Unsupervised Learning

Matias Valdenegro-Toro

Hochschule Bonn-Rhein-Sieg &
German Research Centre for Artificial Intelligence (Bremen, Germany)

June 4, 2018

Introduction

- Supervised Learning requires training labels, which can be expensive to obtain.
- Learning without labels is possible but considerably harder, as now the algorithm has to find hidden "patterns" in the data. This is Unsupervised Learning.
- A clear example is that children can recognize basic visual concepts like edges, corners, and parts, in order to use them to recognize objects.
- A quote from Geoffrey Hinton: "The brain has about 10^{14} synapses and we only live for about 10^9 seconds. So we have a lot more parameters than data. This motivates the idea that we must do a lot of unsupervised learning since the perceptual input (including proprioception) is the only place we can get 10^5 dimensions of constraint per second. "

Unsupervised Learning Techniques

Clustering
: Group data points into clusters based into similarity. K-Means fits here and it was already covered.

Feature Learning
: Learn features that are specific for a given task directly from data.

Hidden/Latent Models
: Discover a set of hidden variables from observations.

Generative Models
: Learn the probability distribution given a limited sample of it.

Anomaly Detection
: Detect deviations from normal behavior in streaming data.

Feature Learning

Unsupervised Feature Learning

- Automatically discover useful representations for classification/regression from raw data.
- Also called representation or embedding learning. It replaces manual feature engineering (pre-Deep Learning era).
- For labeled data it is very easy, just train a neural network and use intermediate layers as features. CNN
- But doing this without labels is harder, only non-obvious structure in the raw data can be exploited.
- Not so many techniques can do it in a fully unsupervised manner.

Dictionary Learning

- Given N unlabeled data points $x_i \in \mathcal{R}^d$, we wish to learn a linear representation of these points based on a set of basis vectors $\mathbf{d}_i$ of dimension d:

$$x = \sum_{i}^{k} r_i \cdot \mathbf{d}_i = \mathbf{D}\,\mathbf{r}$$

- Where $\mathbf{r}$ are the representation weights corresponding to each basis vector $\mathbf{d}$. It can be represented as a matrix product, where the representation depends on the input data point x.
- The dictionary $\mathbf{D} \in \mathcal{R}^{d \times k}$ contains the basis vectors, while the representation $\mathbf{r} \in \mathcal{R}^k$ contains the weights. Each weight is a scalar.
- A constraint is typically introduced in order to limit the values of the rows in $\mathbf{D}$, namely $||d_i|| \leq 1$.

Dictionary Learning

- The dictionary and representations can be easily learned by minimizing the following loss:

$$L(X) = n^{-1} \sum_{i}^{n} ||x_i - \mathbf{D}\mathbf{r}_i||^2 + \lambda \sum_{i} |\mathbf{r}_i|$$

- The term $\lambda \sum_i |\mathbf{r}_i|$ corresponds to L^1 regularization over the weights r in order to obtain a **sparse representation**.

Only some of them activate.

- Once learned, the sparse representations can be used as discriminative features. But computing these representations is expensive.

Learning the Dictionary and Sparse Representation

- There are many techniques to learn both the dictionary $\mathbf{D}$ and the representation $\mathbf{r}$.
- A simplified version is to start with a random dictionary and representation, and the iteratively update $\mathbf{D}$ while keeping $\mathbf{r}$ fixed, then find the best $\mathbf{r}$ while keeping $\mathbf{D}$ fixed, and repeat the process until the loss converges.
- The constrains $||d_i|| \leq 1$ have to be considered while updating $\mathbf{D}$. This is usually done by moving $\mathbf{D}$ in a direction that minimizes the loss, then projecting the solution back so it fulfills the constraints.
- This can be done thorough gradient descent or quadratic optimization.

