UNSUPERVISED LEARNING

INF5860 — Machine Learning for Image Analysis

Ole-Johan Skrede

18.04.2018

University of Oslo

Messages

- Mandatory exercise 3 is hopefully out next week.
- No lecture next week.
- But there will be group sessions.

Outline

- Introduction and motivation
- Repetition / background
  - K nearest neighbours, k-means clustering
  - Principal component analysis
  - Independent component analysis
- t-SNE
- Autoencoders, variational autoencoders

INTRODUCTION AND MOTIVATION

Supervised learning

- Given a training set with pairs of inputs x and corresponding desired outputs y

$$\Omega_{\text{train}} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$$

- Create a function f that "approximates" this mapping

$$f(x) \approx y, \quad \forall (x, y) \in \Omega_{\text{train}}$$

- Hope that this generalises well to unseen examples, such that

$$f(x) = \hat{y} \approx y, \quad \forall (x, y) \in \Omega_{\text{test}}$$

  where Ω_{test} is a set of relevant unseen examples.
- Hope that this is also true for all unseen relevant examples.

Unsupervised learning

- In contrast with supervised learning, we have *no* labeled data points in unsupervised learning.
- Since there is no “ground truth”, there is no accuracy evaluation in the supervised sense.
- Applications
  - Data clustering
  - Anomaly detection
  - Signal generation
  - Signal compression

Semi-supervised learning

- We have *some* labeled data
- Usually a majority of unlabeled data
- Can be thought of as supervised learning extended to utilise unlabeled data
- Will not be covered today

Methods

What we *will* cover today

- K-means clustering (background)
- Principal component analysis (PCA) (background)
- t-SNE
- Autoencoders
- Variational autoencoders

What we *will not* cover today

- Independent component analysis (ICA)
- Matrix factorization and decomposition
- Expectation-maximization (EM) algorithm
- Generative-adverserial networks (GAN) (next lecture)

CLUSTERING

Data clustering

- Grouping together data based on some similarity metric
- Data points within the same group (cluster) will be more similar to each other than to data points outside the group
- Many different versions of clustering

Connectivity-based clustering

- Also called hierarchical clustering
- See figures for example with the L_2 distance metric
- Different level thresholds yields different clusters

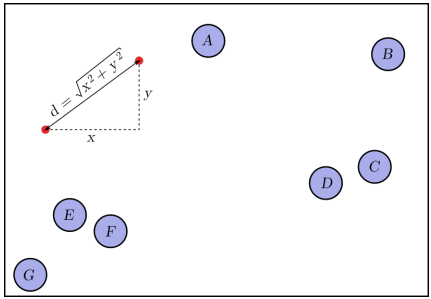


Figure 1: Raw data

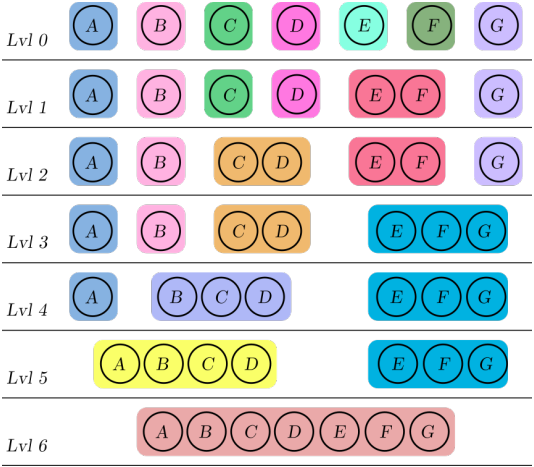


Figure 2: Bottom up (agglomerative) hierarchy of clusters

Graph clustering — cliques

- A clique is a set of nodes
- A node in a clique shares an edge with all other nodes in the clique
- Can have cliques of different sizes
- Useful in applications such as random fields

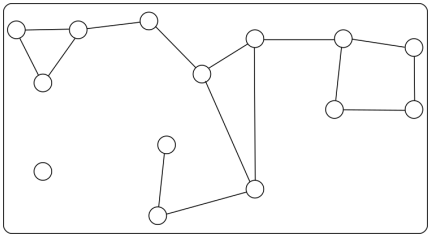


Figure 3: Undirected graph

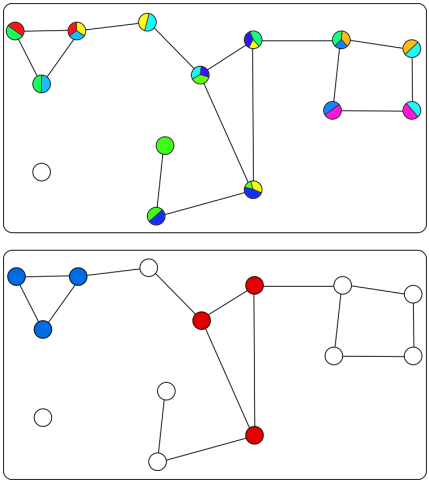


Figure 4: Top: Cliques with 2 members. Bottom: Cliques with 3 members. Nodes with multiple colors belong to more than one clique.

Centroid-based clustering

- Clusters are represented by a central vector
- Example: K-means clustering

K-means clustering

- Conseptually simple clustering algorithm
- We want to partition a set of data $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ into k clusters.
- $x^{(i)} \in \mathbb{R}^n, i = 1, \ldots, m$
- With some distance norm $||\cdot||$ the procedure is
  1. Initialize at random k cluster centroids (or means) $\mu_j \in \mathbb{R}^n, j = 1, \ldots, k$
  2. Repeat until convergence
     2.1 For every example $x^{(i)}, i = 1, \ldots, m$, let

$$c^{(i)} = \arg\min_j ||x^{(i)} - \mu_j||.$$

     2.2 For every centroid $\mu_j, j = 1, \ldots, k$, set

$$\mu_j = \frac{\sum_{i=1}^m I[c^{(i)} = j] x^{(i)}}{\sum_{i=1}^m I[c^{(i)} = j]},$$

     where the *Iverson bracket* is defined as

$$I[a = b] = \begin{cases} 1, & \text{if} \quad a = b, \\ 0, & \text{if} \quad a \neq b \end{cases}.$$

- Minimizes the objective function

$$J(c, \mu) = \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||$$

- Not guaranteed to find a *global minimum*
- Common to run the algorighm several times with different initializations, and then pick the run with the smallest value of J
- The k-means clustering algorithm partitions the feature space into *Voronoi cells*

K-means clustering — example

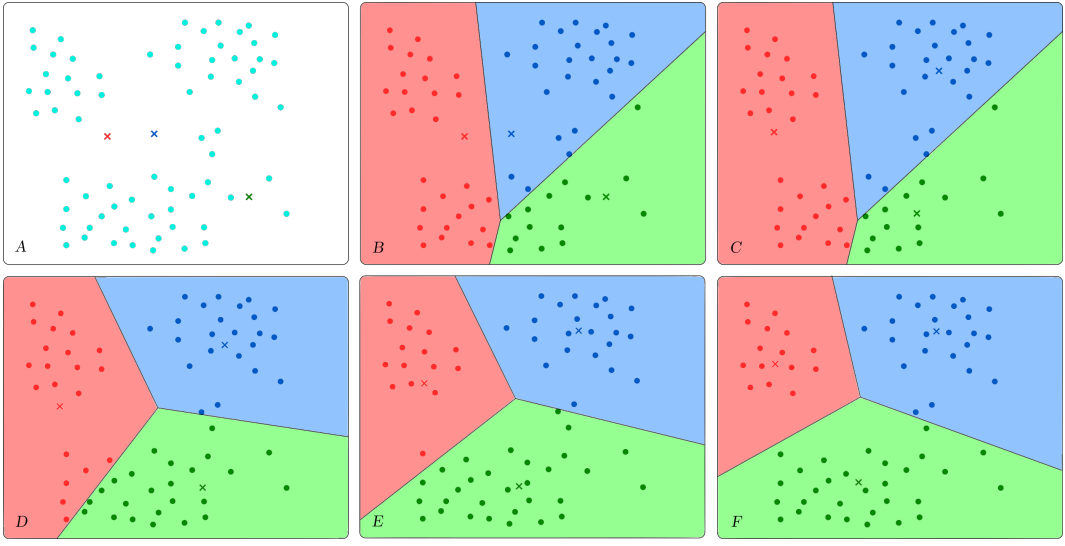


Figure 5: A: Initialize centroids. B: Assign points to clusters. C: Move centroids. D, E, F: Assign points to clusters and move centroids. No change after F (convergence).

