ICCV 2013 submission

Slow Feature Networks for Representation Learning

"prefer z_j to be piecewise constant through time"

Abstract

It is long-understood that incorporating prior knowledge can aid representation learning. A prior is particularly valuable if it is both general-purpose and helps to 'disentangle' the representation. Smoothness, sparsity, spatial and temporal coherence are all prominent examples. We focus on temporal coherence, or *slowness*. We show that a standard, deep feed-forward architecture can be modified to find 'slow' representations, here using unlabeled video. We also present the training calculations in vectorized form, for fast and easy implementation in MATLAB/NumPy.

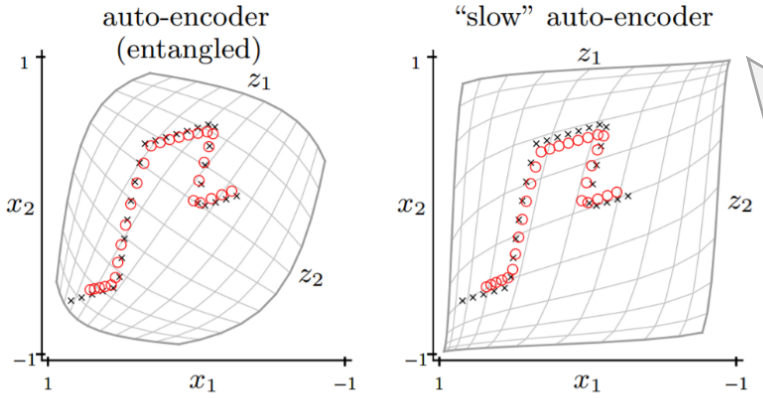


Figure 1. Illustration of how prior knowledge helps improve representations. An auto-encoder was trained to encode each input x to some hidden representation z, and then decode (the red circles). The mesh shows complete domain $z \in [-1, 1]^2$ decoded to input space. The slow feature encoder prefers embeddings that change slowly and sparsely over the input sequence, hence 'disentangled'.

Example: if you know the encoding should be piecewise "smooth" over time (e.g. from video), you can explicitly enforce that.

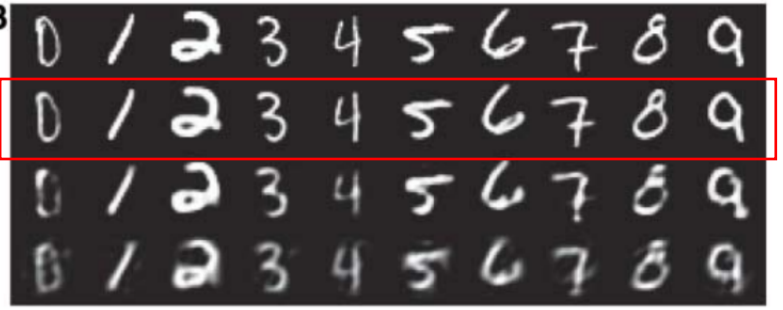
Reducing the Dimensionality of Data with Neural Networks

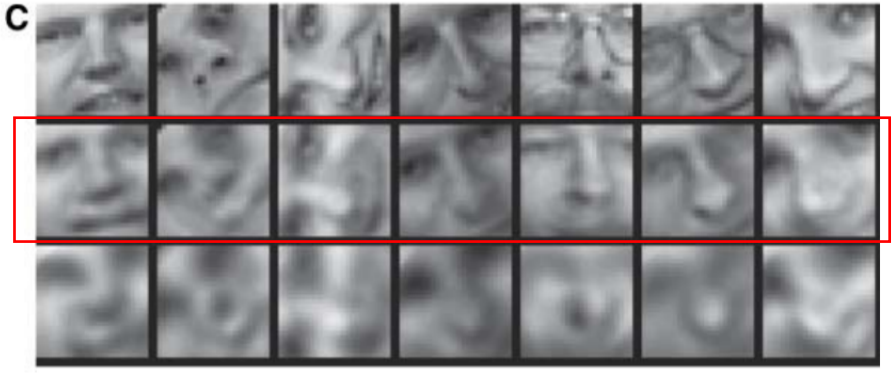
G. E. Hinton* and R. R. Salakhutdinov

10,500 citations

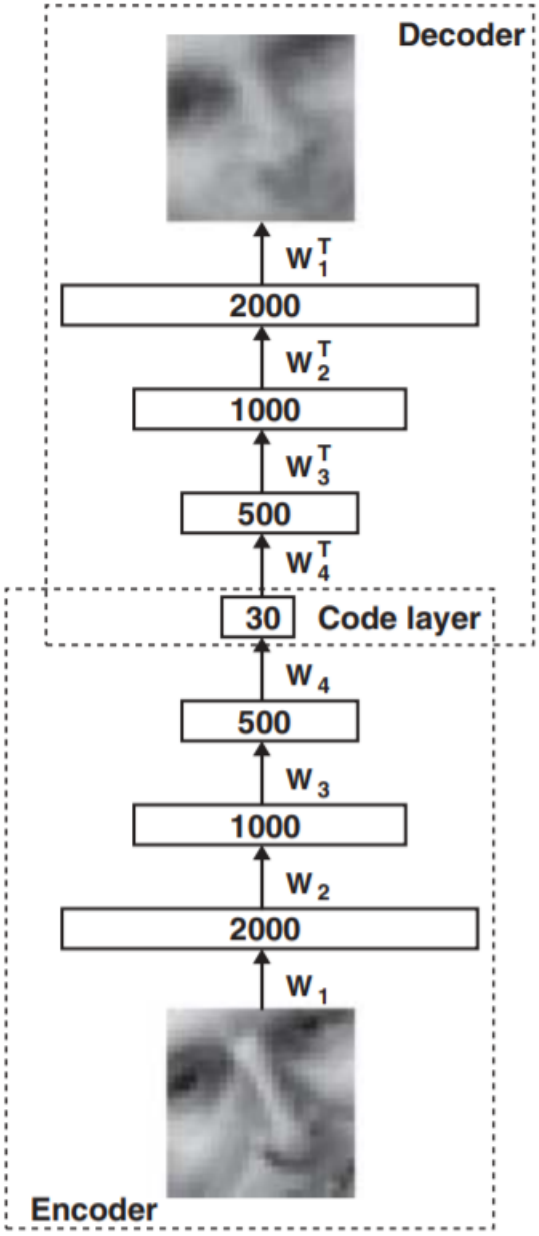
High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

(**B**) Top to bottom: A random test image from each class; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional logistic PCA and standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87.





(**C**) Top to bottom: Random samples from the test data set; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.



https://www.cs.toronto.edu/~hinton/science.pdf

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

PCA representation

Auto-encoder (AE) representation

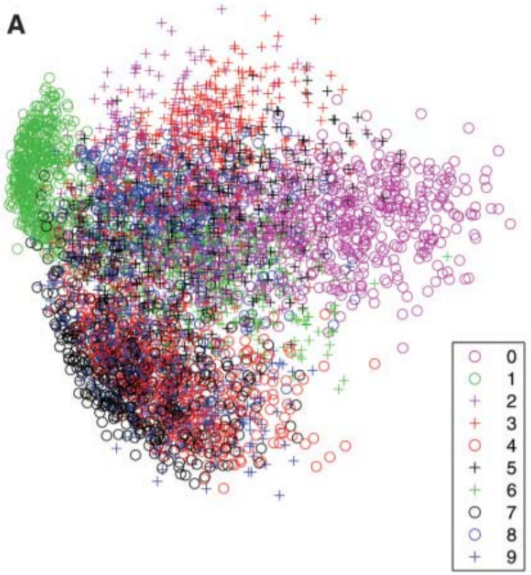
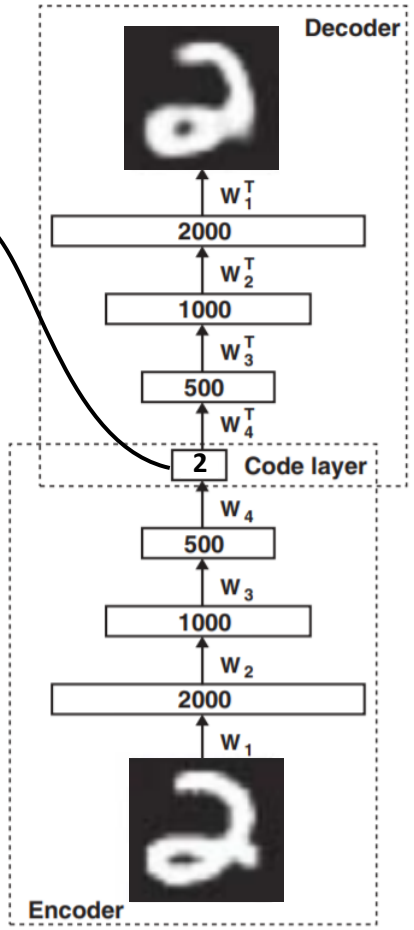


Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).