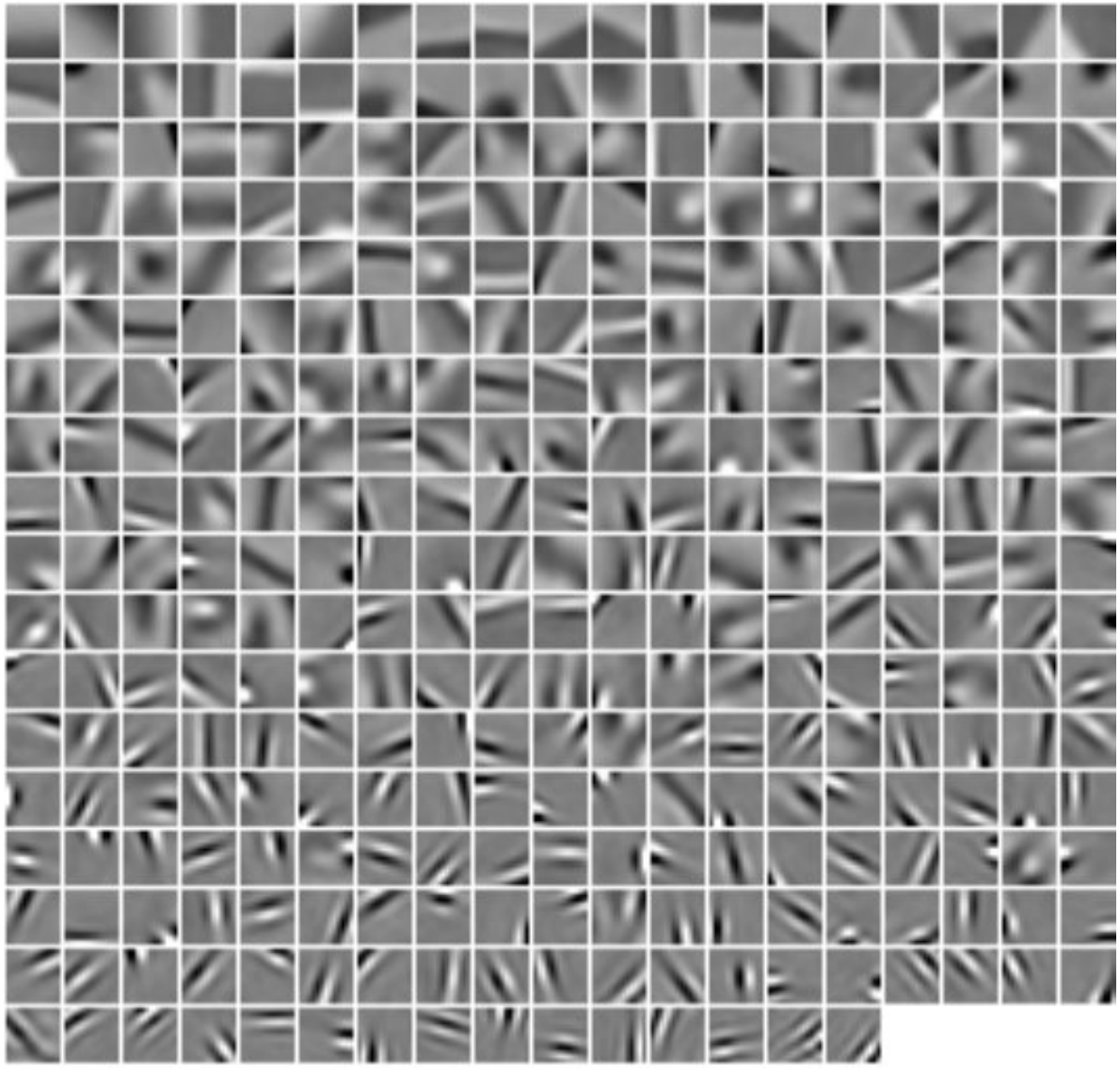
Predicting Representations for Unseen Data

- Given a new data point $\mathbf{x}$ and a trained dictionary $\mathbf{D}$, the representation $\mathbf{r}$ can be obtained as:

$$\mathbf{r} = \text{argmin}_{\mathbf{r}} \; ||\mathbf{x} - \mathbf{D}\mathbf{r}_i||^2 + \lambda \sum_i |\mathbf{r}_i|$$

- This means that once the dictionary is fixed, learning a new representation entails a new optimization problem.

Learned Dictionary Example



Semi-Supervised Feature Learning

- Sometimes it is easier to obtain weak labels instead of full class labels. For example, instance A is similar to B, but not to C.
- Two methods can exploit this information in order to learn an embedding that respects the weak labels.
- Contrastive Loss: Minimize distance between similar inputs, while maximizing distances between dissimilar inputs.
- Triplet Loss: Given an anchor, pull similar inputs closer, while push away dissimilar inputs.

Contrastive Loss

- A Feature space/Embedding is learned through neighborhood relations. The dimensionality of the Embedding (d) is lower than the input space (D).
- This is encoded through a learnable function $G_\theta(x) : \mathcal{R}^D \to \mathcal{R}^d$ that encodes a input to embedding transformation. Typically this is a neural network.
- The only labels required are binary y_i indicating if two pairs of inputs (x_1, x_2) are "similar" $(y_i = 1)$ or not $(y_i = 0)$.
- Then the contrastive loss function is:

$$D_\theta(x_1, x_2) = ||G_\theta(x_1) - G_\theta(x_2)||_2$$

$$L(\theta, y, x_1, x_2) = \frac{(1-y)(D_\theta(x_1, x_2))^2}{2} + \frac{y\{\max(0, m - D_\theta(x_1, x_2))\}^2}{2}$$

Contrastive Loss

- Where $m > 0$ is a enforced margin between similar and dissimilar inputs. The supervision through this loss only encodes relative distances between training pairs.
- For training, labeled pairs (y, x_1, x_2) are generated, which can be done automatically if the neighborhood criteria is known.
- The effect of this loss is to pull similar points closer in embedding distance (gradient descent), while pushing away dissimilar points (gradient ascent). The max-margin loss makes sure that only dissimilar pairs with minimum distance m contribute to the loss.
- An analogy to this learning process is a spring-mass system, where some nodes repel each other, and others attract.

Contrastive Loss

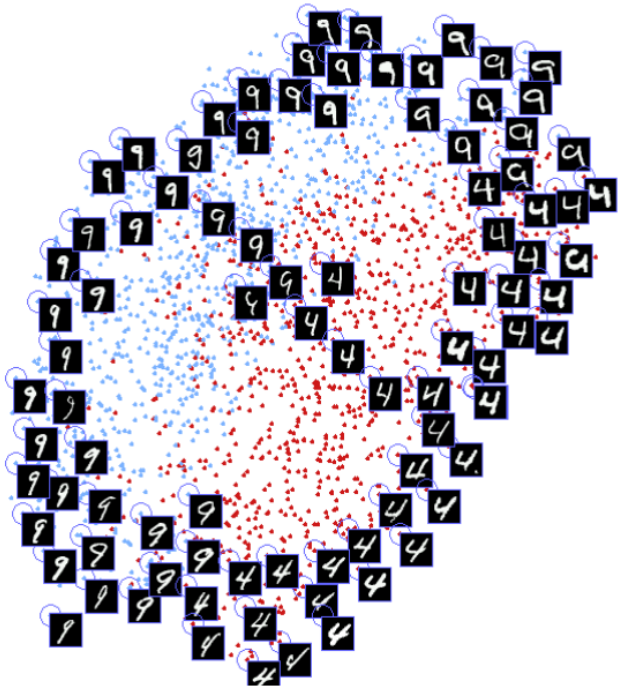


Figure 4. Experiment demonstrating the effectiveness of the DrLIM in a trivial situation with MNIST digits. A Euclidean nearest neighbor metric is used to create the local neighborhood relationships among the training samples, and a mapping function is learned with a convolutional network. Figure shows the placement of the *test* samples in output space. Even though the neighborhood relationships among these samples are unknown, they are well organized and evenly distributed on the 2D manifold.