PCA

Principal Component Analysis (PCA)

- Reducing the dimensionality of a dataset of correlated variables
- Retaining as much as possible of the variance present in the dataset

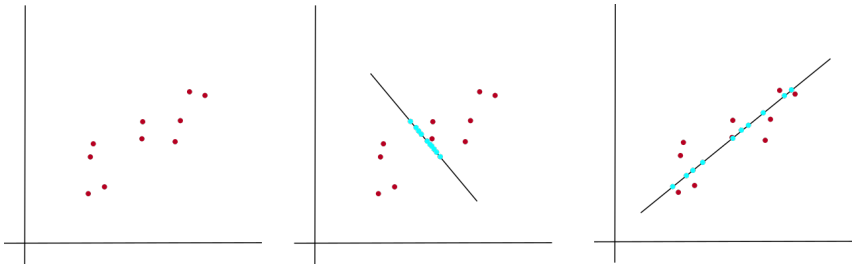


Figure 6: Representing 2D data as 1D

PCA — DERIVATION OUTLINE

- Let $X \in \mathrm{R}^{n_d}$ be a random vector
- We are looking for a set of *uncorrelated* variables Y_k which we will call the *principal components* of X
- The first component, Y_1, will account for most of the variance in X
- The second component, Y_2, will account for most of the variance in X, conditionied on being uncorrelated with Y_1
- The third component, Y_3, will account for most of the variance in X, conditioned on being uncorrelated with *both* Y_1 and Y_2
- We continue untill we have $n_p << n_d$ principal components that account for most of the variance in X

PCA — FIRST PRINCIPAL COMPONENT

- Let $Y_1 \in \mathbb{R}$ be some linear combination of the elements in X

$$Y_1 = \sum_{i=1}^{N_d} a_{1i} X_i = a_1^\intercal X,$$

- This random variable has variance

$$Var[Y_1] = Var[a_1^\intercal X] = a_1^\intercal \Sigma a_1.$$

- Here, Σ is the covariance matrix of X with elements

$$\Sigma_{ij} = Cov(X_i, X_j)$$

- We want to maximize the variance of Y_1
- In order to achieve finite solutions, we constrain the optimization on

$$a_1^\intercal a_1 = 1$$

- It turns out that, for $k = 1, \ldots, n_p$, a_k well be an eigenvector of Σ corresponding to the kth largest eigenvalue λ_k

- For a dataset with n_s samples $\{x_{i1}, \ldots, x_{in_s}\}$ for all features $i = 1, \ldots, n_d$, the elements in the covariance matrix can be estimated as

$$\hat{\Sigma}_{ij} = \frac{1}{n_s - 1} \sum_{q=1}^{n_s} (x_{iq} - \hat{\mu}_i)(x_{jq} - \hat{\mu}_j),$$

- Here $\hat{\mu}_i$ is the sample mean of the ith feature

$$\hat{\mu}_i = \frac{1}{n_s} \sum_{q=1}^{n_s} x_{iq}$$

- Arranginging the feature samples and sample means into vectors of size n_d

$$x_q = [x_{1q}, \ldots, x_{n_d q}]^\mathsf{T}$$
$$\hat{\mu} = [\hat{\mu}_1, \ldots, \hat{\mu}_{n_d}]^\mathsf{T}$$

- With this, the estimate of the covariance matrix can be written as

$$\hat{\Sigma} = \frac{1}{n_s - 1} \sum_{q=1}^{n_s} (x_q - \hat{\mu})(x_q - \hat{\mu})^\mathsf{T}.$$

PCA — Optimizing the variance

- We use the technique of *Lagrangian multipliers* to incorporate the unit length constraint
- This means that we are going to maximize the expression

$$J(a_1) = a_1^\intercal \Sigma a_1 - \lambda(a_1^\intercal a_1 - 1).$$

- Computing the gradient of J w.r.t. a_1, and setting it equal to zero, yields

$$\Sigma a_1 - \lambda a_1 = 0,$$

or

$$(\Sigma - \lambda I) a_1 = 0,$$

where I is the $n_d \times n_d$ identity matrix.

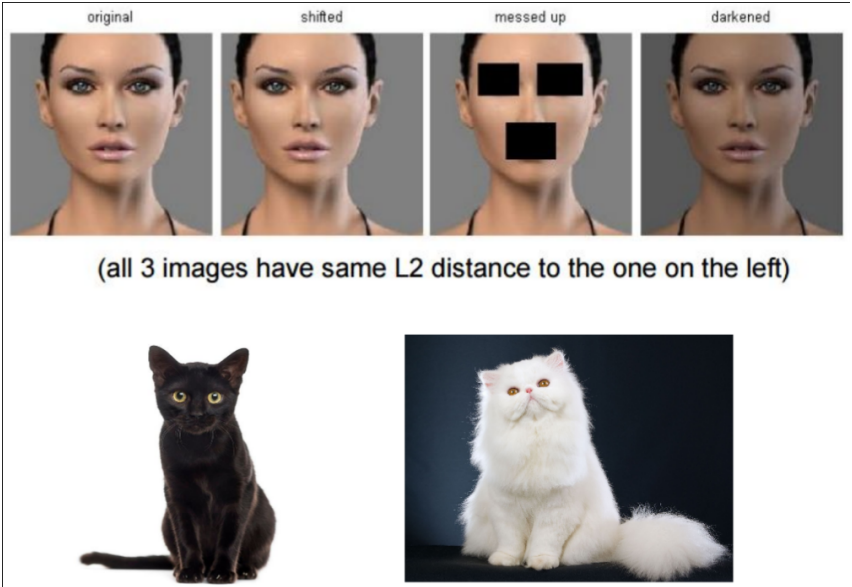
PCA — Optimizing the variance

- From our last expression
$$(\Sigma - \lambda I)a_1 = 0,$$
we see that λ is an eigenvalue of Σ, and a_1 is the corresponding eigenvector.
- Furthermore, λ is the largest eigenvalue
- This is because maximizing the variance subject to the constraint of unit length coefficients is equivalent to choosing the largest eigenvalue
$$\begin{aligned} a_1^\intercal \Sigma a_1 &= a_1^\intercal \lambda a_1 \\ &= \lambda a_1^\intercal a_1 \\ &= \lambda. \end{aligned}$$
- In general, the kth principal component of X is
$$a_k^\intercal X$$
where a_k is the eigenvector of the covariance matrix Σ of X, corresponding to the kth largest eigenvalue λ_k

PCA — Applications

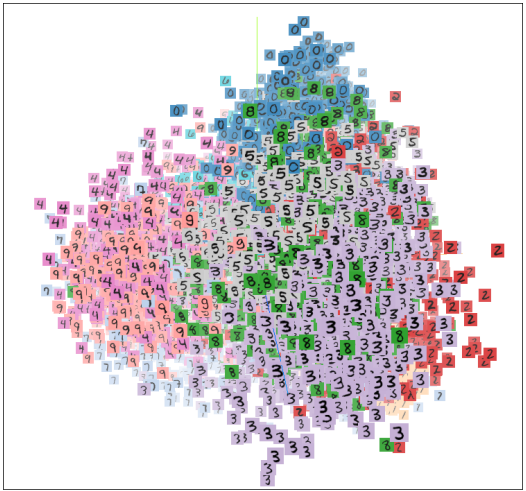
- Dimensionality reduction
- Preprocessing in supervised learning: acts as a regularizer
- Noise reduction

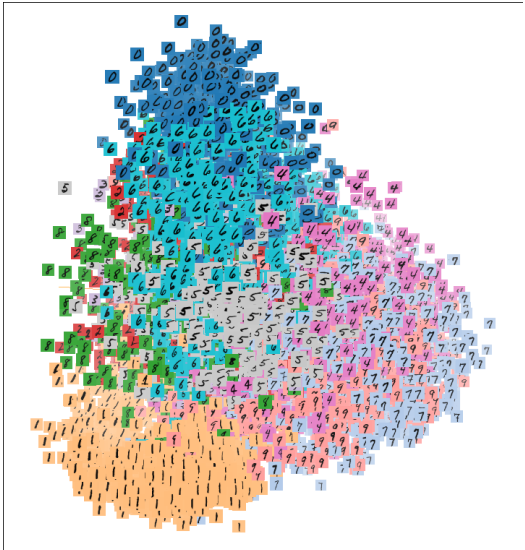
Problems with image data



MNIST clustering with PCA

Explains about 26% of the variance. Not very suited.





T-SNE

Stochastic Neighbour Embedding (SNE)

- Introduced by Geoffrey Hinton and Sam Roweis in 2003
- PDF: http://papers.nips.cc/paper/2276-stochastic-neighbor-embedding.pdf
- A stochastic dimensionality reduction method
- Transforms high-dimensional ($h - d$) data points to low-dimensional ($l - d$) data ponts
- Aims to preserve neighbourhood identity between data points
- Similar (close) $h - d$ points should also be similar (close) in the $l - d$ representation

SNE — overview

- The high-dimensional points have some dimension h
- The low-dimensional points have some dimension $l << h$, which is determined manually
- For each point i, we are going to define two distributions:
  - $p_{X_i|X_j}(x_i|x_j)$: The probability that x_i and x_j are "neighbours", given the location of x_j
  - $q_{Y_i|Y_j}(y_i|y_j)$: The probability that y_i and y_j are "neighbours", given the location of y_j
- We will use the following shorthand
  - $p_{i|j} := p_j(x_i) := p_{X_i|X_j}(x_i|x_j)$
  - $q_{i|j} := q_j(y_i) := p_{Y_i|Y_j}(y_i|y_j)$
- We are then going to define a similarity measure between these distributions
- The low-dimensional representations will be altered such as to minimize this distribution similarity

SNE — high-dimension neighbour probability

- Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of points with h dimensions, $x_i \in \mathbb{R}^h, i = 1, \ldots, n$
- These are the data points which we want to represent in a lower dimensional space
- We model the data point distribution as a asymmetric Gaussian

$$p_{i|j} = \frac{\exp\left\{-\frac{||x_i - x_j||^2}{\sigma_j^2}\right\}}{\sum_{k \neq j} \exp\left\{-\frac{||x_k - x_j||^2}{\sigma_j^2}\right\}}$$

- The scaling parameter σ_j can be set manually
- More often adjusted with a *perplexity* parameter
- Often found by a binary search such that the entropy of the distribution over its neighbours is equal to $\log k$
- k is a *perplexity* parameter that is set by hand
- See e.g. https://distill.pub/2016/misread-tsne/ how to interpret t-SNE results

SNE — PERPLEXITY

- We want a larger σ in sparse areas
- We want a smaller σ in dense areas
- σ is often found by a binary search such that the entropy of the distribution over its neighbours is equal to $\log k$

$$H(p_j) = -\sum_i p_{i|j} \log p_{i|j}$$

- k is a *perplexity* parameter that is set by hand
- Perplexity can be interpreted as a measure of how many neighbours we want to influence a point

SNE — LOW-DIMENSION NEIGHBOUR PROBABILITY

- Let $\mathcal{Y} = \{y_1, \ldots, y_n\}$ be a set of points with l dimensions, $x_i \in \mathbb{R}^l, i = 1, \ldots, n$
- These are the lower-dimensional data points corresponding to X, so $u >> v$
- We choose a Gaussian neighbourhood with fixed variance $\sigma^2 = 1/2$

$$q_{i|j} = \frac{\exp\{-||y_i - y_j||^2\}}{\sum_{k \neq j} \exp\{-||y_k - y_j||^2\}}$$

Repetition: Kullback-Leibler divergence

- The Kullback-Liebler divergence over a discrete random variable $\mathcal{X}$

$$D_{KL}(p_{\mathcal{X}}||q_{\mathcal{X}}) = \sum_x p_{\mathcal{X}}(x) \log \frac{p_{\mathcal{X}}(x)}{q_{\mathcal{X}}(x)}$$

- Measures the distance between two probability distributions $p_{\mathcal{X}}$ and $q_{\mathcal{X}}$ over the same set of events, modeled with the random variable $\mathcal{X}$.
- Expectation of logarithmic difference between p and q when expectation is taken w.r.t. p.
- Measures the amount of information that is lost when using q to approximate p.
- It is non-negative
- Zero for $p = q$
- Increasing for "increasing difference" between p and q.