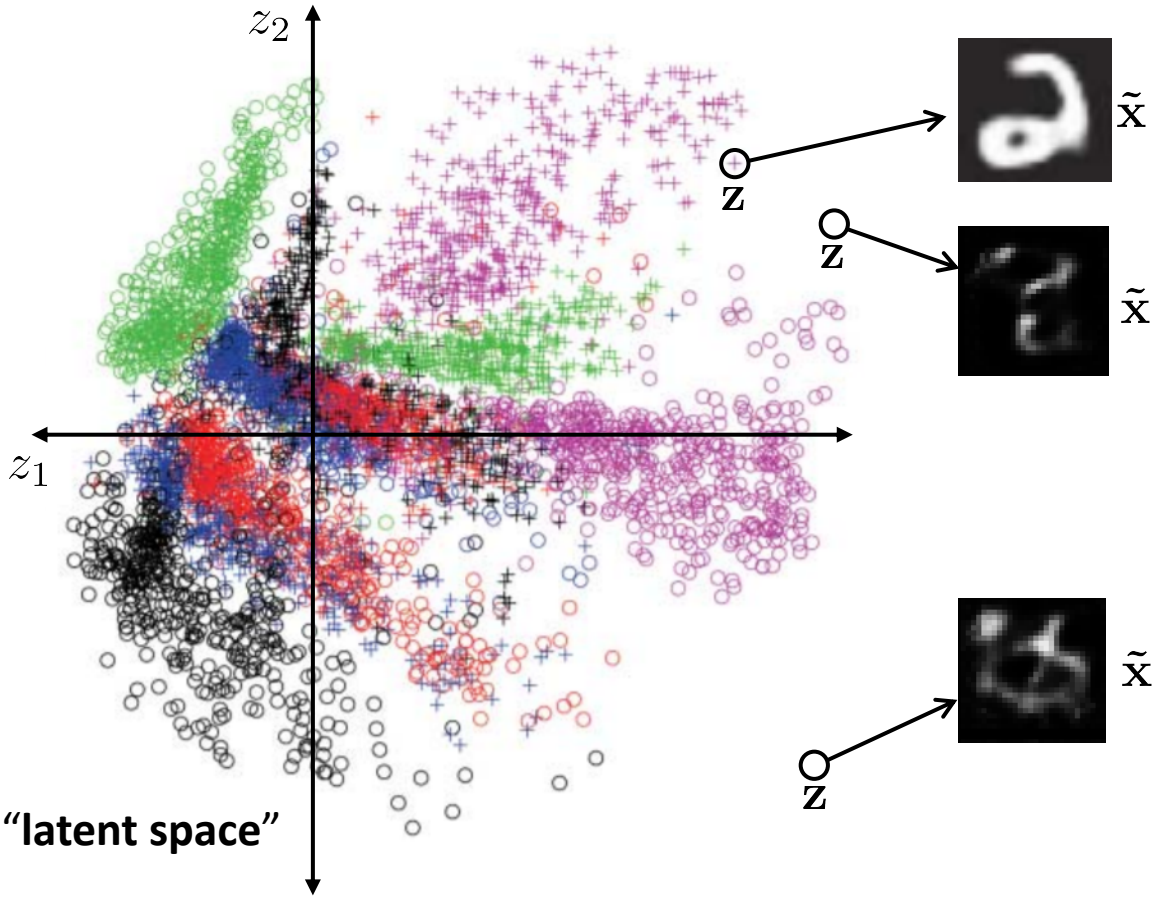
With linear encoder/decoder, PCA representation is worse at reconstruction and worse for classification

With more model capacity, reconstruction loss *incidentally* finds structure that would also be useful for classification!

https://www.cs.toronto.edu/~hinton/science.pdf

Limitation of basic auto-encoders

- After training, most of $\mathbf{z}$-space decodes to nonsense



Q: How might you build a *generative* model from a representation like this?

How could we sample a 'latent code' that could be decoded to a 'realistic' data item?

$$\mathbf{z} \sim p$$

$$\tilde{\mathbf{x}} = g(\mathbf{z}, \boldsymbol{\omega})$$

Autoencoder types

- **Weight decay:** penalizing $\|\mathbf{w}\|^2$ essentially smooths the embedding while also biasing it
- **Sparse autoencoder:** $\mathbf{z}$ very high dimensional, but encourage $p(z_j = 0 \mid \mathbf{x})$ to be large for most $\mathbf{x}$

  https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf
- **Denoising autoencoder:** encourage "corrupted" data $\mathbf{x}_i + \boldsymbol{\epsilon}$ to have same encoding $\mathbf{z}_i$ as uncorrupted $\mathbf{x}_i$
  - Gives meaningful encodings to points $\mathbf{x}$ farther from $\mathbf{x}_i$
- **Variational autoencoder:** encoder $f(\mathbf{x}, \mathbf{w})$ outputs a *distribution* over $\mathbf{z}$, so each $\mathbf{x}$ has a "soft" encoding
  - Also tries to "pack" the code distributions tightly

Variational Autoencoders (VAEs)

Auto-Encoding Variational Bayes

Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Abstract

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions is two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

ICLR 2014
>10,000 citations

Variational Autoencoders (VAEs)

Intuition: for an input example $\mathbf{x}$, we may be more certain about some "latent attributes" than others

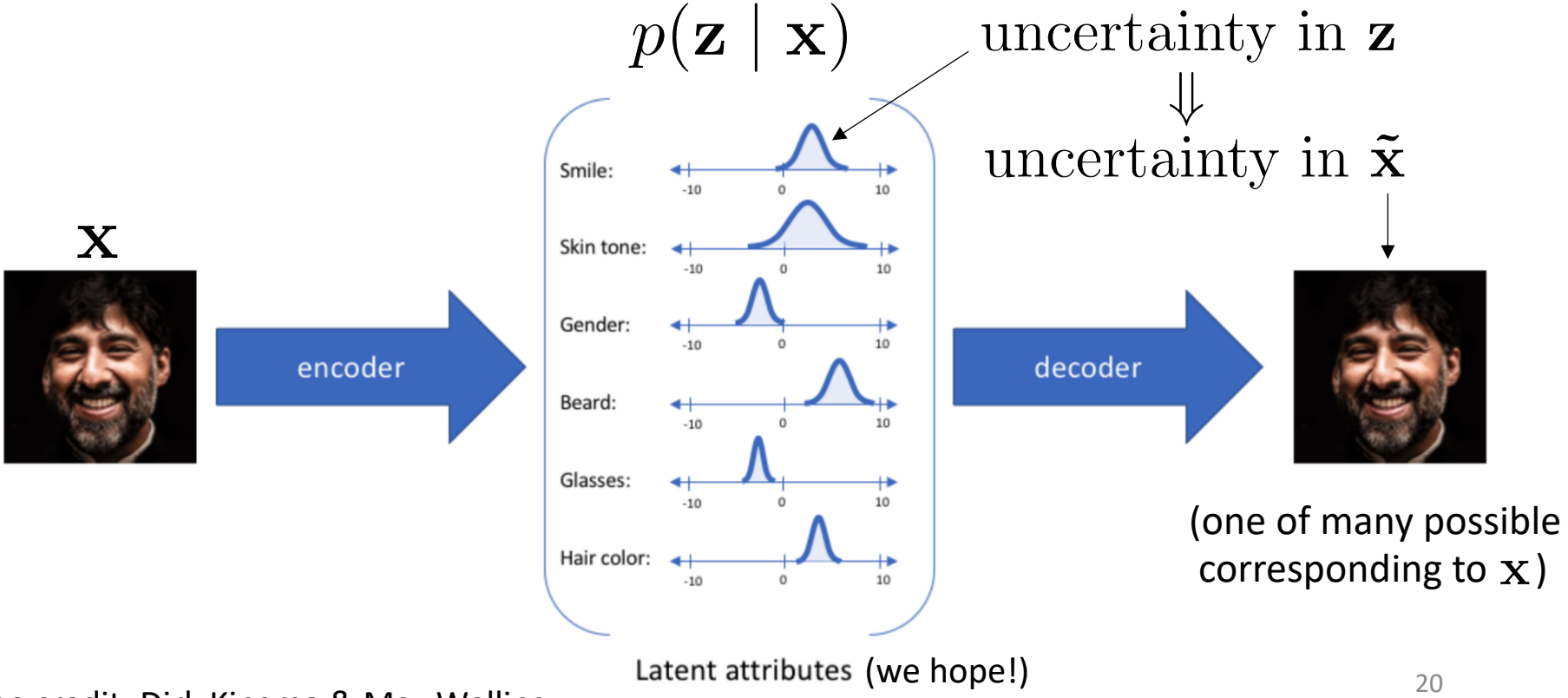


Image credit: Dirk Kingma & Max Welling

Variational Autoencoders (VAEs)

Idea 1: Encoder should output a distribution over codes

For AE code $\mathbf{z} = f(\mathbf{x}, \mathbf{w})$ is *deterministic* function.
For VAE it is a *stochastic* function, typically

$$p(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) \quad \text{where} \quad \underbrace{\boldsymbol{\mu}, \boldsymbol{\sigma}} = f(\mathbf{x}, \mathbf{w})$$

encoder determines parameters of **z**'s stochastic distribution, rather than determining **z** itself.

Reconstruction is now:

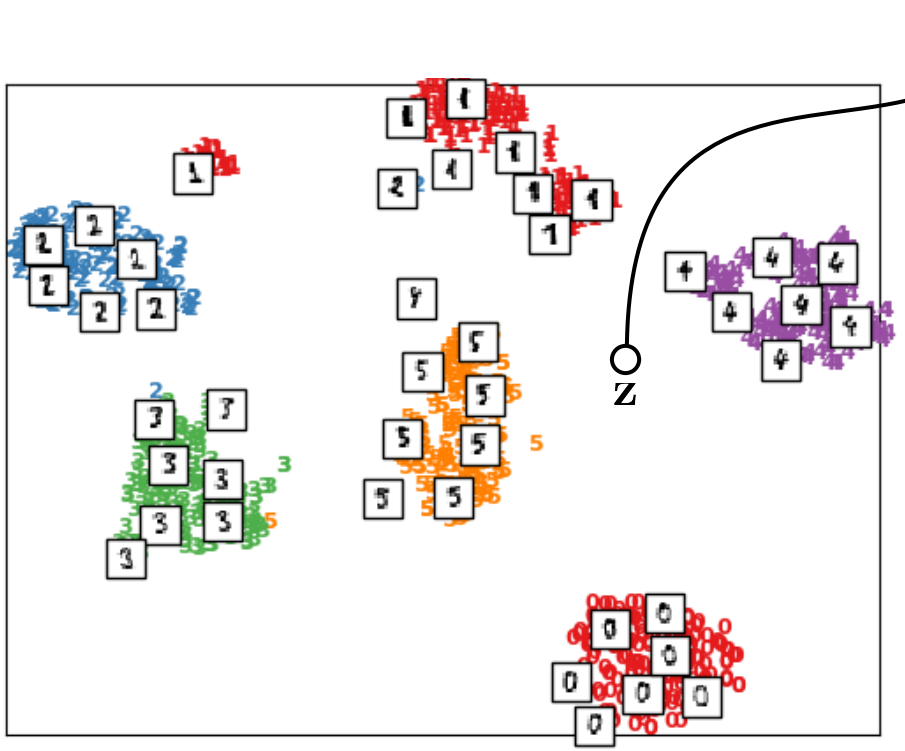
$$\boldsymbol{\mu}, \boldsymbol{\sigma} = f(\mathbf{x}, \mathbf{w}_1)$$
$$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$$
$$\tilde{\mathbf{x}} = g(\mathbf{z}, \mathbf{w}_2)$$

```
mu, sigma = f(x)
z = sample_normal(mu, sigma)
x_recon = g(z)
```

slight abuse of notation: means $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$