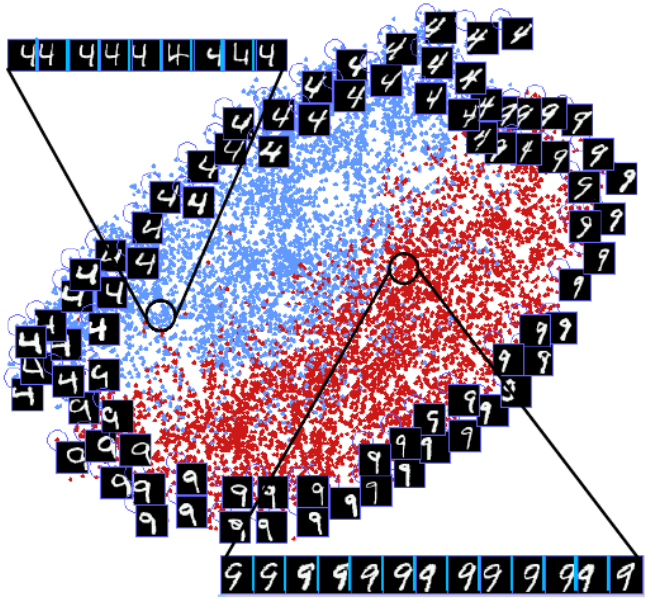


Figure 7. This experiment measured DrLIM's success at learning a mapping from high-dimensional, shifted digit images to a 2D manifold. The mapping is invariant to translations of the input images. The mapping is well-organized and globally coherent. Results shown are the test samples, whose neighborhood relations are unknown. Similar characters are mapped to nearby areas, regardless of their shift.

Figure extracted from " Dimensionality Reduction by Learning an Invariant Mapping" by Hadsell et al. 2006

Triplet Loss

This was usefull for embedding faces.

- This method is quite similar to the Contrastive Loss, but it is trained on triplets (x^a, x^p, x^n). The anchor and the positive sample are neighbors, while the anchor and the negative sample are not neighbors.
- During learning, embedding distances between the anchor and the positive sample are decreased, while distances between anchor and the negative are increased.

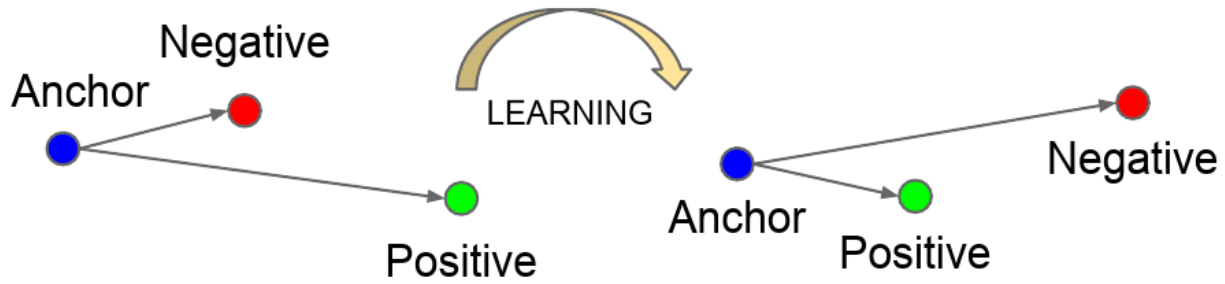


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

Triplet Loss

- The triplet loss was initially used for embedding faces (in a network called FaceNet).
- For each triplet (x_a, x_p, x_n) the following must hold:

$$||f(x^a) - f(x^p)||^2 + \alpha < ||f(x^a) - f(x^n)||^2$$

- Where α is a margin enforced between positives and negatives. The loss to minimize is then:

$$L(\theta) = \sum_i^n \left(||f(x_i^a) - f(x_i^p)||^2 - ||f(x_i^a) - f(x_i^n)||^2 + \alpha \right)$$

- Where as in the contrastive loss, the function f encodes the embedding with parameters θ.
- Different strategies exist in order to generate triplets that improve learning, as many easy triples slow down learning.

Triplet Loss

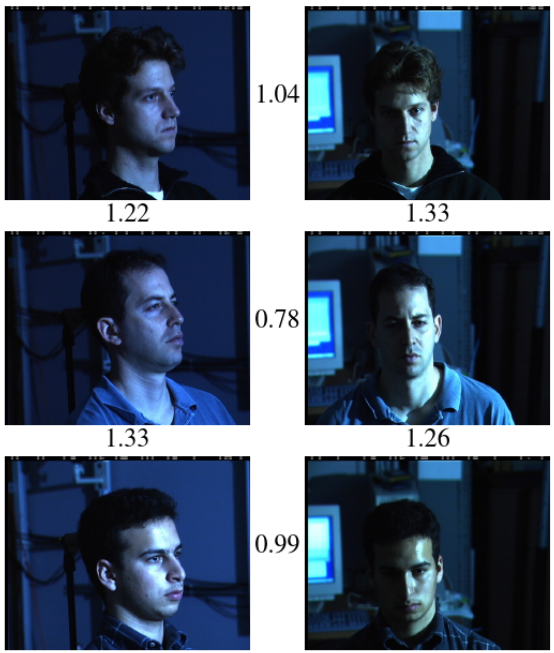


Figure 1. **Illumination and Pose invariance.** Pose and illumination have been a long standing problem in face recognition. This figure shows the output distances of FaceNet between pairs of faces of the same and a different person in different pose and illumination combinations. A distance of 0.0 means the faces are identical, 4.0 corresponds to the opposite spectrum, two different identities. You can see that a threshold of 1.1 would classify every pair correctly.