- We want to measure the similarity between p_j and q_j
- This is done by summing the KL-divergence between the original (p_j) and the induced (q_j) distributions over neighbours of each data point

$$\begin{aligned} C &= \sum_j \sum_i p_{i|j} \log \frac{p_{i|j}}{q_{i|j}} \\ &= \sum_j D_{KL}(p_j || q_j) \end{aligned}$$

- Large cost of confusing a small distance in the high-dimensional space with a large distance in the low-dimensional space (small $p_{i|j}$ and large $q_{i|j}$)
- Larger cost of confusing a large distance in the high-dimensional space with a small distance in the low-dimensional space (large $p_{i|j}$ and small $q_{i|j}$)

SNE — discussion

- The cost can be minimized with stochastic gradient descent
- Emphazis of local objects
- Keeps nearby points in $h - d$ nearby in $l - d$
- Also keeps distant points in $h - d$ relatively far apart in $l - d$
- Drawback: Can be difficult to optimize
- Drawback: Tendency to crowd $l - d$ representations at the center of the map

t-distributed Stochastic Neighbour Embedding (t-SNE)

- A variant of the SNE method
- Introduced by Laurens van der Maaten and Geoffrey Hinton
- PDF:
  https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf
- An improvemet over SNE
  - Much easier to optimize
  - Significantly better visualization
- Two major differences between t-SNE and SNE
  - *Symmetric* Gaussian point similarity distribution for the $h - d$ data points
  - Student-t point similarity distribution for the $l - d$ map points

t-SNE — Crowding problem

- Standard SNE (and other similar methods) suffer from what is known as the *crowding problem*
- Too many map points are placed near the center of the map
- Intuition of why:
  - In h dimensions, the volume of a sphere centered at x scales as r^h, where r is the "radius" of the sphere
  - In $l << h$ dimensions, the volume of a corresponding sphere will be much smaller
  - Less room to place points while preserving natural cluster structure
  - This causes a crowding of points that should have been modeled with a larger distance
- This can be leviated by forcing moderately distant data points to be placed far apart

t-SNE — Symmetric cost

- Standard SNE used the asymmetric KL-divergence

$$C = \sum_j \sum_i p_{i|j} \log \frac{p_{i|j}}{q_{i|j}}$$
$$= \sum_j D_{KL}(p_j || q_j)$$

- This is assymmetric because $p_{i|j} \neq p_{j|i}$ and $q_{i|j} \neq q_{j|i}$
- Because of this, different types of errors in the pairwise distances in the map are weighted differently
- In particular
  - The cost of representing distant data points as close map points is smaller than
  - The cost of representing close data points as distant map points
- A symmetric cost could ease optimization, and leviate the crowding problem

Symmetric distribution in the low-dimensional space

- In a symmetric SNE, we use a *joint* Gaussian to model the similarity between map points y_i and y_j

$$q_{ij} = \frac{\exp\{-||y_i - y_j||^2\}}{\sum_{k \neq l} \exp\{-||y_k - y_l||^2\}}$$

- Compare this with the *conditional* Gaussian we used in SNE

$$q_{i|j} = \frac{\exp\{-||y_i - y_j||^2\}}{\sum_{k \neq j} \exp\{-||y_k - y_j||^2\}}$$

- The difference is quite subtle and is present in the sum in the denominator
- In the asymmetric version, we sum over the difference between all points $y_k, k \neq j$ and the reference point y_j
- In the symmetric version, we sum over the difference between *all* unequal points y_k and y_l
- Note that this is not what is used in t-SNE, we will come back to that in two slides

t-SNE — Symmetric distribution in the high-dimensional space

- We could have used the same distribution for the data points x_j

$$p_{ij} = \frac{\exp\{-||x_i - x_j||^2\}}{\sum_{k \neq l} \exp\{-||x_k - x_l||^2\}}$$

- The problem is that for an outlier x_j, $||x_i - x_j||$ will be very large (and p_{ij} very small) for all points
- The placement of the corresponding map point y_j will have very little effect on the cost
- We can fix this by simply choosing

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

  where n is the number of data points
- With this, we ensure that $\sum_j p_{ij} < 1/2n$ for all data points x_i
- Hence, all points x_i are guaranteed to make significant contributions to the cost

- To mitigate the crowding problem, we want to give more weight to representing moderately distant data points as close map points
- The Student-t distribution with one degree of freedom is used

$$q_{ij} = \frac{\left(1 + ||y_i - y_j||^2\right)^{-1}}{\sum_{k \neq l} \left(1 + ||y_k - y_l||^2\right)^{-1}}$$

- Notice that it is symmetric $q_{ij} = q_{ji}$
- The Student-t distribution has a much heavier tail than the Gaussian distribution

AUTOENCODERS

Autoencoders — introduction

- An autoencoder is a neural network which purpose is to discover interesting representations of data
- The idea is to create identity mappings, that is, functions f such that $f(x) \approx x$ for some input x
- It is able to discover interesting representations by enforcing constraints on the network
- The method requires no labeled data, and is therefore unsupervised

- An autoencoder consist of an encoder g and an decoder h
- The encoder maps the input x to some representation z

$$g(x) = z$$

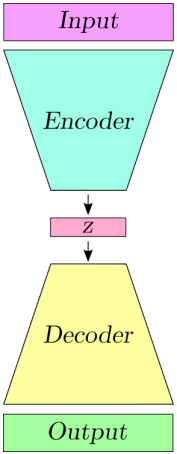
- We often call this representation z for the code
- The decoder maps this representation z to some output $\hat{x}$

$$g(x) = \hat{x}$$

- We want to train the encoder and decoder such that

$$\begin{aligned} f(x) &= h(g(x)) \\ &= \hat{x} \end{aligned}$$

  such that $\hat{x} \approx x$
- The loss is commonly either a cross-entropy loss or a mean



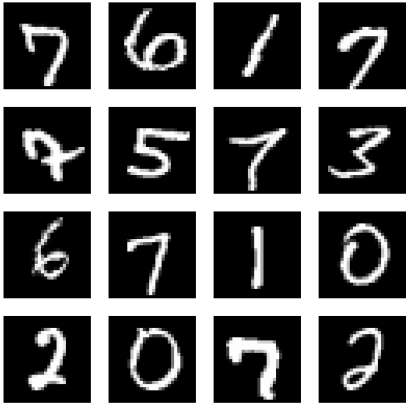
Autoencoders — variants

- Different network constraints leads to different representations z
- Reducing the dimensionality of the representation z
  - If x has d_x dimensions and z has d_z dimensions, and $d_x > d_z$
  - Most common way of constraining the network
- Sparse autoencoder
  - z can actually have a greater dimension than x
  - Only allowing a subset of the hidden units to fire at the same time
- Denoising autoencoder
  - Distorting the input x with some random noise
  - Leads to robust representations, resiliant to corrupted input

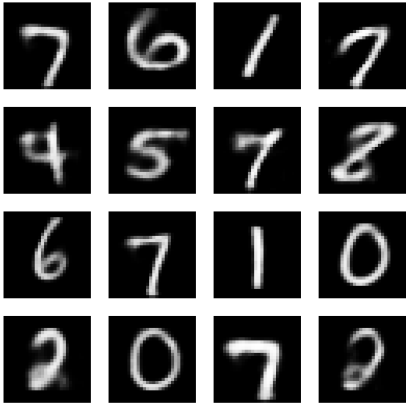
Compression autoencoder — MNIST example

- Encoder:
  - Input -> first hidden layer: fully connected 784 -> 128, relu
  - 1. hidden -> 2. hidden: fully connected, 128 -> 32, relu
- Decoder:
  - 2.hidden -> 3. hidden: fully connected 32 -> 128, relu
  - 3. hidden -> output: fully connected, 128 -> 784, sigmoid
- Binary cross entropy
- Adam optimizer

Compression autoencoder — MNIST example



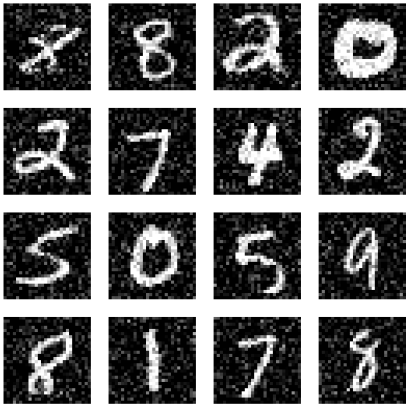
(a) Original

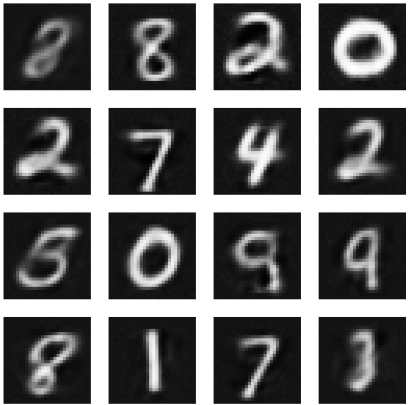


(b) Reconstructed

Denoising autoencoder — MNIST example

Same setup as for the compression autoencoder. Zero mean gaussian noise with standard deviation 0.1 is added to the input. The input values are clipped to lay in $[0, 1]$.