Variational Autoencoders (VAEs)

Idea 1 alone doesn't solve the problem that code space can be arbitrarily empty, filled with nonsense regions

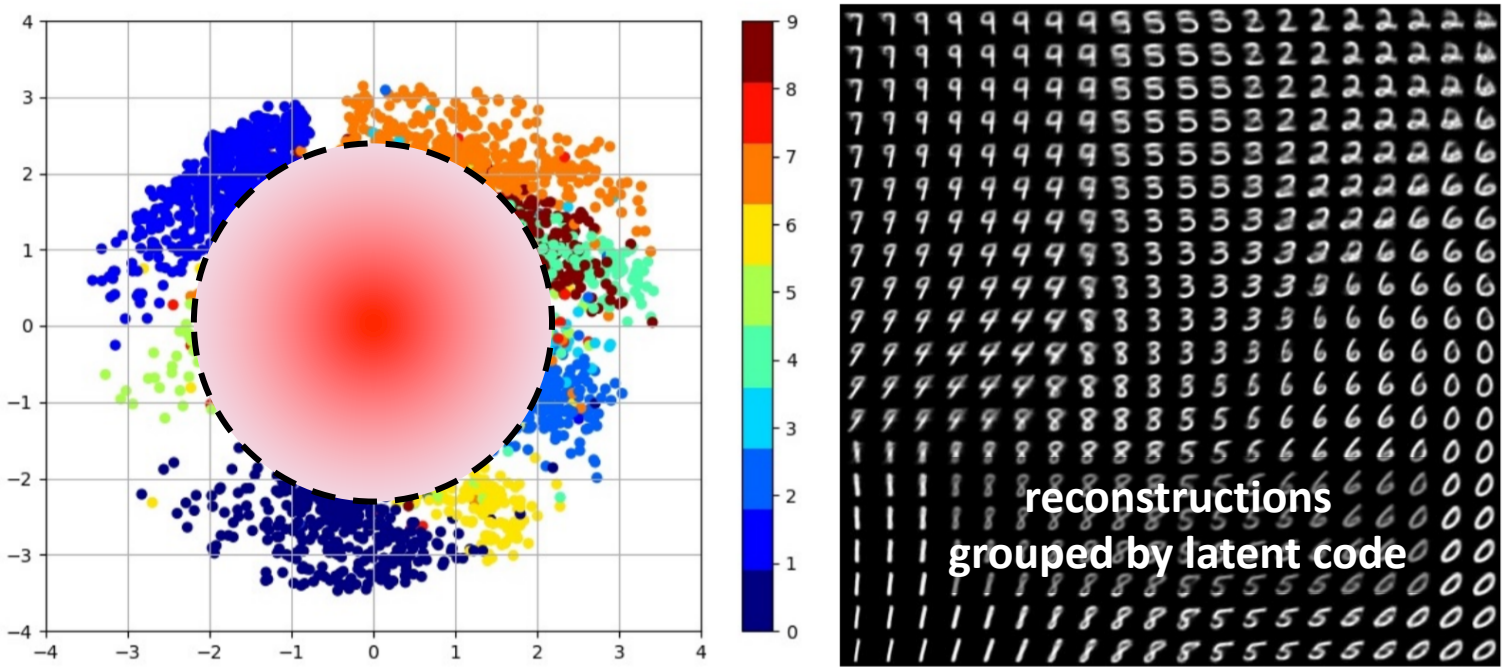


Rather than coding for nonsense, wouldn't we want this particular z to encode an image that looks *"halfway between a 5 and a 4"*?

What if we could make it easier to generate new $\tilde{x}$ at the same time?

Variational Autoencoders (VAEs)

<u>Idea 2</u>: Pack the codes together so that they overlap the high-density regions of a standard distribution

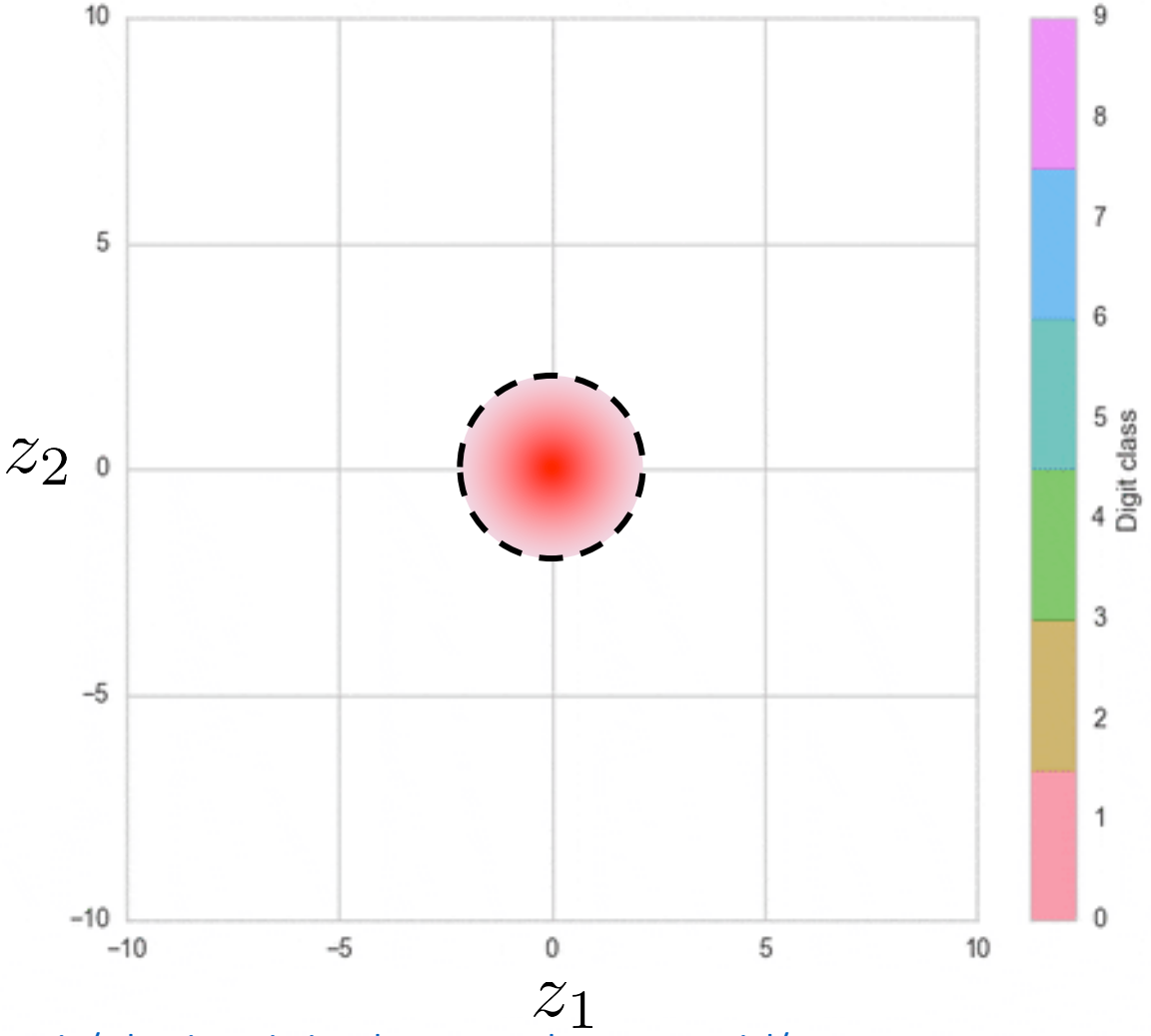


Typically choose $\mathcal{N}(\mathbf{0}, \mathbf{I})$ as standard distribution

Image source: Tschannen *et al.* 2018 https://arxiv.org/pdf/1812.05069.pdf

Variational Autoencoders (VAEs)

Animation of how the means of the latent codes shift around during learning



(latent codes of MNIST digits, where mean of code is coloured by its digit)

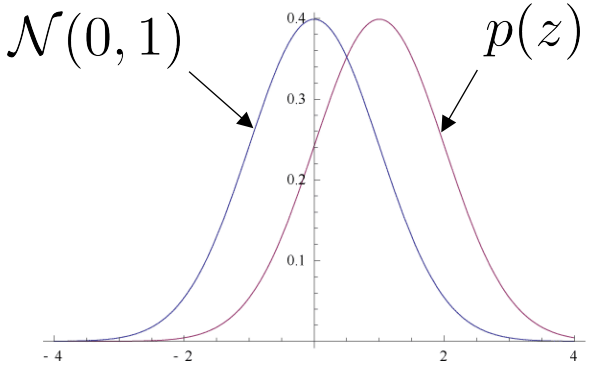
Image source: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

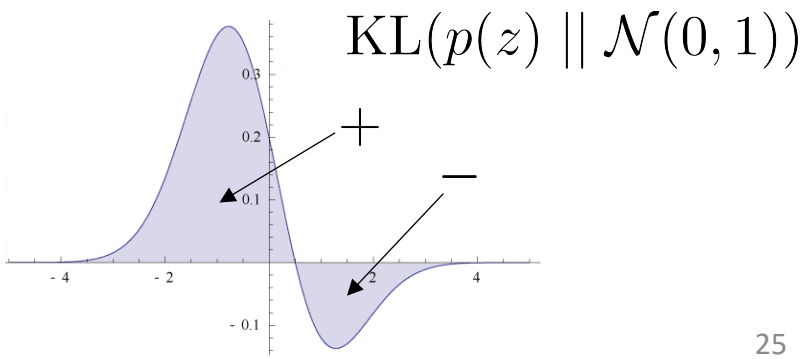
Variational Autoencoders (VAEs)