'igure 7. **Face Clustering.** Shown is an exemplar cluster for one ser. All these images in the users personal photo collection were lustered together.

Figure extracted from "FaceNet: A Unified Embedding for Face Recognition and Clustering" by Schroff et al. 2015

Autoencoders

Autoencoders

- An Autoencoder is a model that is used to learn an unsupervised representation of the input data. As a basic concept, an Autoencoder performs regression by predicting a reconstruction of the input data.
- It contains two internal modules, the encoder taking the input into a feature space (also called code), and a decoder transforming the feature space into the reconstructed input (the autoencoder's output).

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$
$$\psi : \mathcal{F} \rightarrow \mathcal{D}$$
$$\phi_\theta, \psi_\theta = \text{argmin}_{\phi_\theta, \psi_\theta} ||X - \psi(\phi(X))||^2$$

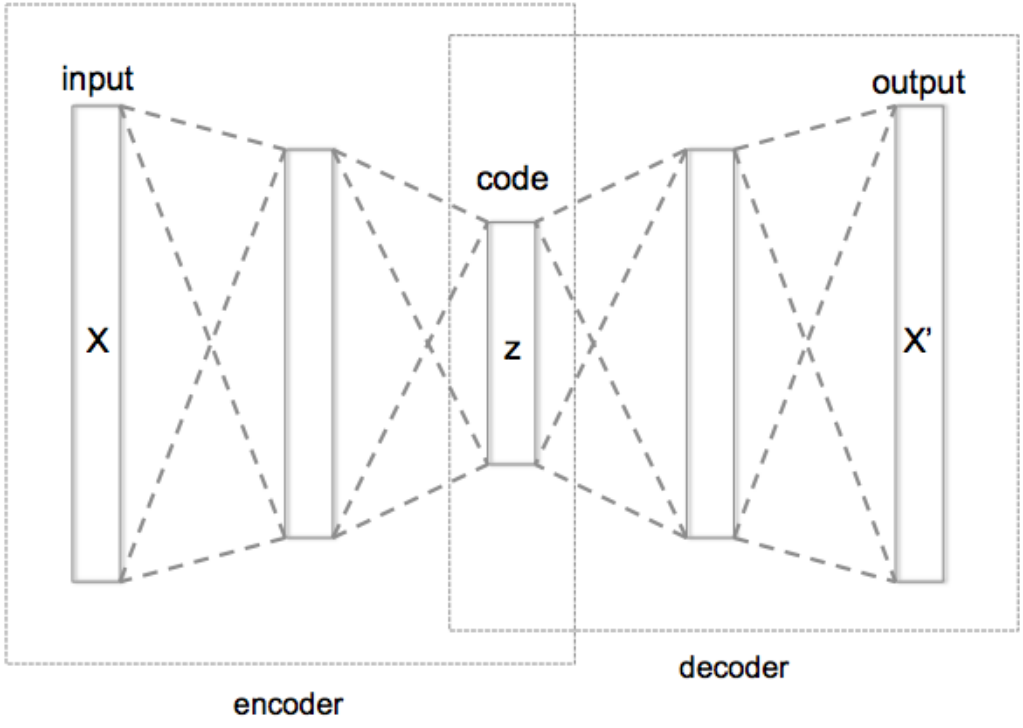
The encoder ϕ and decode ψ have trainable parameters that are learned using gradient descent.

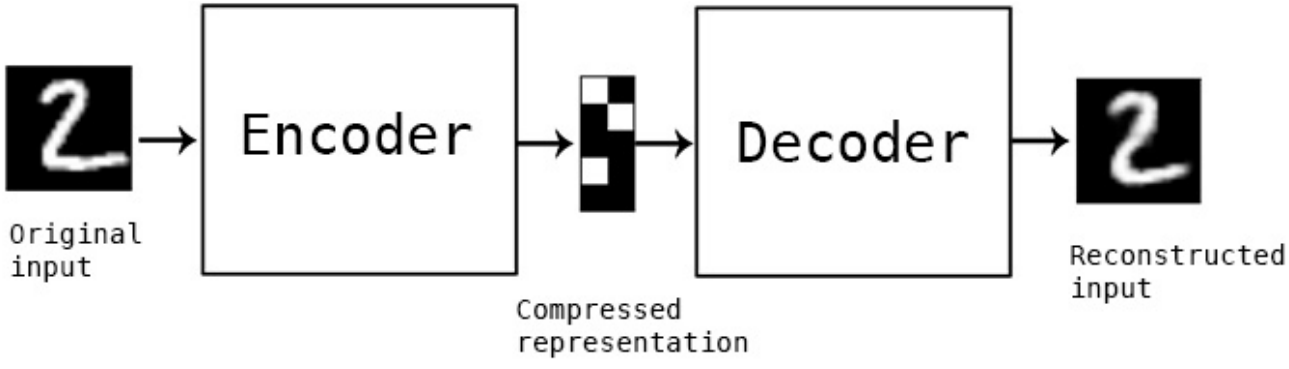
Autoencoders

- The key difficulty in training an Autoencoder is when the dimensionality of the feature space ($\mathcal{F}$) is smaller than the dimensionality of the input ($\mathcal{X}$).
- This implies that the Autoencoder cannot learn the identify function, and has to exploit patterns in the input data. A compressed representation of the input (the code) usually makes very good features for classification or regression problems.
- The encoder and decoder are usually implemented using a neural network. The loss function during learning is just the mean squared error, called the reconstruction error. No labels are needed, this method is fully unsupervised.

$$L(x) = \sum_i ||x_i - \psi(\phi(x_i))||^2$$

Autoencoder Structure





Reconstruction Example

Top row are original MNIST digits, bottom row are AE reconstructions. This model was trained using binary cross-entropy, and a 784-32-784 architecture with Fully Connected layers.