Sparse autoencoder

- We want to constrain the number of *active* nodes in the coding layer
- We can think of a node being active (or *firing*) if is
  - close to 1 for the sigmoid tanh activation functions
- We can think of a node being inactive
  - close to 0 for the sigmoid activation function
  - close to -1 for the tanh activation function
- We would like to constrain the nodes to be inactive most of the time

Sparse autoencoder

- Let $a_j^{[c]}(x^{(i)})$ be the activation in node j in the coding layer $[c]$ given an input $x^{(i)}$ to the network
- Then, activation for this node averaged over all m input examples is

$$\hat{\rho} = \frac{1}{m} \sum_{i=1}^{m} a_j^{[c]}(x^{(i)})$$

- We would like to limit this average activation by enforcing the constraint

$$\hat{\rho} = \rho$$

  for some predetermined *sparsity parameter* ρ
- Choosing a small ρ (e.g. 0.1) forces the activations to be small

- The way we enforce this constraint is to regularize the loss function

$$\mathcal{L} = \mathcal{L}_{\text{reconstruction}} + \beta \mathcal{L}_{\text{sparsity}}$$

  with some regularization strength $\beta \in \mathbb{R}$.
- We are going to use the KL-divergence between the distributions p and q as our sparsity loss

$$\mathcal{L}_{\text{sparsity}} = \sum_{j=1}^{n^{[c]}} D_{KL}(p||q_j)$$

  where $n^{[c]}$ is the number of nodes in layer $[c]$
- p will be a Bernoulli distribution with mean ρ for a node j
- q_i will be a Bernoulli distribution with mean $\hat{\rho}_j$ for a node j
- Remember that the Bernoulli distribution models the probability of an event with two outcomes (e.g. coin toss)
- In our case p will represent a node being active with probability ρ, and q_i a node being active with probability $\hat{\rho}_i$

In this case, the KL divergence is

$$\begin{aligned} D_{KL}(p||q_i) &= \int p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x \\ &= \int p(x) \log p(x) \, \mathrm{d}x - \int p(x) \log q(x) \, \mathrm{d}x \end{aligned}$$

In our case, the support of the distributions is only two outcomes $x \in \{0, 1\}$, and the pmf is

$$p(x) = \begin{cases} (1-\rho), & x = 0 \\ \rho, & x = 1 \end{cases}$$

and conversely for $q(x)$. With this, our KL divergence is simply

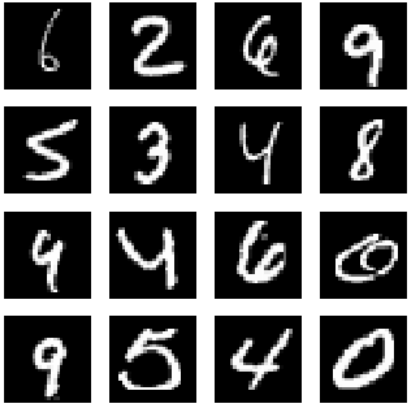
$$\begin{aligned} D_{KL}(p||q_i) &= \rho \log \rho + (1-\rho) \log(1-\rho) - [\rho \log \hat{\rho} + (1-\rho) \log(1-\hat{\rho}_i)] \\ &= \rho \log \frac{\rho}{\hat{\rho}_i} + (1-\rho) \log \frac{(1-\rho)}{(1-\hat{\rho}_i)}. \end{aligned}$$

- With this, we get our final loss

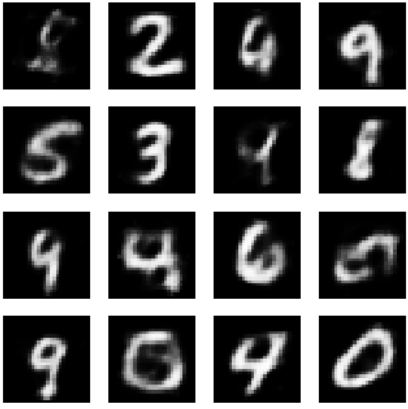
$$\mathcal{L} = \mathcal{L}_{\text{reconstruction}} + \beta \sum_{j=1}^{n^{[c]}} \rho \log \frac{\rho}{\hat{\rho}_i} + (1-\rho) \log \frac{(1-\rho)}{(1-\hat{\rho}_i)}$$

- Note that we need to average over all examples to compute $\hat{\rho}$
- This means that we have to encode all said examples
- In practice, with batch optimization, we average over all examples in a batch

Sparse autoencoder — MNIST example



(a) Original



(b) Reconstructed

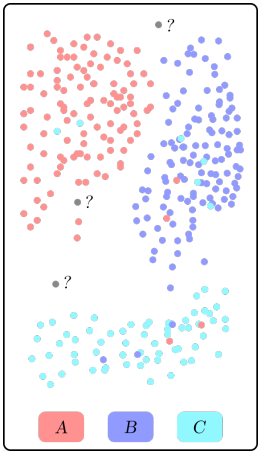
VARIATIONAL AUTOENCODERS

Introduction

- Popular method for signal generation (images, sound, language, etc.)
- Creating completely new signals
- Or altering existing data
- Especially powerful when you want to alter your data in a specific way, not just randomly

Problems with autoencoders for signal generation

- An autoencoder works great if you want to reconstruct a *replica* of the input
- Not well suited for generating new signal
- The reason for this is an “unintuitive” latent variable space
- The latent space might be discontinuous
- Random sampling from an “unseen” region of the latent space produces undesired results
- No reasonable way to interpolate between categories in the latent space

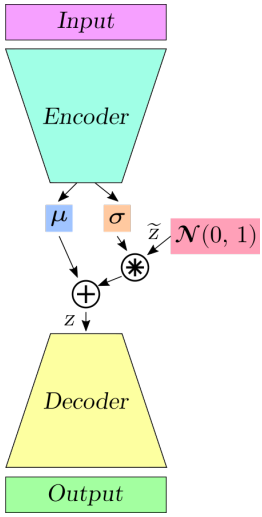


Variational autoencoders

- A variational autoencoder is designed to have a continuous latent space
- This makes them ideal for random sampling and interpolation
- It achieve this by forcing the encoder g to generate Gaussian representations, $z \sim \mathcal{N}(\mu, \sigma^2)$
- More precisely, for one input, the encoder generates a mean μ and a variance σ^2
- We sample then sample a zero-mean, unit-variance Gaussian $\tilde{z} \sim \mathcal{N}(0, 1)$
- Construct the input z to the decoder from this

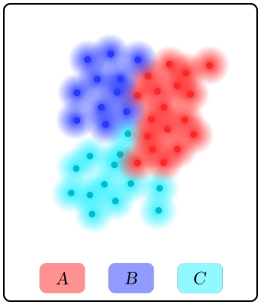
$$z = \mu + \tilde{z} \cdot \sigma^2$$

- With this, z is sampled from $q = \mathcal{N}(\mu, \sigma^2)$



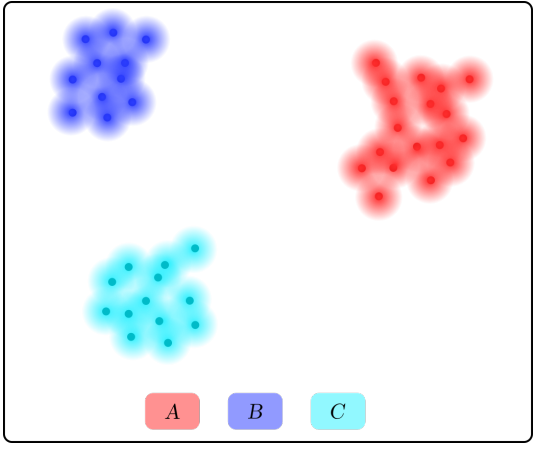
Intuition

- This is a stochastic sampling
- That is, we can sample different z from the same set of (μ, σ^2)
- The intuition is that the decoder "learns" that for a given input x:
  - the point z is important for reconstruction
  - but also a neighbourhood of z
- In this way, we have smoothed the latent space, at least locally



Problem

- No restriction on μ or σ^2
- Realisticly, clusters of different classes can be placed far apart



Guiding the generative distribution

- We can guide the solutions by restricting the generative distribution q
- We do this by making it approximate some distribution p
- In that way, the latent vectors, even for different categories, will be relatively close
- The desired distribution used in variational autoencoders is the standard normal $p = \mathcal{N}(0, 1)$
- We use the familiar KL-divergence between the desired and the generated distribution as a regularizer in the loss function
- With this, the total loss for an example x_i is something like

$$\mathcal{L}(x_i) = |x^{(i)} - f(x^{(i)})| + D_{KL}(p||q_{\mu_i,\sigma_i})$$

- That is, the sum what we call the *reconstruction loss* and the *latent loss*
- The latent loss for a single variable x_i can be shown to be equal to

$$D_{KL}(p||q_{\mu_i,\sigma_i}) = \frac{1}{2}(\mu_i^2 + \sigma_i^2 - \log \sigma_i^2 - 1)$$