Typical VAE loss combines "reconstruction" term with a "how well the code distributions pack into $\mathcal{N}(\mathbf{0}, \mathbf{I})$" term

Overlap between each code distribution $p(\mathbf{z}_i \mid \mathbf{x}_i)$ and the "code prior" distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ is measured by the *Kullback-Leibler divergence* between them

$$D_{\mathrm{KL}}(p(\mathbf{z}_i \mid \mathbf{x}_i) \,||\, \mathcal{N}(\mathbf{0}, \mathbf{I}))$$





- Entropy (in bits) of a discrete distribution $p(\mathbf{z})$

$$H(p) = -\sum_{\mathbf{z}} p(\mathbf{z}) \log_2 p(\mathbf{z}) = \mathbb{E}_{\mathbf{z}\sim p}\left[\log_2 \frac{1}{p(\mathbf{z})}\right]$$

sum over all possible states

higher entropy means "harder to compress data sampled from p using a dictionary-like scheme"

expected # bits to encode state $\mathbf{z}$ when sampled from p and when using a dictionary designed for p

code length of state $\mathbf{z}$ when using a dictionary designed to compress data based on frequencies under p

- Cross-entropy (in bits) of $q(\mathbf{z})$ relative to $p(\mathbf{z})$

$$H(p, q) = -\sum_{\mathbf{z}} p(\mathbf{z}) \log_2 q(\mathbf{z}) = \mathbb{E}_{\mathbf{z}\sim p}\left[\log_2 \frac{1}{q(\mathbf{z})}\right]$$

$H(p, p) = H(p)$

$H(p, q) \geq H(p)$

since dictionary for q might not be ideal for encoding data from p

expected # bits to encode state $\mathbf{z}$ when sampled from p and when using a dictionary designed for q

code length of state $\mathbf{z}$ when using a dictionary designed to compress data based on frequencies under q

- KL-divergence (in bits) of $q(\mathbf{z})$ relative to $p(\mathbf{z})$

$$D_{\mathrm{KL}}(p \,||\, q) = \sum_{\mathbf{z}} p(\mathbf{z}) \log_2 \frac{p(\mathbf{z})}{q(\mathbf{z})} = H(p, q) - H(p)$$

expected # *extra* bits to encode state $\mathbf{z}$ when sampled from p and when using a dictionary designed for q

Kullbach-Leibler (KL) divergence

- Measures dissimilarity of two probability distributions:

$$D_{\mathrm{KL}}(p \,||\, q) = \int p(\mathbf{z}) \ln \frac{p(\mathbf{z})}{q(\mathbf{z})} \mathrm{d}\mathbf{z}$$

zero when $p(\mathbf{z}) = q(\mathbf{z})$

$D_{\mathrm{KL}}(\cdot \,||\, \cdot) \geq 0$

or: $\int p(\mathbf{z}) \left(\ln p(\mathbf{z}) - \ln q(\mathbf{z})\right) \mathrm{d}\mathbf{z}$

$D_{\mathrm{KL}}(p \,||\, q) = 0$ iff $p = q$ a.e. ("almost everwhere")

$D_{\mathrm{KL}}(p \,||\, q) \neq D_{\mathrm{KL}}(q \,||\, p)$ so not symmetric, not a 'distance'

- Additive for independent distributions, i.e., when

$$p(\mathbf{z}_1, \ldots, \mathbf{z}_N) = p_1(\mathbf{z}_1) \cdots p_N(\mathbf{z}_N)$$
$$q(\mathbf{z}_1, \ldots, \mathbf{z}_N) = q_1(\mathbf{z}_1) \cdots q_N(\mathbf{z}_N)$$

then $D_{\mathrm{KL}}(p \,||\, q) = \sum_{i=1}^{N} D_{\mathrm{KL}}(p_i \,||\, q_i)$

Variational Autoencoder loss

- Typical VAE loss is therefore

but what's the actual formula for *this* term?

$$\ell(\mathbf{w}, \boldsymbol{\omega}) = \sum_{i=1}^{N} \frac{1}{2} \left\| \mathbf{x}_i - \tilde{\mathbf{x}}_i \right\|^2 + \beta D_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2) \,||\, \mathcal{N}(\mathbf{0}, \mathbf{I})\right)$$

where $\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i = f(\mathbf{x}_i, \mathbf{w})$ are the encoded params,

$\mathbf{z}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2)$ are the sampled latent codes

$\tilde{\mathbf{x}}_i = g(\mathbf{z}_i, \boldsymbol{\omega})$ are the reconstructed data

$\beta \geq 0$ is the strength of KL "packing force"

important VAE hyperparameter introduced by "β-VAE" paper

(original VAE just assumed $\beta = 1$)

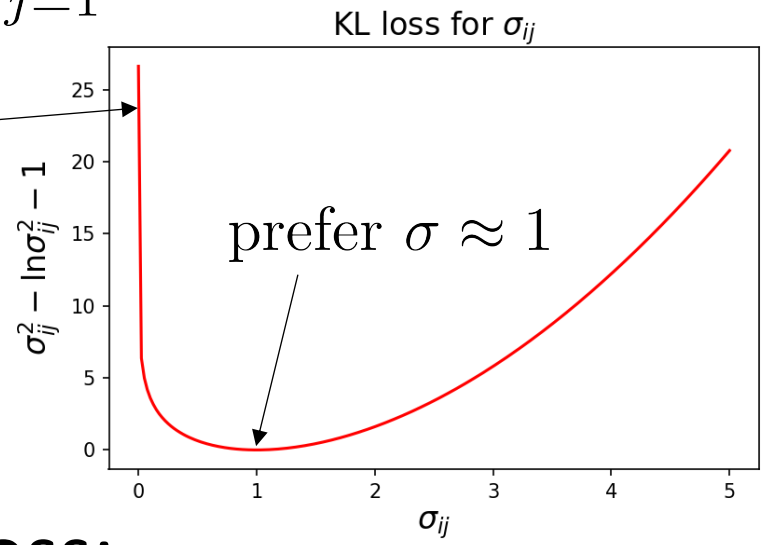
Higgins et al ICLR 2017

$(\text{assume } \mathbf{z} \in \mathbb{R}^M)$

Variational Autoencoder loss

$$D_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2) \,||\, \mathcal{N}(\mathbf{0}, \mathbf{I})) = \frac{1}{2} \sum_{j=1}^{M} \left(\mu_j^2 + \sigma_j^2 - \ln \sigma_j^2 - 1\right)$$

KL automatically prevents $\sigma \to 0$



Choosing $\mathcal{N}$ gives specific loss:

$$\ell(\mathbf{w}, \boldsymbol{\omega}) = \sum_{i=1}^{N} \frac{1}{2} \left\|\mathbf{x}_i - \tilde{\mathbf{x}}_i\right\|^2 + \frac{\beta}{2} \left(\left\|\boldsymbol{\mu}_i\right\|^2 + \left\|\boldsymbol{\sigma}_i\right\|^2 - \sum_{j=1}^{M} (\ln \sigma_{ij}^2 + 1) \right)$$

Backpropagate through "$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$"

- VAE performs a sampling operation between encoding and reconstruction.
- How do we "backpropagate" through this step?
  - How would we get $\frac{\partial \ell}{\partial \mu_j}$ and $\frac{\partial \ell}{\partial \sigma_j}$ and thereby $\nabla_{\mathbf{w}} \ell$?
- **<u>Reparameterization trick</u>:** rewrite $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ as

```
mu, sigma = f(x)
eps = sample_normal(0, 1)
z = mu + sigma*eps
x_recon = g(z)
```

$$\text{step 1: } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

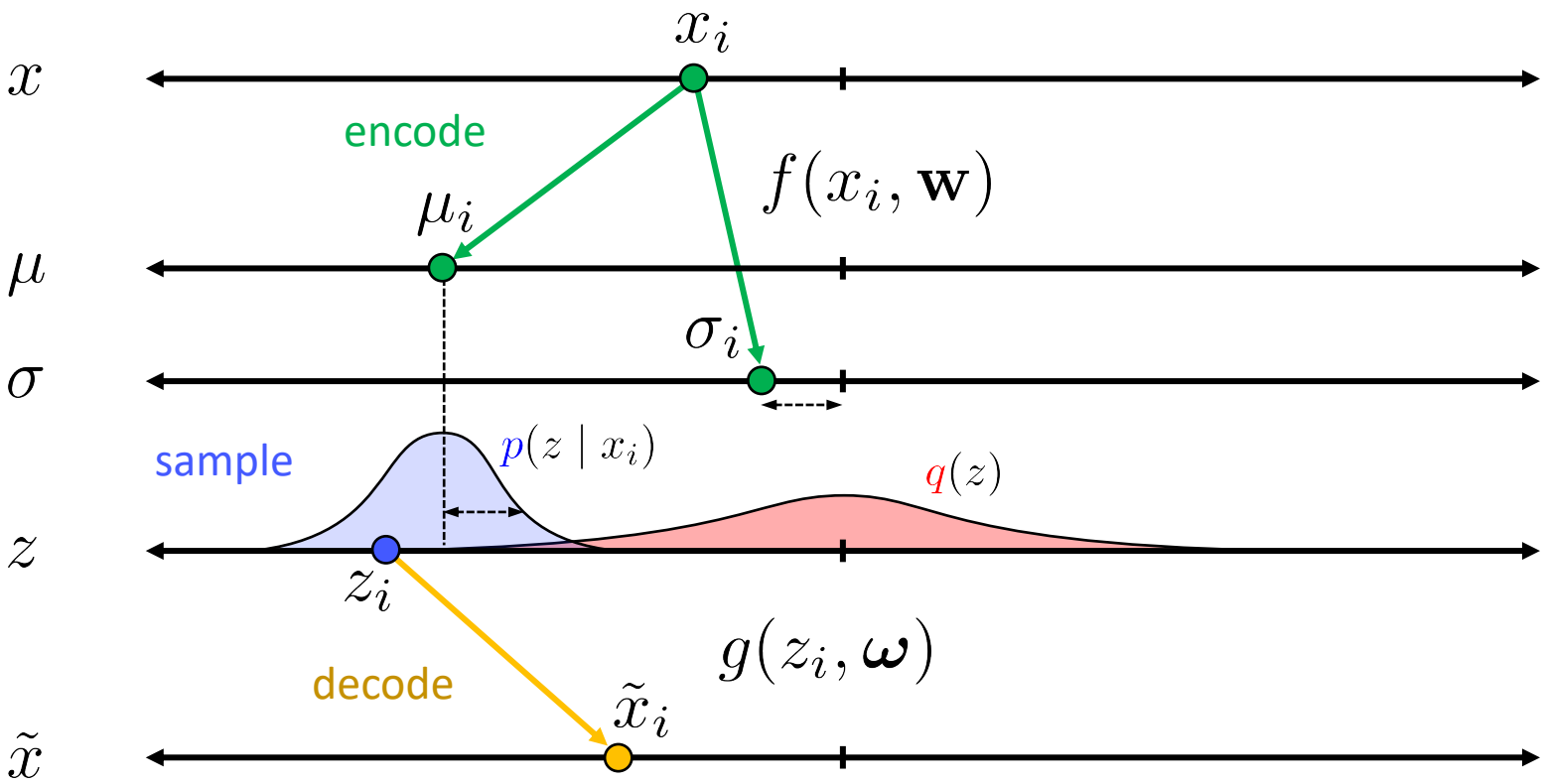
$$\text{step 2: } \mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$$