KL Divergence between Gaussian distributions

For reference, I will spend some slide deriving the KL Divergence between two Gaussian distributions $p = \mathcal{N}(\mu_p, \sigma_p^2)$ and $q = \mathcal{N}(\mu_q, \sigma_q^2)$.
We are going to derive it for the continuous case, where the KL-Divergence can be expressed as

$$D_{KL}(p||q) = \int p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x \tag{1}$$

$$= \int p(x) \log p(x) \, \mathrm{d}x - \int p(x) \log q(x) \, \mathrm{d}x \tag{2}$$

We will derive the two terms in the last line seperately

First, for the first term

$$\begin{aligned}\int p(x)\log p(x) &= \int p(x)\log\left[(2\pi\sigma_p^2)^{-\frac{1}{2}}\exp\left\{-\frac{(x-\mu_p)^2}{2\sigma_p^2}\right\}\right]\mathrm{d}x\\ &= -\frac{1}{2}\log(2\pi\sigma_p^2)\int p(x)\,\mathrm{d}x - \frac{1}{2}\int p(x)\frac{(x-\mu_p)^2}{\sigma_p^2}\,\mathrm{d}x\\ &= -\frac{1}{2}\log(2\pi\sigma_p^2) - \frac{1}{2\sigma_p^2}\int p(x)(x^2-2x\mu_p+\mu_p^2)\,\mathrm{d}x.\end{aligned}$$

Similarly, for the second term

$$\int p(x)\log q(x) = -\frac{1}{2}\log(2\pi\sigma_q^2) - \frac{1}{2\sigma_q^2}\int p(x)(x^2-2x\mu_q+\mu_q^2)\,\mathrm{d}x.$$

Remeber that for a random variable X with pdf f, the expectation is given by

$$E[X] = \int f(x)x \, \mathrm{d}x.$$

Also, we have

$$\begin{aligned} E[X^2] &= \int f(x)x^2 \, \mathrm{d}x \\ &= Var[X] + E[X]^2 \end{aligned}$$

For the integral in eq. (62), we then get

$$\begin{aligned} \frac{1}{2\sigma_p^2} \int p(x)(x^2 - 2x\mu_p + \mu_p^2) \, \mathrm{d}x &= \frac{1}{2\sigma_p^2} \int p(x)[(\sigma_p^2 + \mu_p^2) - 2\mu_p^2 + \mu_p^2] \, \mathrm{d}x \\ &= \frac{1}{2}. \end{aligned}$$

The integral in eq. (62) is similar,

$$\begin{aligned}\frac{1}{2\sigma_q^2}\int p(x)(x^2 - 2x\mu_q + \mu_q^2)\,\mathrm{d}x &= \frac{1}{2\sigma_q^2}\int p(x)[(\sigma_p^2 + \mu_p^2) - \mu_p\mu_q + \mu_q^2]\,\mathrm{d}x \\ &= \frac{\sigma_p^2 + (\mu_p - \mu_q)^2}{2\sigma_q^2}.\end{aligned}$$

Finishing up, using eq. (62) and eq. (62) via eq. (63) and eq. (64), we finally get

$$\begin{aligned} D_{KL}(p||q) &= \int p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x \\ &= \int p(x) \log p(x) \, \mathrm{d}x - \int p(x) \log q(x) \, \mathrm{d}x \\ &= -\frac{1}{2} \log(2\pi\sigma_p^2) - \frac{1}{2} + \frac{1}{2} \log(2\pi\sigma_q^2) + \frac{\sigma_p^2 + (\mu_p - \mu_q)^2}{2\sigma_q^2} \\ &= \frac{1}{2} \left[\log \frac{\sigma_q^2}{\sigma_p^2} + \frac{\sigma_p^2 + (\mu_p - \mu_q)^2}{\sigma_q^2} - 1 \right] \end{aligned}$$

When, as in our case $p = \mathcal{N}(\mu, \sigma)$ and $q = \mathcal{N}(0, 1)$, we get

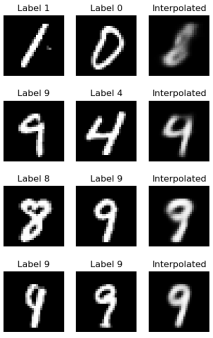
$$D_{KL}(p||q) = \frac{1}{2} \left[\mu^2 + \sigma^2 - \log \sigma^2 - 1 \right].$$

Generate new signals — interpolation

- Say you want to generate a signal c that is an interpolation between two signals a and b
- You can do this by first computing the average of the two mean vectors

$$\mu_c = \frac{1}{2}(\mu_a + \mu_b)$$

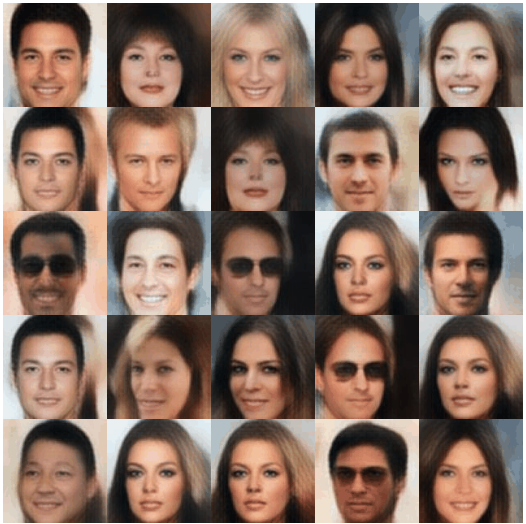
- Then, sample a latent variable with this mean



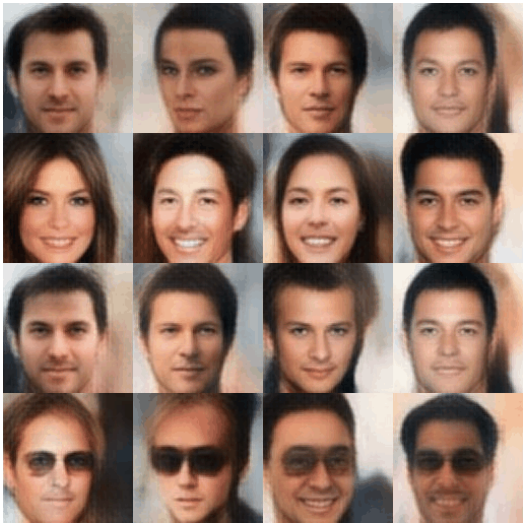
Generate new signals — adding features

- Say you want to add a feature of a signal a to the signal b
- You can do this by finding a signal c that is equal to a, except for the specific feature you want
- You can then subtract the latent variable of c from the latent variable of a, and add it to the latent variable of b
- Then you simply decode the new latent variable
- Example: “Face with glasses = face + (face with glasses — face)”
- See examples on the next slides