(symbol for elementwise multiplication)

gradients can "flow" to the encoder, through standard multiple-add ops

treated as constant

Example 1-1-1 linear VAE



$$\ell(\mathbf{w}, \boldsymbol{\omega}) = \sum_{i=1}^{N} \frac{1}{2}(x_i - \tilde{x}_i)^2 + D_{\mathrm{KL}}\big(\mathcal{N}(\mu_i, \sigma_i^2) \,||\, \mathcal{N}(0, 1)\big)$$

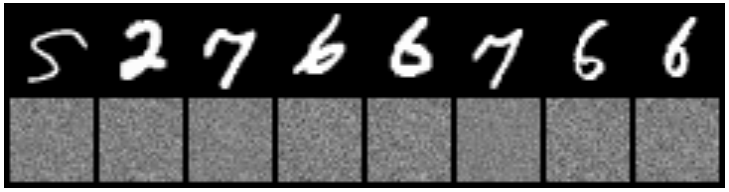
VAEs are generative models

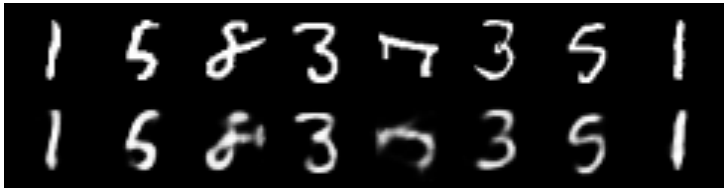
before training

after 10 epochs

$\mathbf{x}_i$

$\tilde{\mathbf{x}}_i$

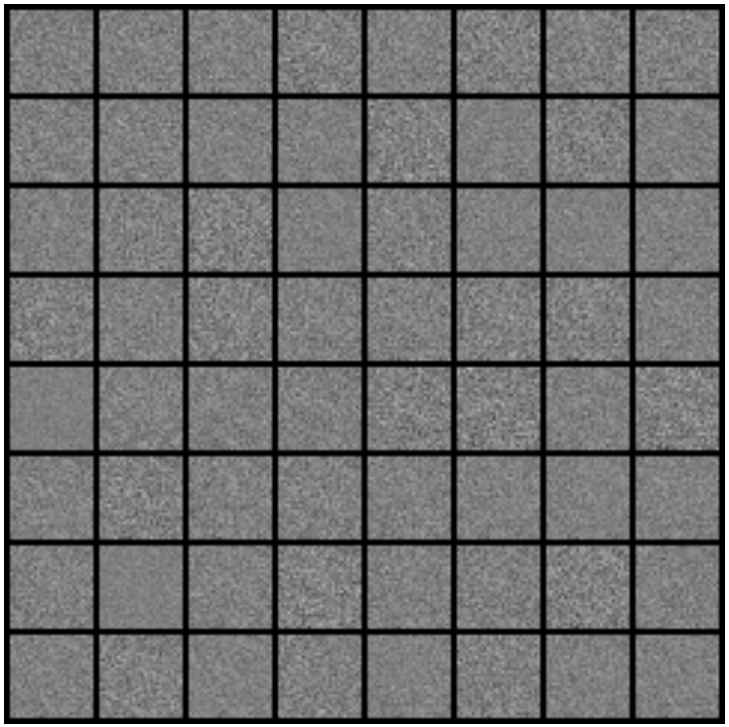


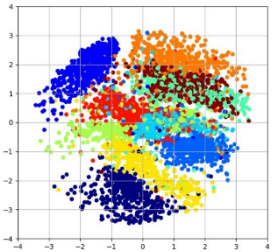


samples

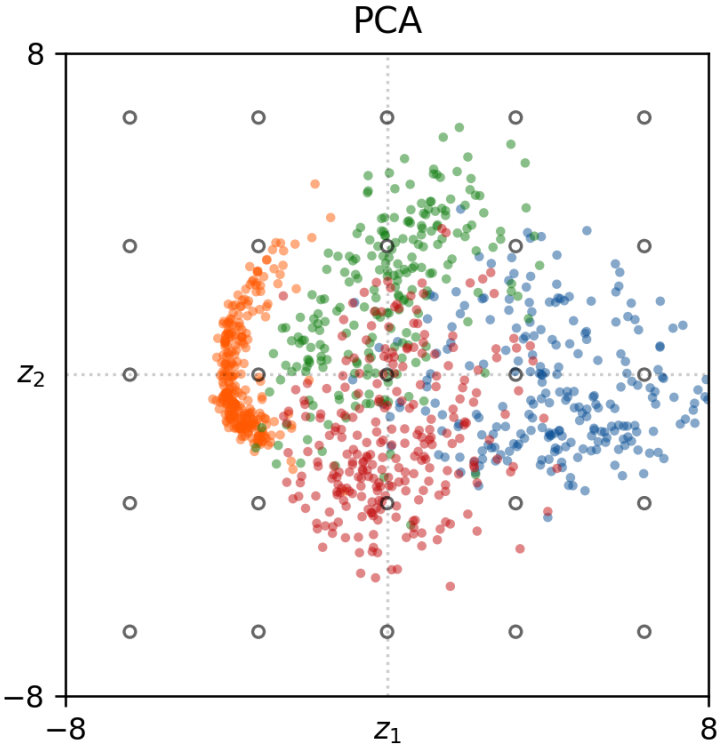
$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

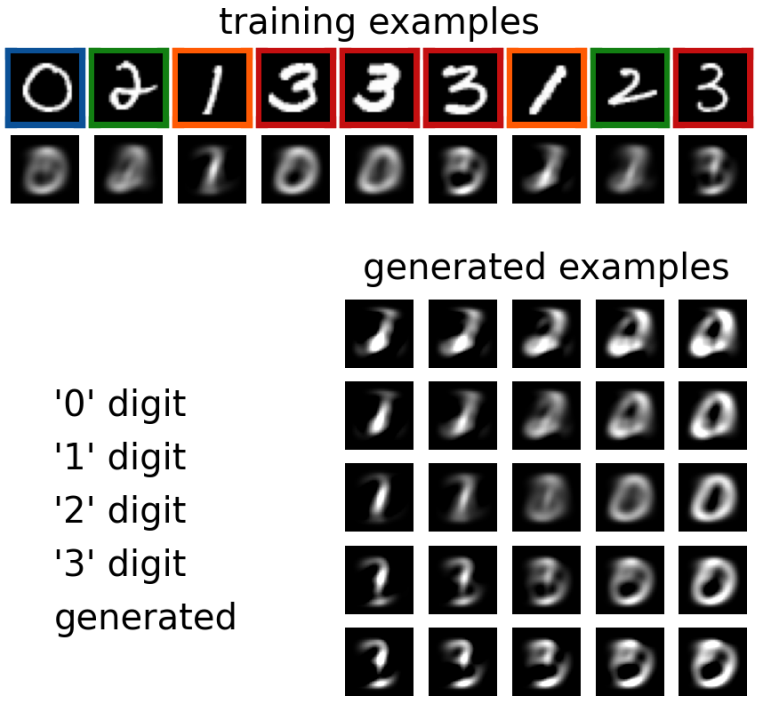
$\tilde{\mathbf{x}} = g(\mathbf{z}, \boldsymbol{\omega})$