Variational autoencoders — Interpolation and combination example



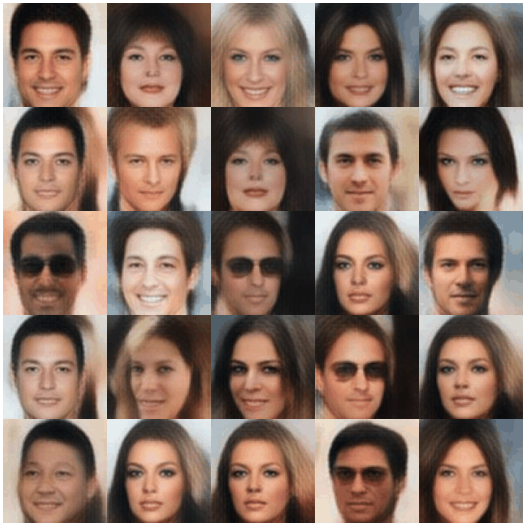
(a) Interpolation between genders



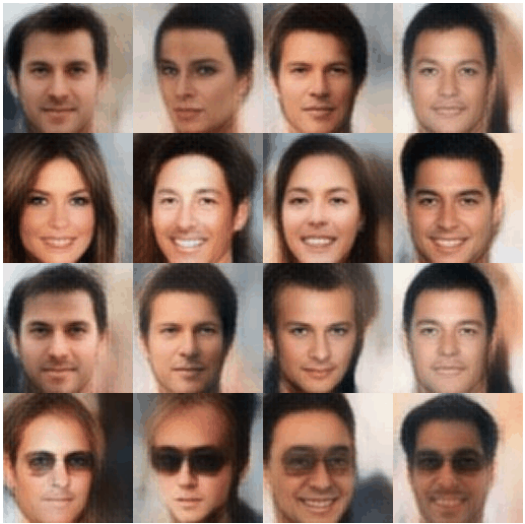
(b) Add or remove facial features

Figure 11: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



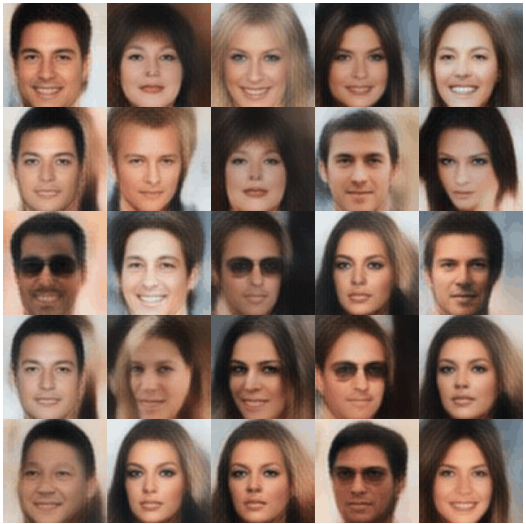
(a) Interpolation between genders



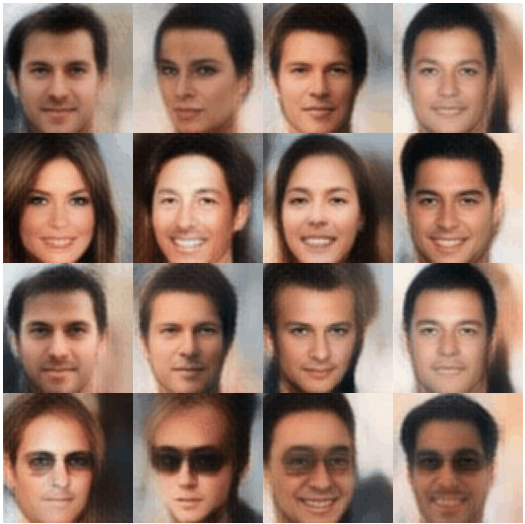
(b) Add or remove facial features

Figure 12: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



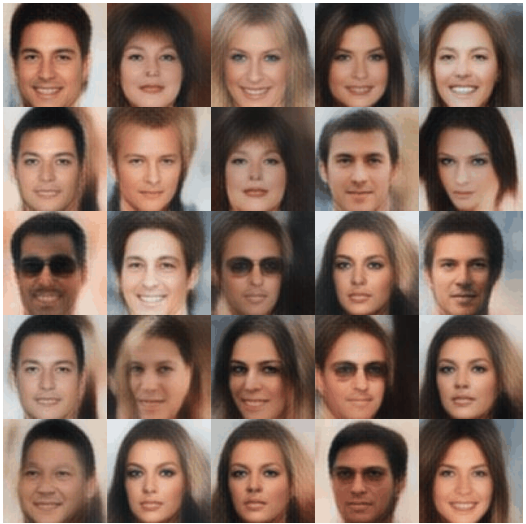
(a) Interpolation between genders



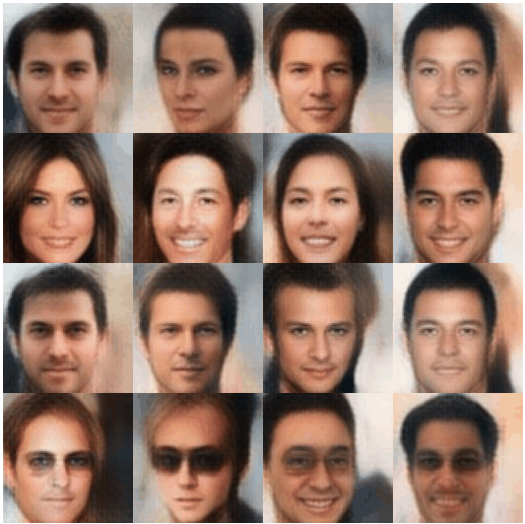
(b) Add or remove facial features

Figure 13: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



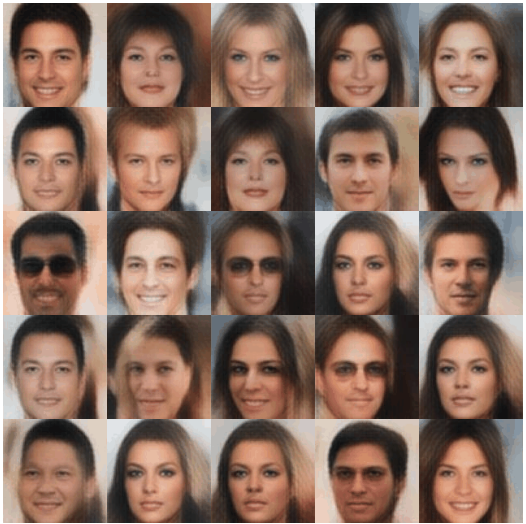
(a) Interpolation between genders



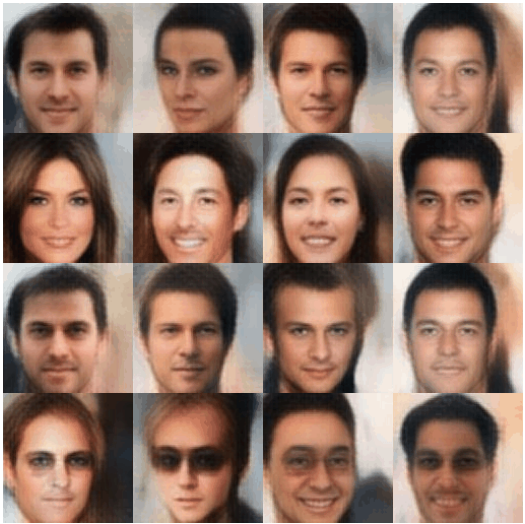
(b) Add or remove facial features

Figure 14: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



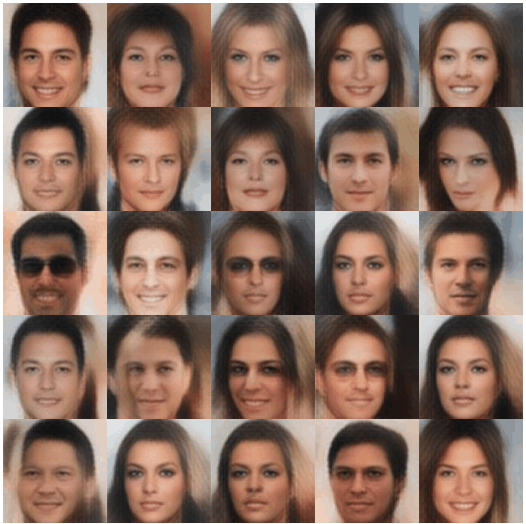
(a) Interpolation between genders



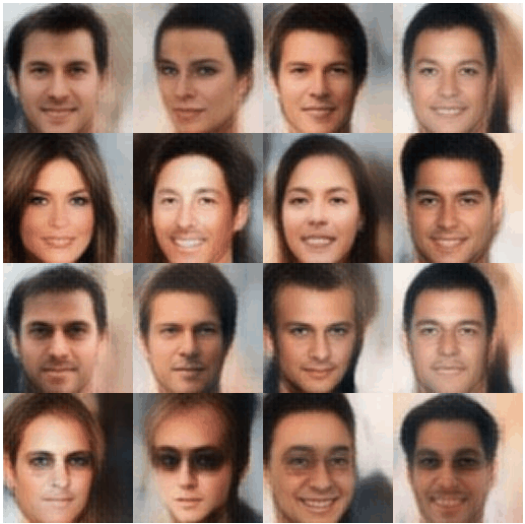
(b) Add or remove facial features

Figure 15: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



(a) Interpolation between genders

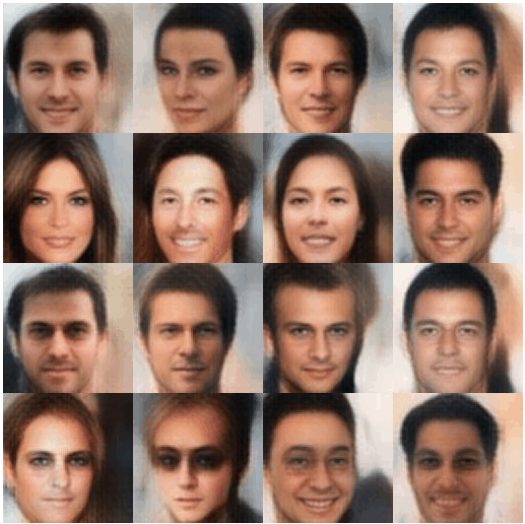


(b) Add or remove facial features

Figure 16: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example

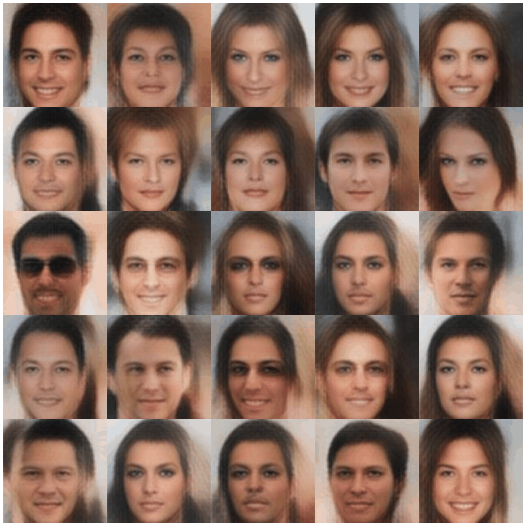
(a) Interpolation between genders



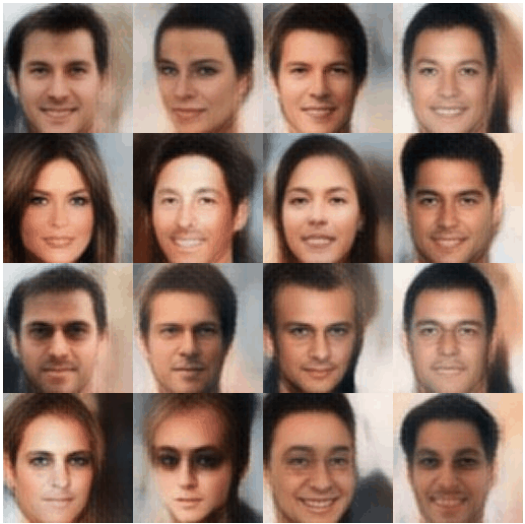