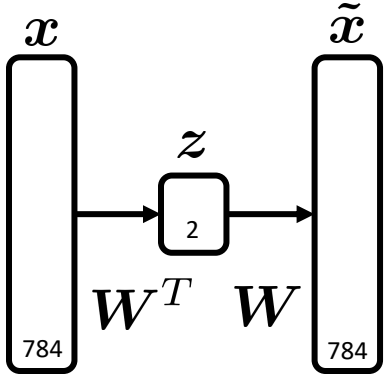




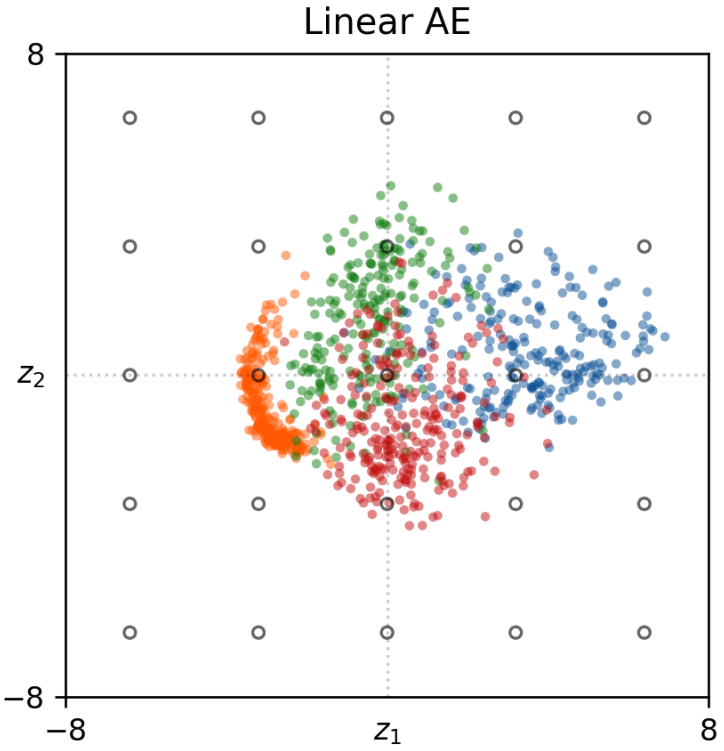
Example: PCA

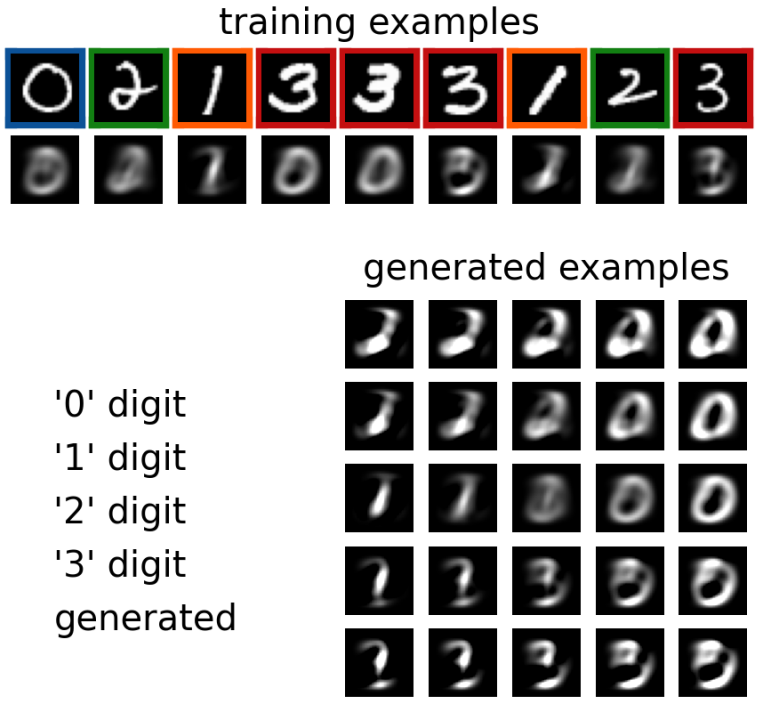

PCA
8
z_2
−8
−8
z_1
8

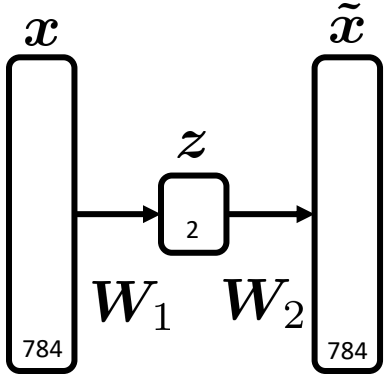

training examples
generated examples
'0' digit
'1' digit
'2' digit
'3' digit
generated


$\boldsymbol{x}$
$\tilde{\boldsymbol{x}}$
$\boldsymbol{z}$
2
$\boldsymbol{W}^T$
$\boldsymbol{W}$
784
784

Example: Linear Autoencoder

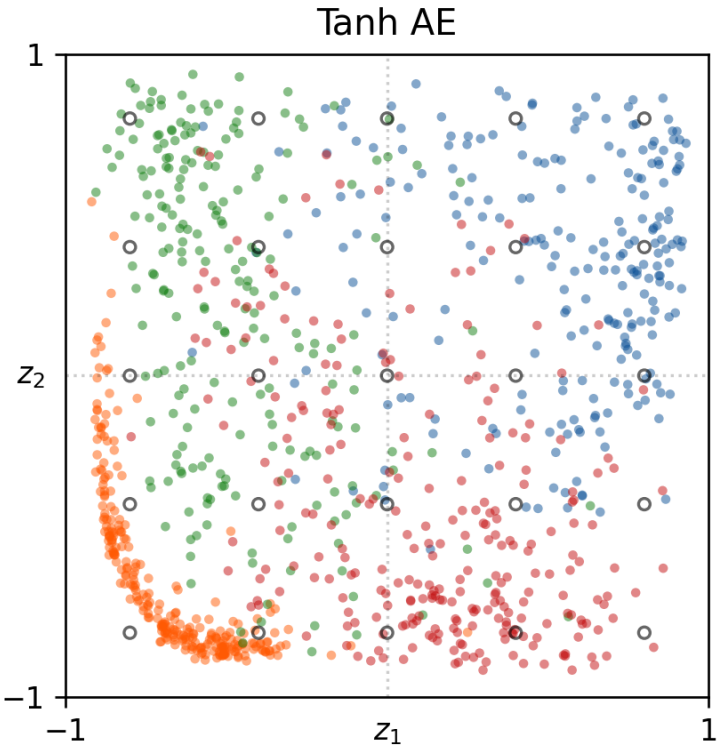


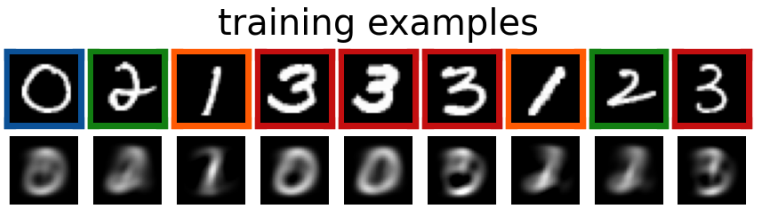


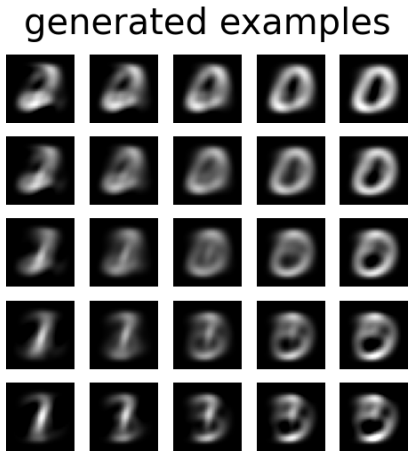


(this slide is a video)

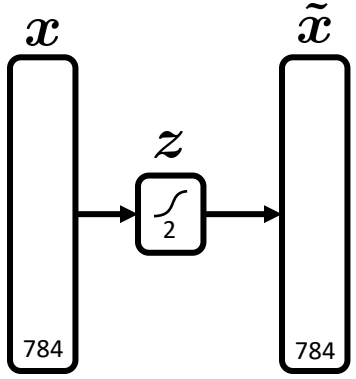
Example: Tanh Autoencoder





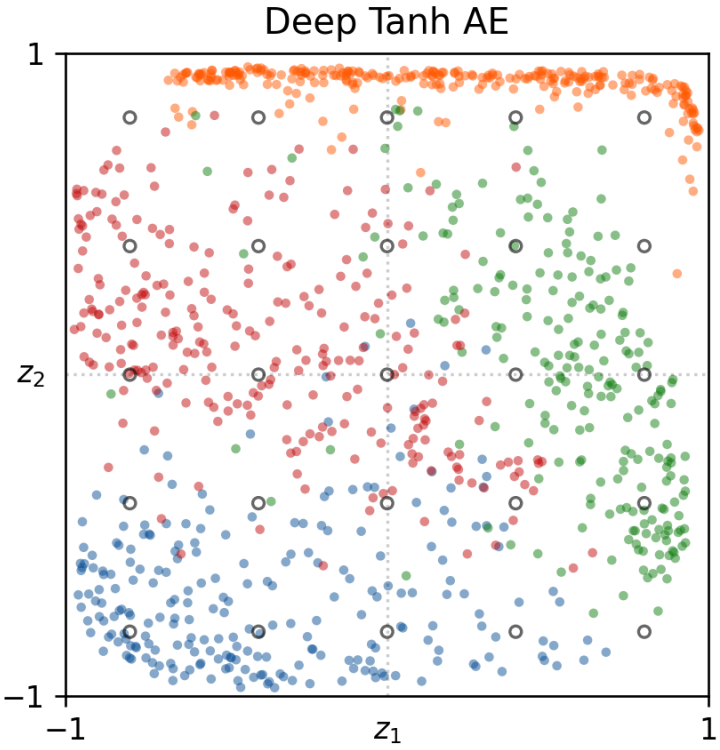


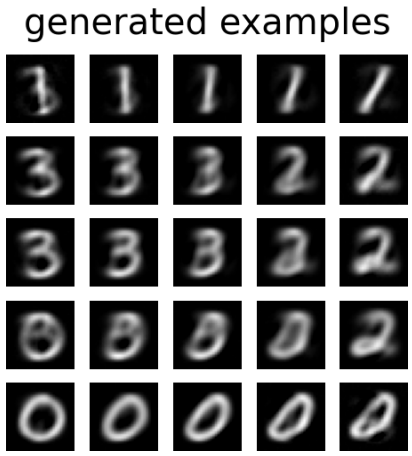
'0' digit
'1' digit
'2' digit
'3' digit
generated

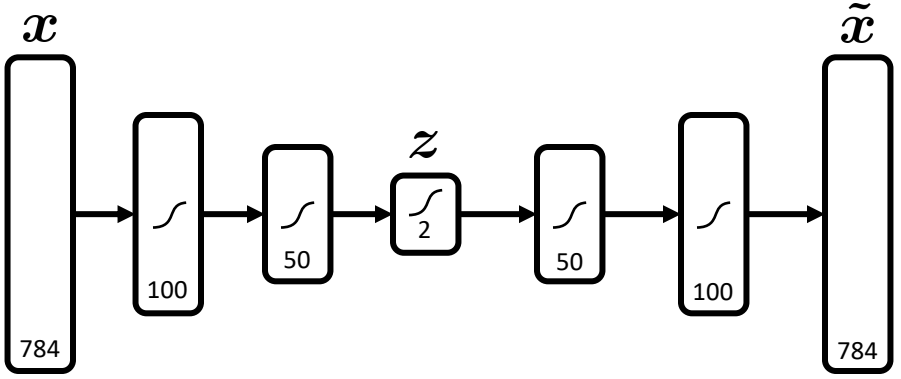


(this slide is a video)