(b) Add or remove facial features

Figure 17: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



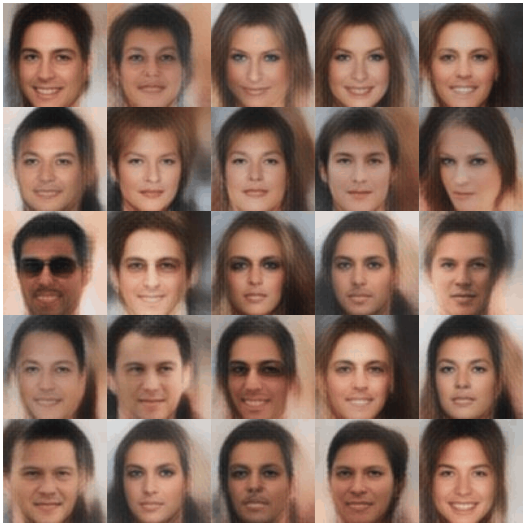
(a) Interpolation between genders



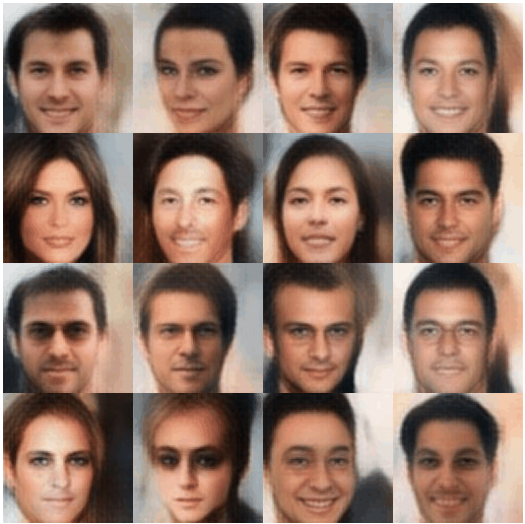
(b) Add or remove facial features

Figure 18: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



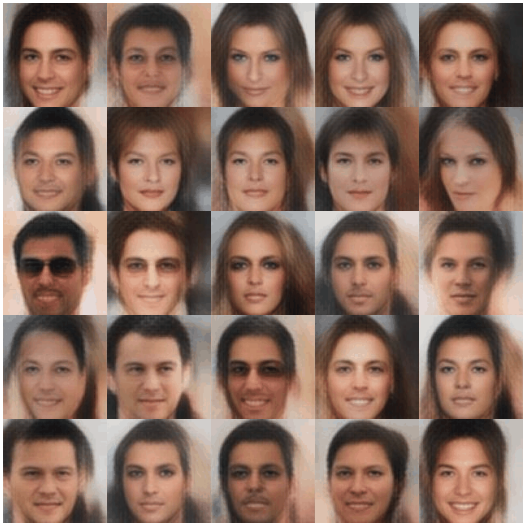
(a) Interpolation between genders



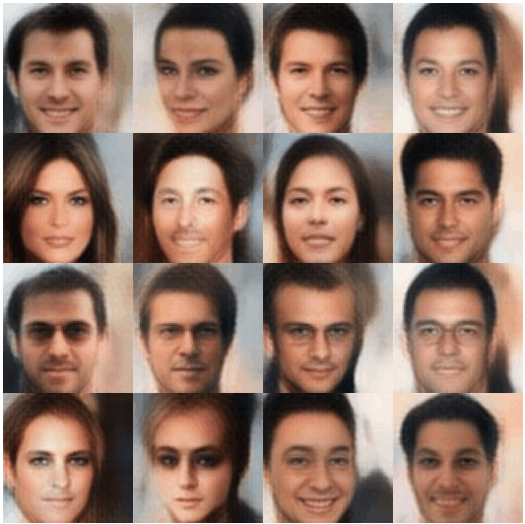
(b) Add or remove facial features

Figure 19: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



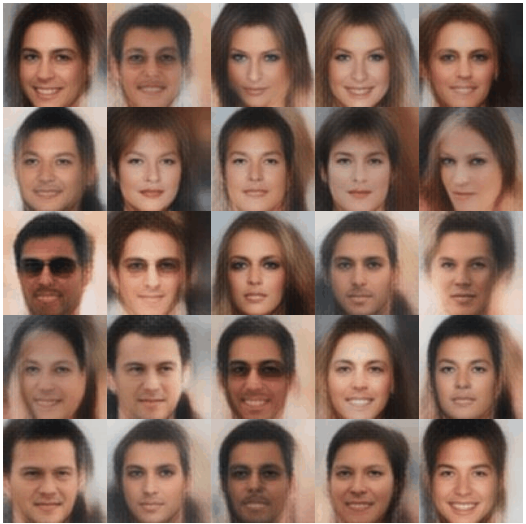
(a) Interpolation between genders



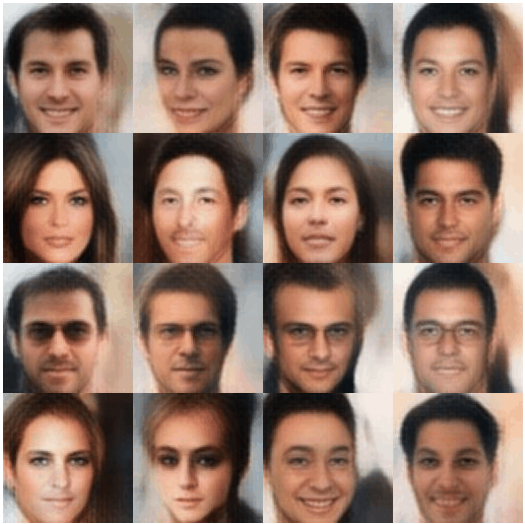
(b) Add or remove facial features

Figure 20: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



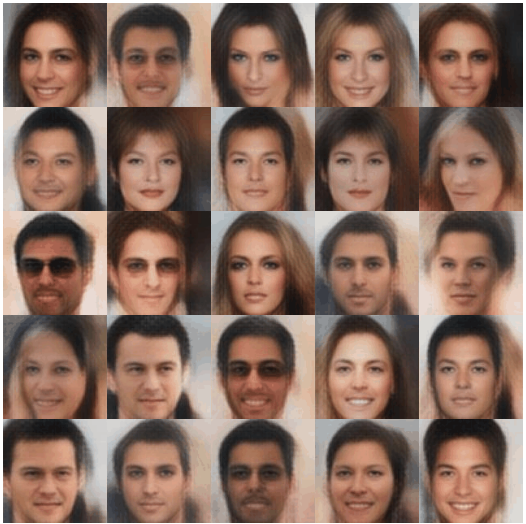
(a) Interpolation between genders



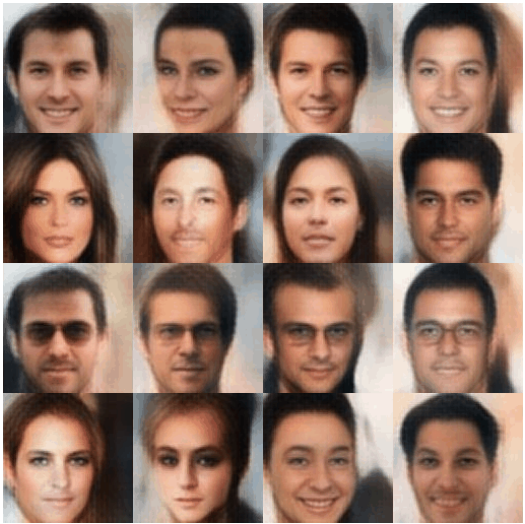
(b) Add or remove facial features

Figure 21: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



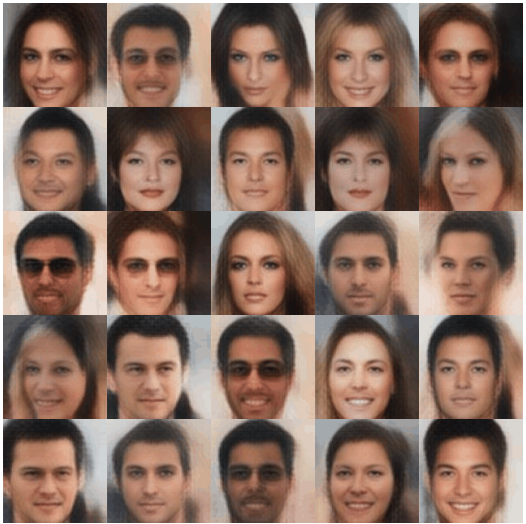
(a) Interpolation between genders



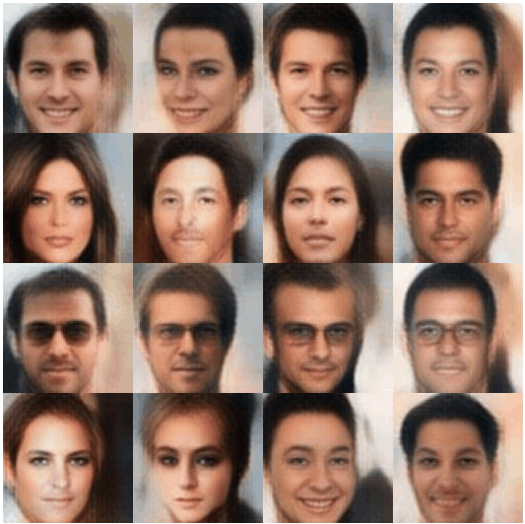
(b) Add or remove facial features

Figure 22: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



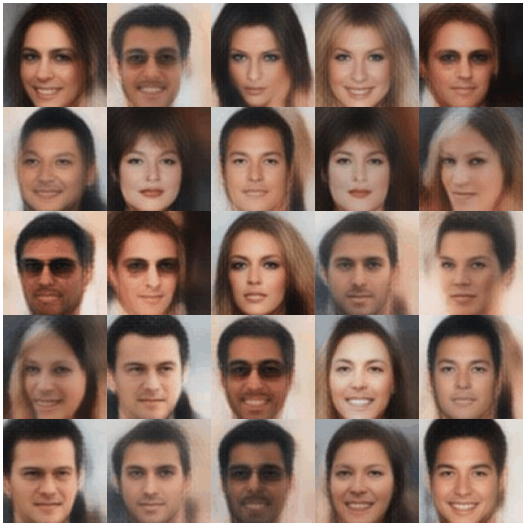
(a) Interpolation between genders



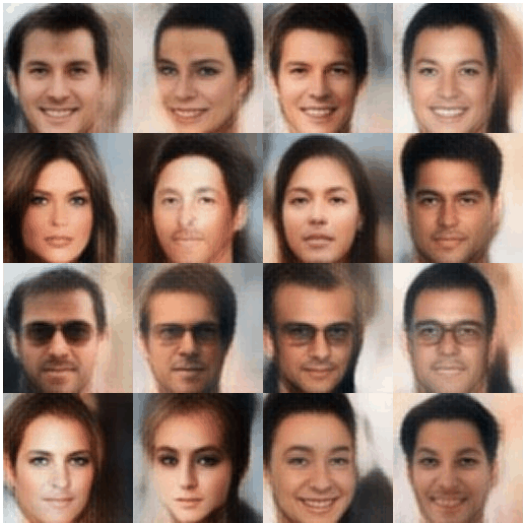
(b) Add or remove facial features

Figure 23: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



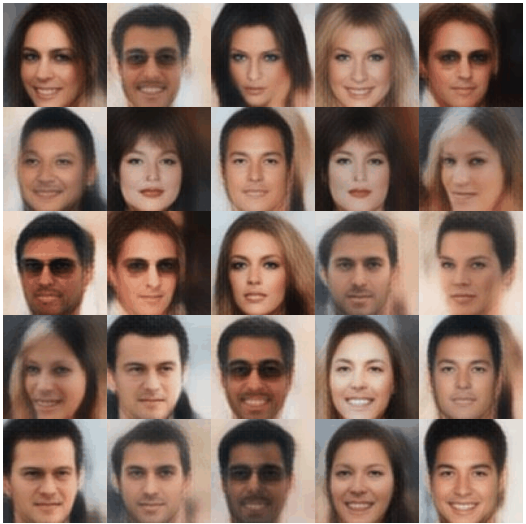
(a) Interpolation between genders