Example: Deep Tanh Autoencoder

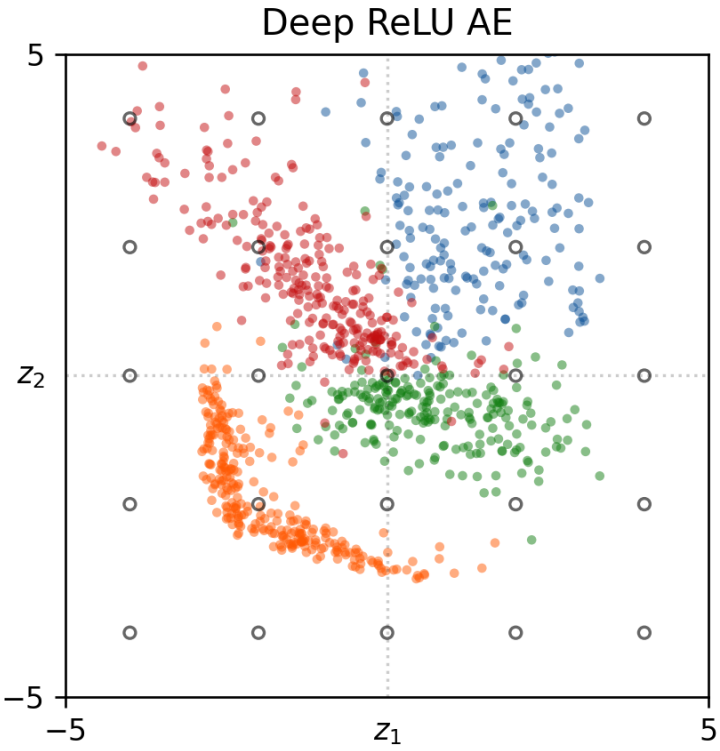


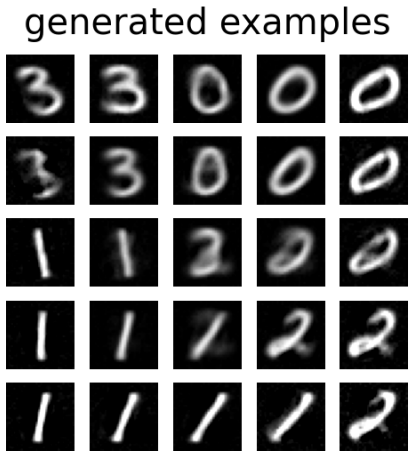




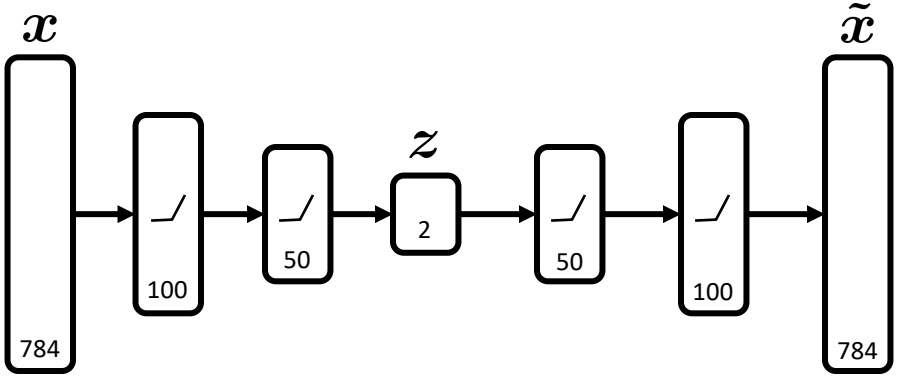
(this slide is a video)

Example: Deep ReLU Autoencoder



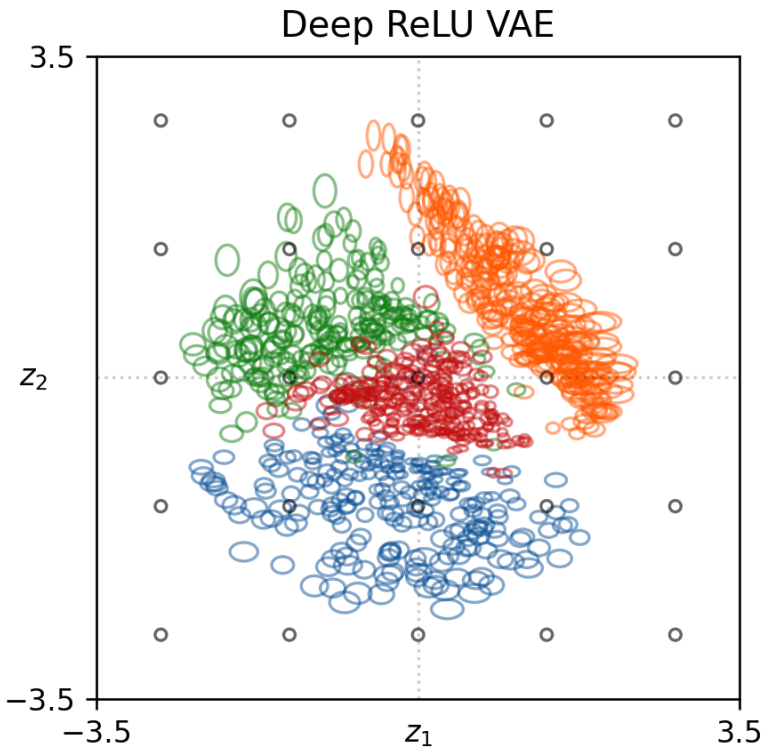


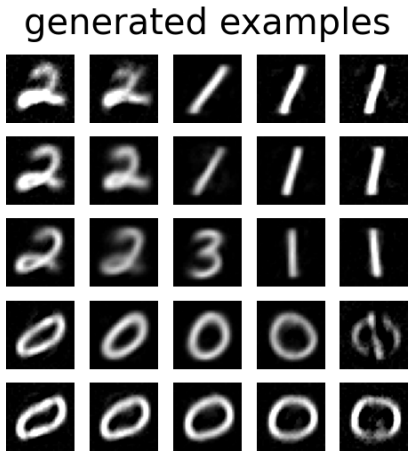
'0' digit
'1' digit
'2' digit
'3' digit
generated



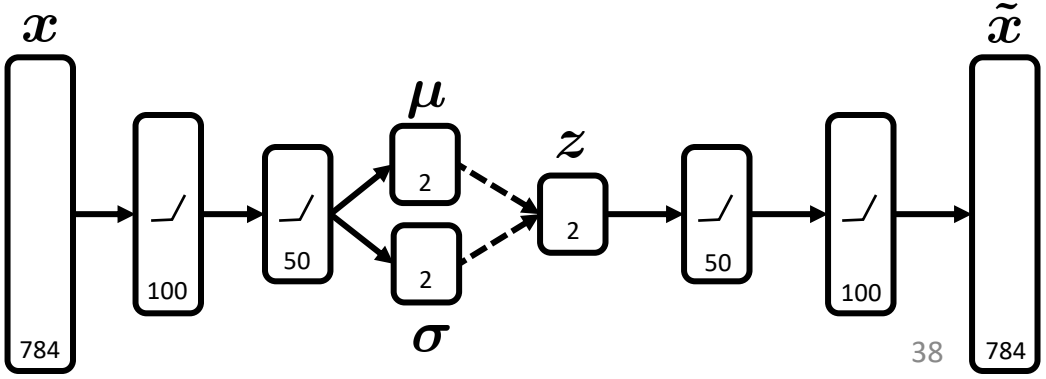
(this slide is a video)

Example: Deep ReLU Variational AE



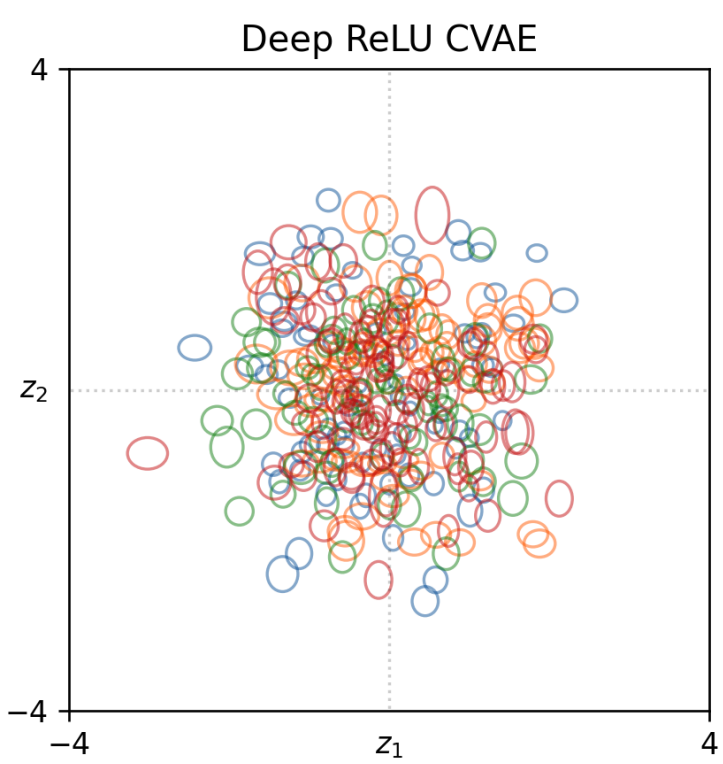


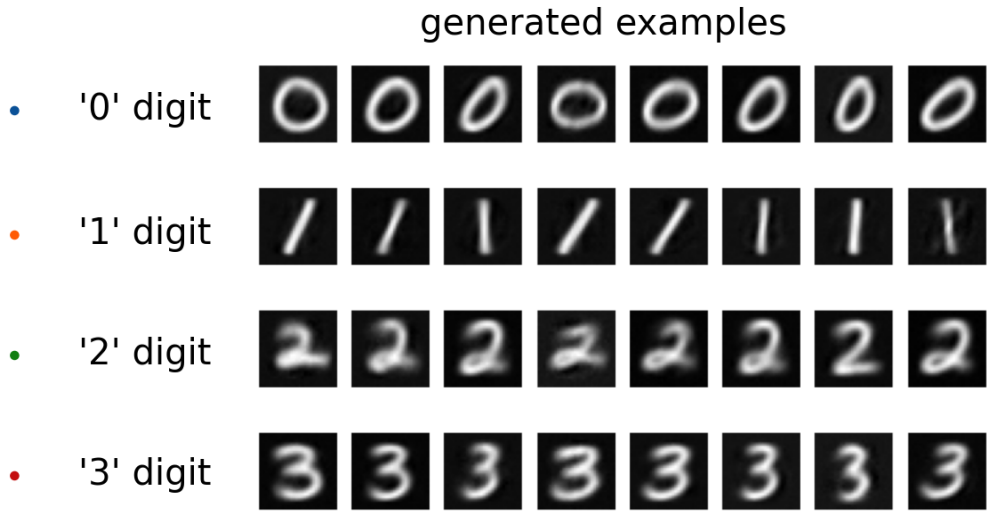
- '0' digit
- '1' digit
- '2' digit
- '3' digit
- generated



(this slide is a video)

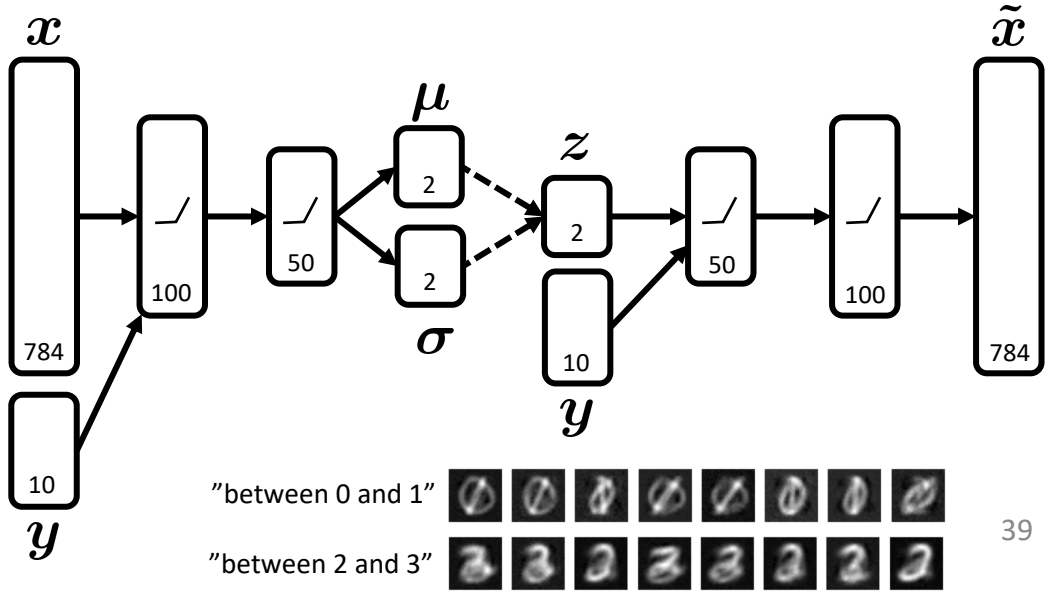
Deep ReLU *Conditional* VAE



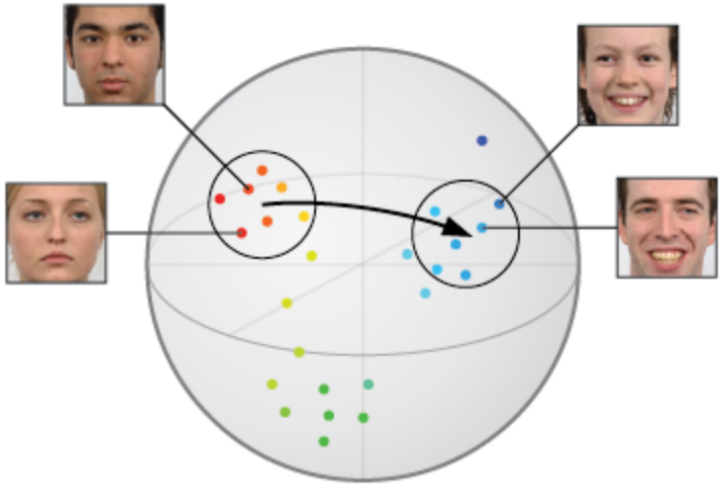


The encoder and decoder both take the class label as input, *i.e.*, they are *conditioned* on class label.

Latent codes *appear* to overlap, but in (2+10)-dimensional space they do not!



Example: VAE to encode faces



After training, can label some data as "not smiling" and other data as "smiling" and compute a "smile vector" (a direction in latent space) by subtracting the average latent codes…

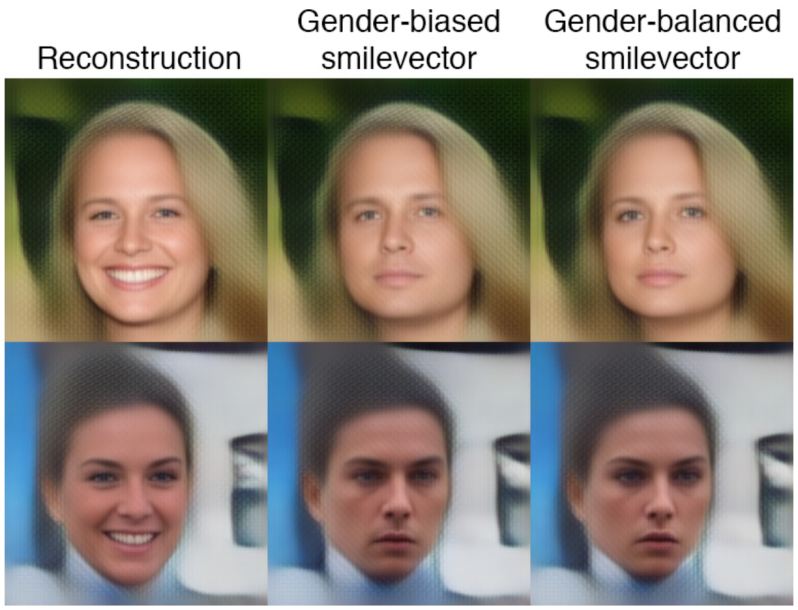


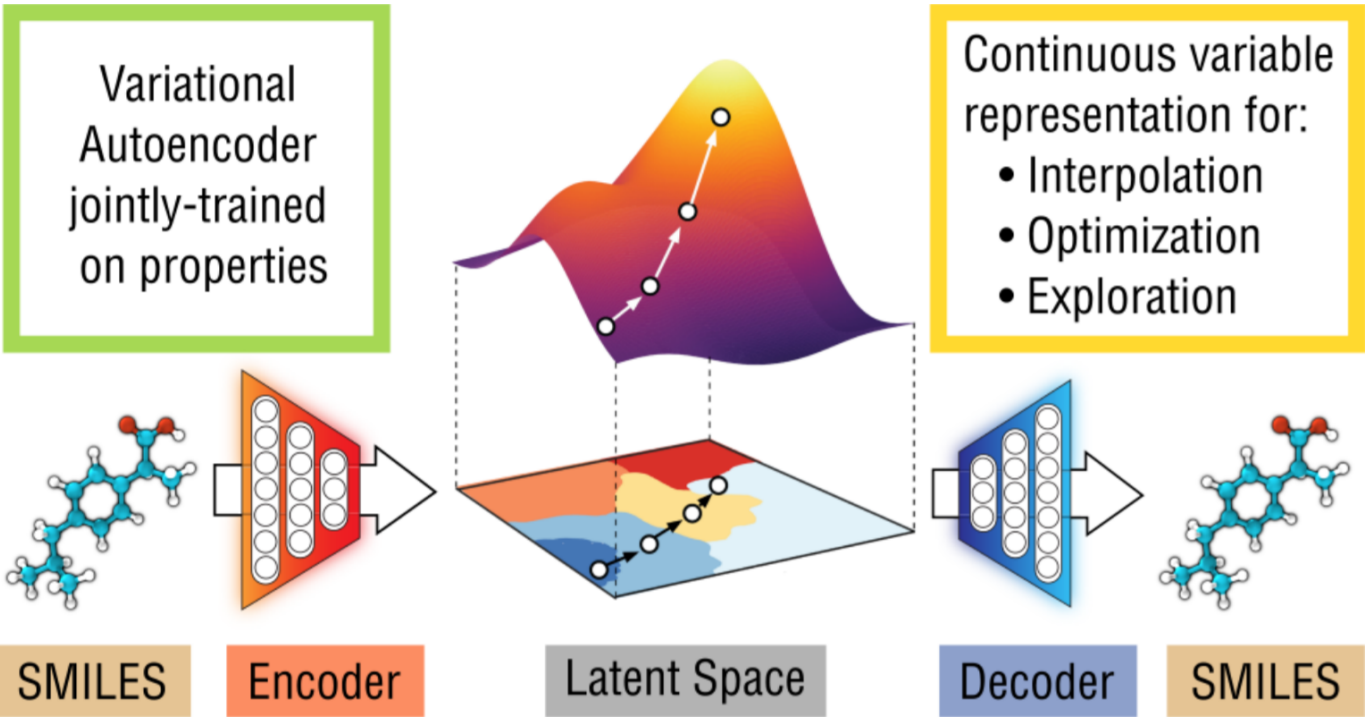
Figure 7: Initial attempts to build a smile vector suffered from sampling bias. The effect was that removing smiles from reconstructions (left) also added male attributes (center). By using replication to balance the data across both attributes before computing the attribute vectors, the gender bias was removed (right). (model: VAE from Lamb 16 on CelebA)

Image credit: Tom White https://arxiv.org/pdf/1609.04468.pdf

Example: VAE to design molecules

If we have training data with scores (e.g. molecular activity scores), map them to latent space, then search for unseen codes that are predicted to have higher score.

If we "decoded" that imaginary molecule, would it turn out to have higher activity than any of the training cases?



https://pubs.acs.org/doi/abs/10.1021/acscentsci.7b00572