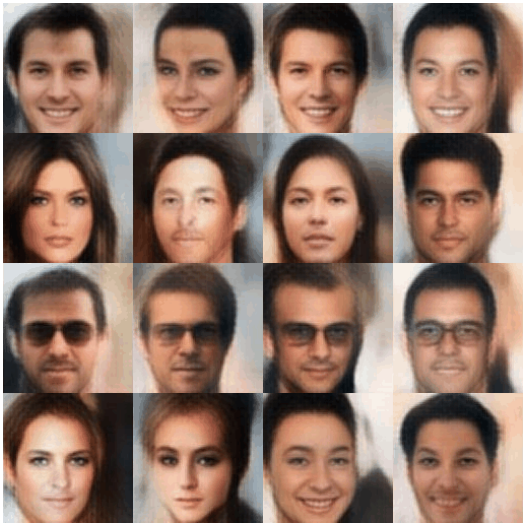
(b) Add or remove facial features

Figure 24: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



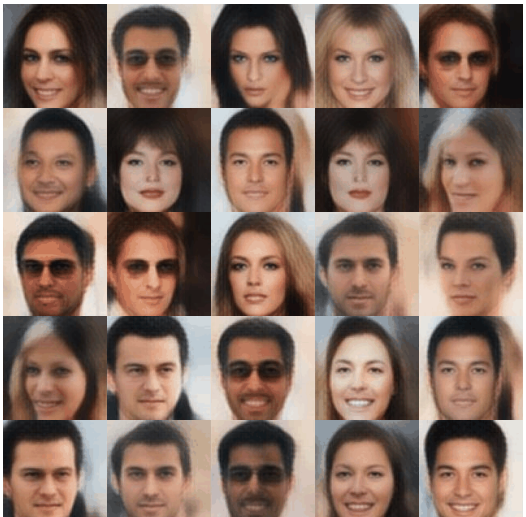
(a) Interpolation between genders



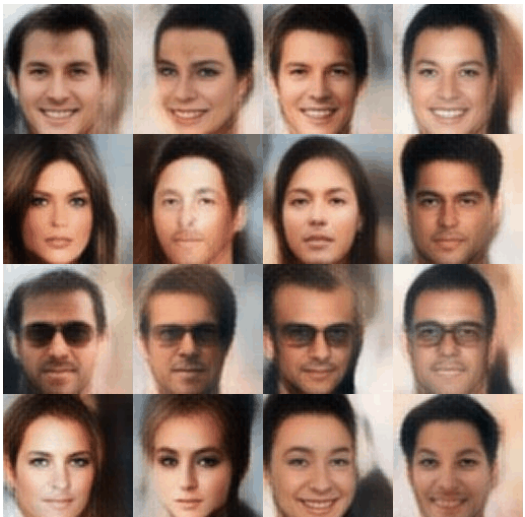
(b) Add or remove facial features

Figure 25: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



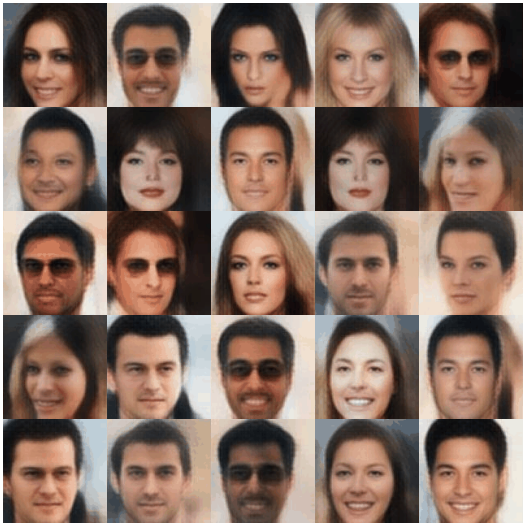
(a) Interpolation between genders



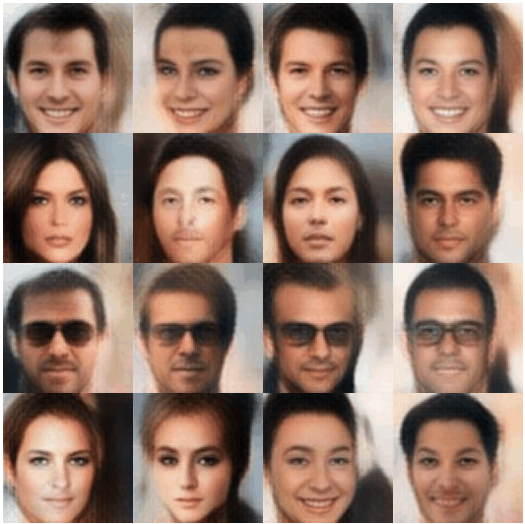
(b) Add or remove facial features

Figure 26: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



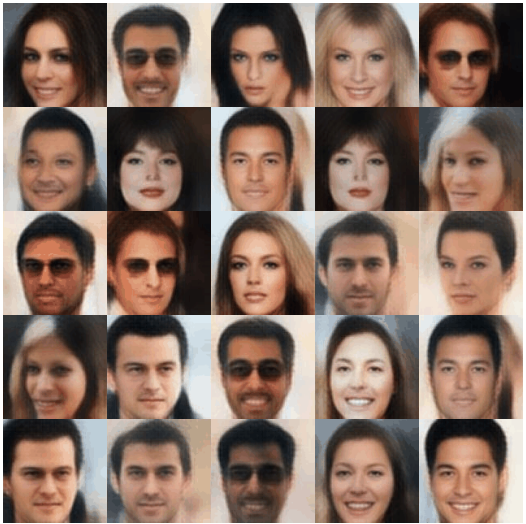
(a) Interpolation between genders



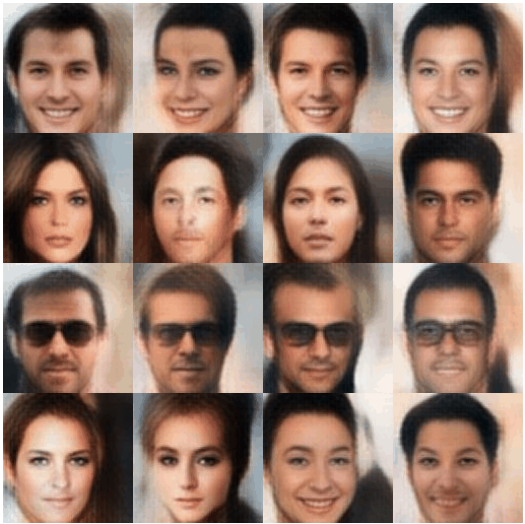
(b) Add or remove facial features

Figure 27: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



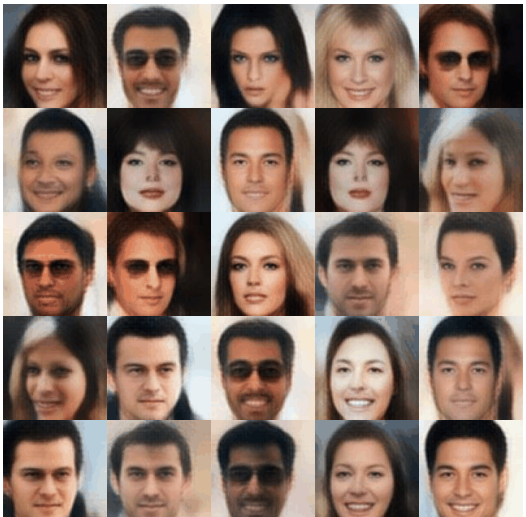
(a) Interpolation between genders



(b) Add or remove facial features

Figure 28: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — Interpolation and combination example



(a) Interpolation between genders

(b) Add or remove facial features

Figure 29: Source: Deep Feature Consistent Variational Autoencoder, https://houxianxu.github.io/assets/project/dfcvae

Variational autoencoders — final notes

- Autoencoders are popular for generating signals such as images and sound
- We have now an idea of how variational autoencoders work
- There is a lot more detail that we skipped, especially in the derivation

Questions?