Denoising Autoencoders

- Usually it is desirable to have features that are robust to noise and variations of the inputs.
- One way to produce robust features is with a modified Autoencoder, where the inputs are corrupted with noise.
- This forces the Autoencoder to also learn how to remove the noise and produce a noise-free reconstruction of the input, effectively denoising it.
- This has the additional advantage of producing stronger features, as the added noise works as a regularizer.

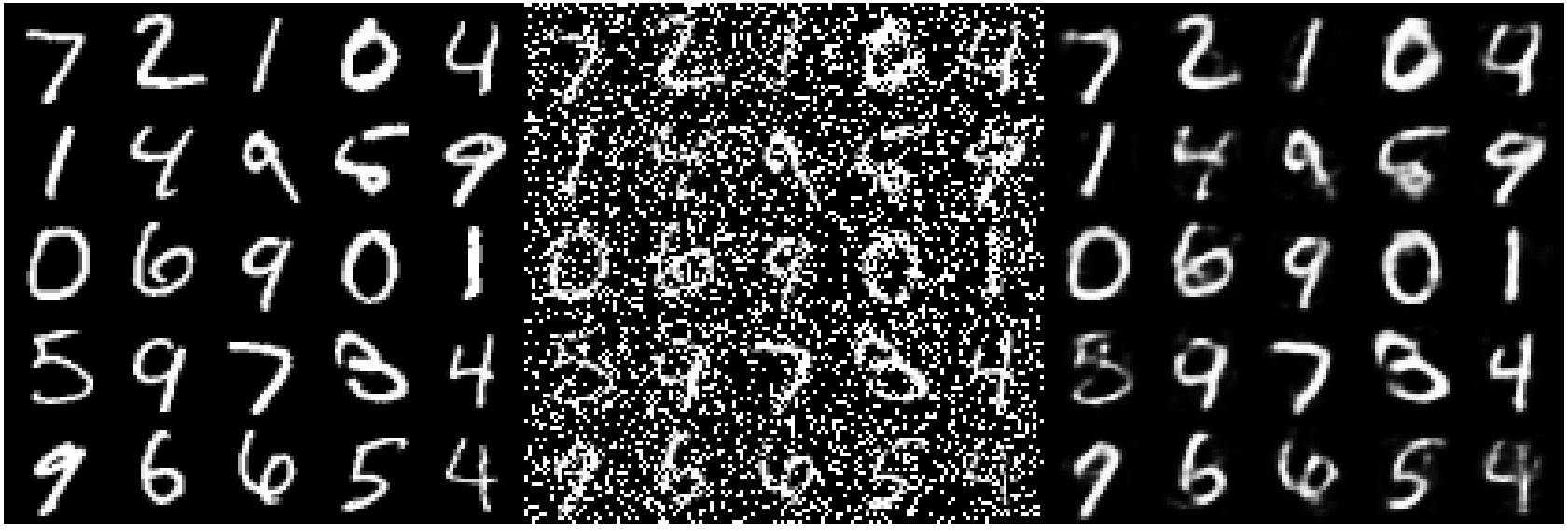
Denoising Autoencoders

- Given a input x, first it is corrupted with noise (to be chosen by a designer) to obtain noisy inputs $\tilde{x}$.
- The noisy inputs are given to the Autoencoder, but the outputs are compared to the clean inputs x. Using the following loss:

$$L(x) = n^{-1} \sum_i ||x_i - \psi(\phi(\tilde{x}_i))||^2$$

- This way the Autoencoder is forced to learn features from the data **and** to produce clean inputs from noisy ones.
- The noise is application and data specific, typical choices are salt-and-pepper noise for images, and Gaussian distributions.

Denoising Autoencoders



Variational Autoencoders

- One big issue with Autoencoders is that there is no control over the learned features.
- For example, one can use the decoder ψ as a generative model, but in order to do this, one has to know how to generate valid code/feature vectors.
- One easy way to add some degree of control, just add a term to the loss function that constraints it. A popular choice is to use the KL-Divergence between the code $\mathcal{F}$ and a given probability distribution $\mathcal{D}$.

$$L(x) = n^{-1} \sum_i ||x_i - \psi(\phi(x_i))||^2 + KL(f_i, d)$$

KL-Divergence is a distance metric of a probabilistic distribution.

Variational Autoencoders

- A popular choice for $\mathcal{D}$ is to use a unit Gaussian distribution (with zero mean and unit variance).
- This allows a reparametrization trick in order to simplify training and implementation. The encoder then outputs a mean μ and a standard deviation σ vectors, which are used to sample from a multivariate Gaussian distribution $\mathcal{N}(\mu, \sigma)$, and the sampled vector is used as input to the decoder.
- In practice is better for the encoder to predict μ and $\log \sigma^2$. Then the code/feature/latent vector is built as:

$$f = \mu + \epsilon e^{2 \log \sigma} \qquad \text{with } \epsilon \sim \mathcal{N}(0, 1)$$

Variational Autoencoder

- The loss can be simplified to:

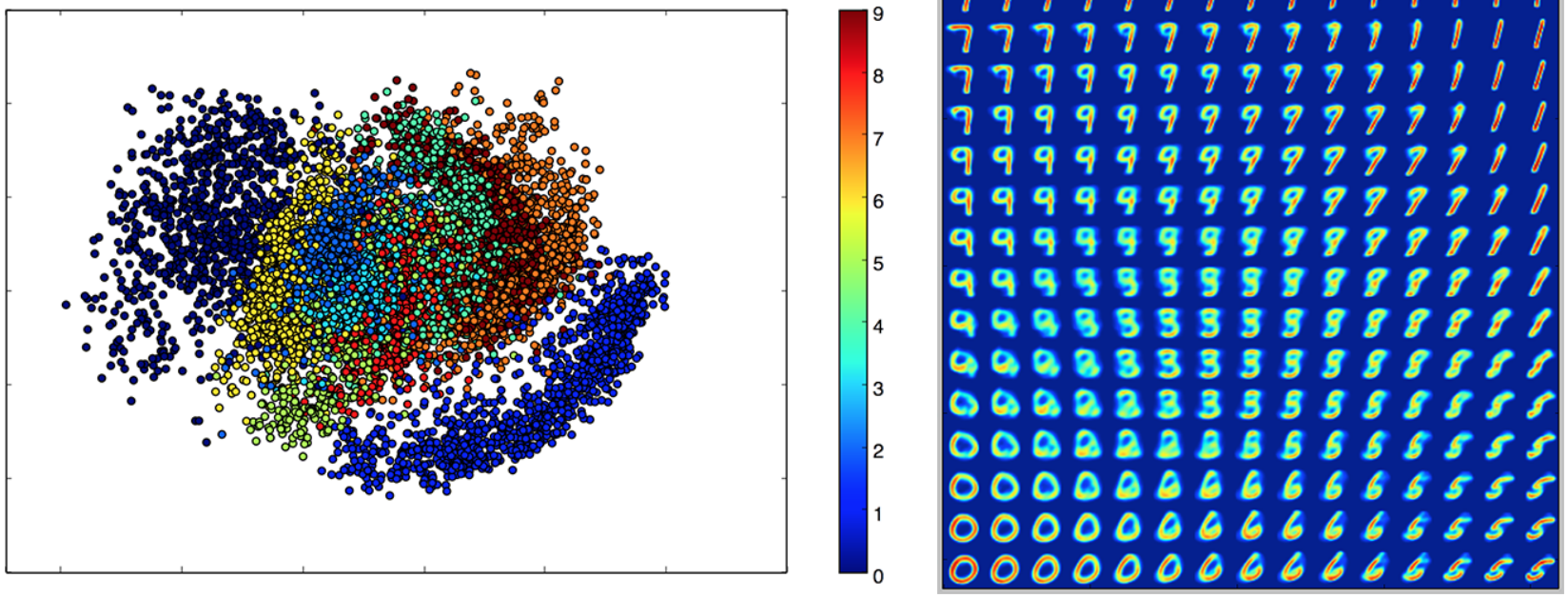
$$L(x) = n^{-1} \sum_i ||x_i - \psi(\phi(x_i))||^2 +$$

$$\frac{1}{2n} \sum_i (e^{\log \sigma(x_i)} + \mu(x_i)^2 - 1 - \log \sigma(x_i))$$

- Where the encoder ϕ predicts both the mean μ and logarithm of the standard deviation $\log \sigma$:

$$\phi(x_i) = (\, \mu(x_i), \log \sigma(x_i)\,)$$

Variational Autoencoder on MNIST



Dimensionality Reduction

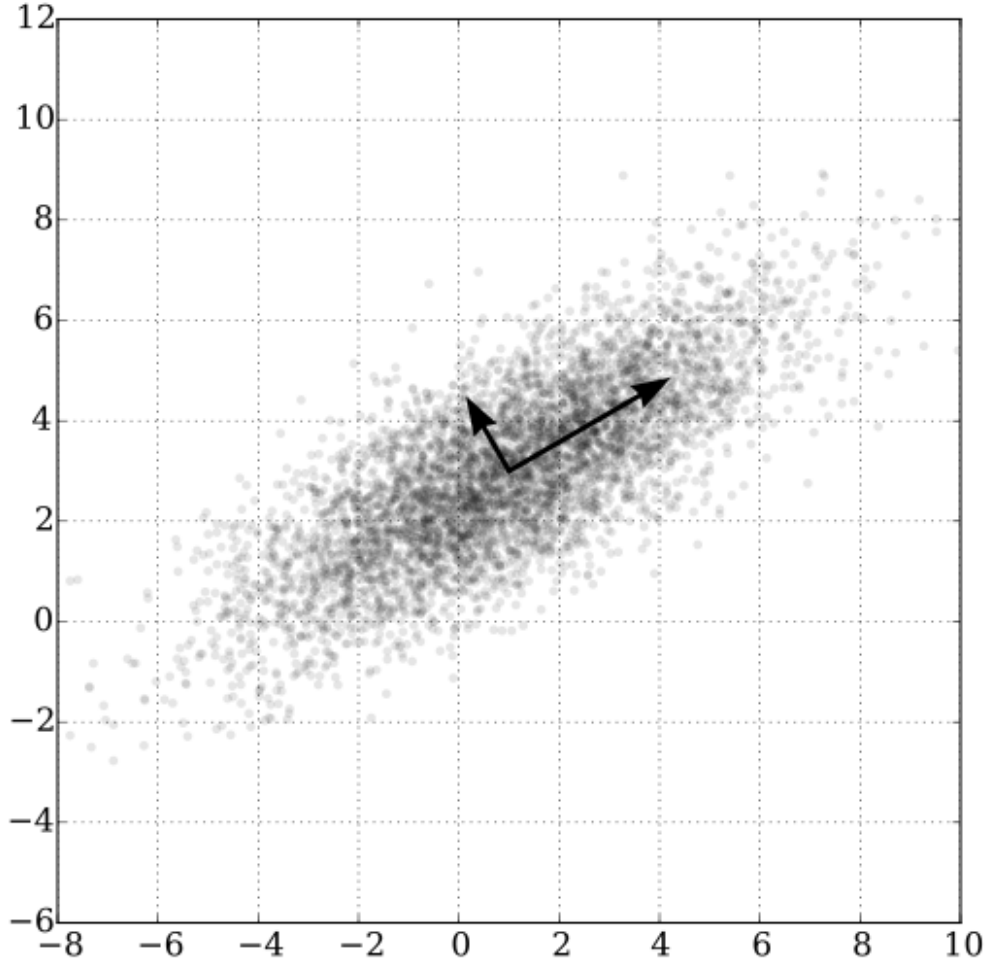
Principal Component Analysis

- Its the classic method for dimensionality reduction.
- The idea is to find a orthogonal transformation U that minimizes the reprojection error:

$$L(U) = ||X - U^T XU||^2$$

- Where the rows of U have unit length ($||u_i|| = 1$) and are orthogonal to each other.
- PCA tries to find a basis representation (the u_i's) that minimizes the reprojection error, and equivalently, expresses the maximum variance along each direction u_i.

Principal Component Analysis



Principal Component Analysis

1. Given data points, arrange into a matrix X of size $n \times d$.
2. Substract the mean $X = X - E[X]$.
3. Compute the covariance matrix of the mean-substracted X: $\Sigma = \text{COV}(X) = (X - E[X])(X - E[X])^T$
4. Σ is a $d \times d$ square matrix. Compute the eigenvalues λ_i and eigenvectors u_i of Σ.
5. Choose a value of k. Order eigenvectors u_i by decreasing eigenvalues λ_i. Choose the top-k eigenvectors and build matrix U consisting of the corresponding eigenvalues u_i.
6. Perform transformation as $Y = U(X - E[X])$
7. The inverse transformation (reconstruction) is $\tilde{X} = U^T Y + E[X]$

Principal Component Analysis

- If one selects $k < d$, dimensionality reduction is performed. U would be a $d \times k$ matrix, reducing the dimension of the input when multiplied with $X - E[X]$.
- The reprojection error (mean squared error) can be obtained from the eigenvalues as:

$$MSE = \sum_{i=k+1}^{d} \lambda_i$$

- This means that increasing the value of k close to d, the MSE is reduced, as the eigenvalues are decreasing.
- The variance explained (as %) by selecting the top k principal components is:

$$V_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$$

t-Stochastic Neighbor Embedding (t-SNE)

- t-SNE is a very popular dimensionality reduction method that is tailored for 2D/3D visualization.
- In a nutshell, t-SNE works by first transforming distances in the input feature space into a probability distribution. Another probability distribution with a low-dimensional support is defined and optimized in order to minimize the KL divergence between distributions.
- Then the low-dimensional support distribution is sampled to obtain the embedding representation. This puts the stochastic part of t-SNE.

t-Stochastic Neighbor Embedding (t-SNE)

- First the probabilities between input points $p_{j|i}$ are defined. σ_i values are selected to finely cover the input space:

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

- Then the low-dimensional support distribution q_{ij}, where the y values are learnable through gradient descent:

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

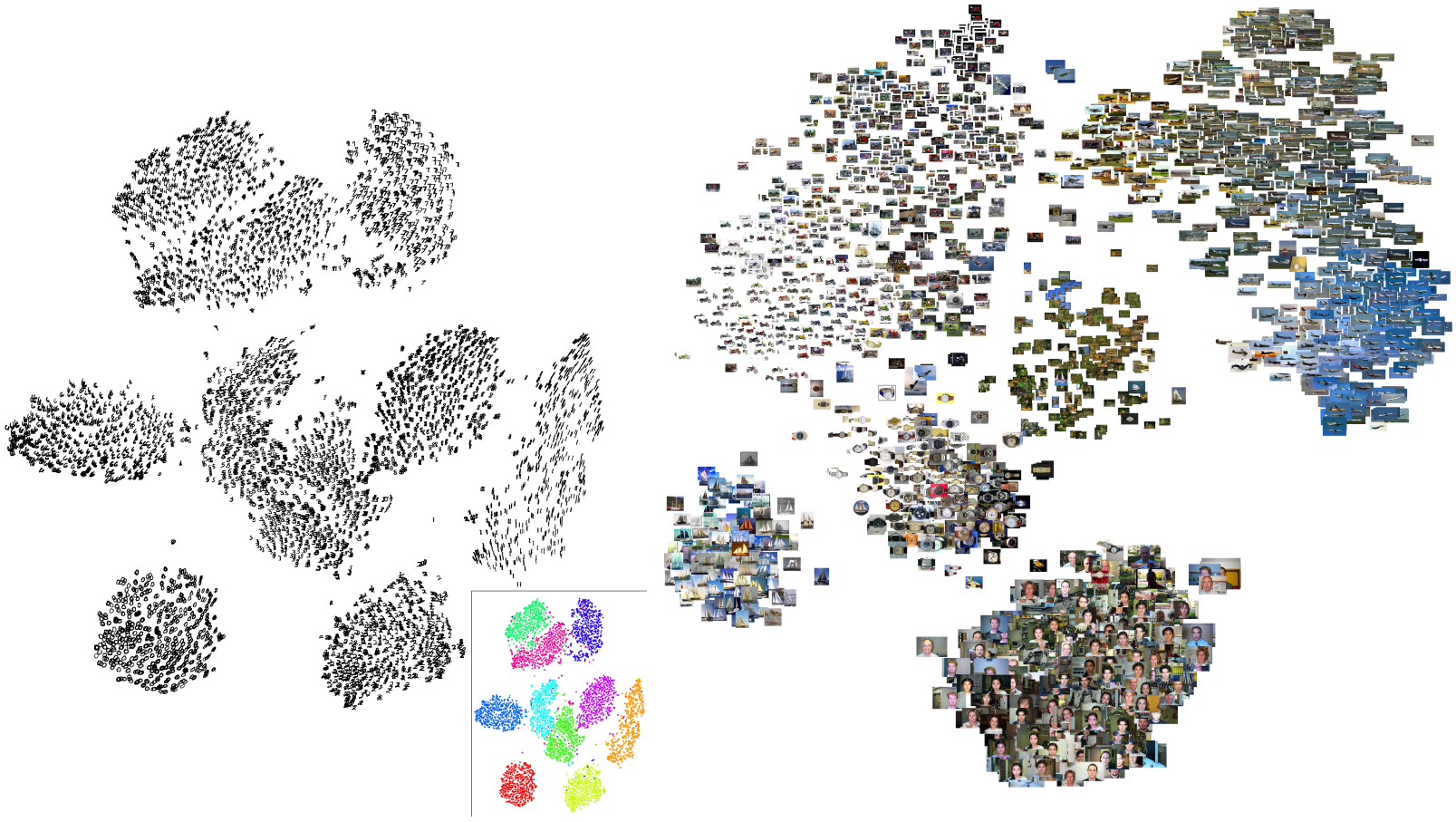
- The q_{ij} distribution is selected to mimic a student's t-distribution, which is robust to outliers.
- Then the KL-Divergence is minimized by moving the y values:

$$KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

t-Stochastic Neighbor Embedding (t-SNE)

It can be used to visualize any method.

This method is not parametric. Each embedding is problematic. That is the reason is used mostly for visualization and not for classification.



Multi-Dimensional Scaling

- Another dimensionality reduction method, similar to t-SNE, but not stochastic.
- t-SNE does not try to preserve distances in the high-dimensional space, but MDS does at least approximate distances from the high to the low-dimensional manifold.
- MDS first computes pair-wise distances between points x in the high-dimensional input space:

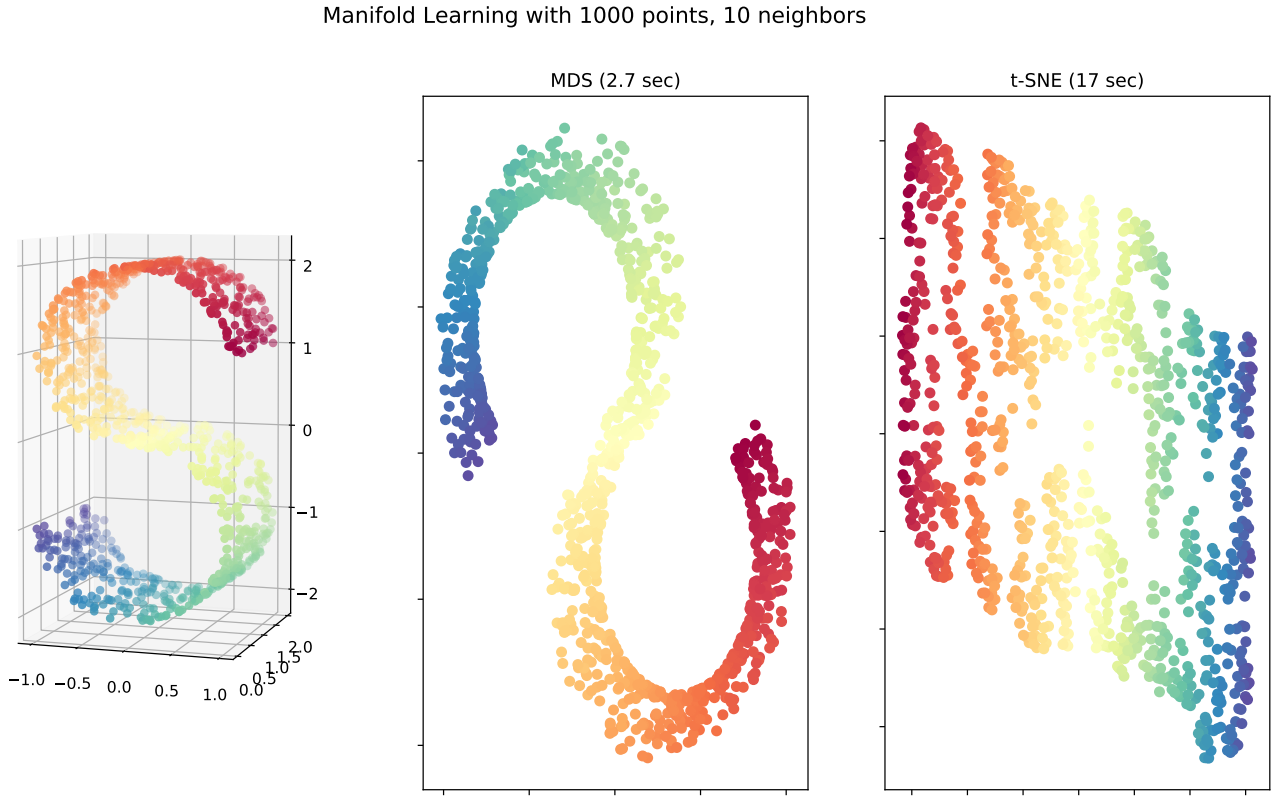
$$d_{ij} = ||x_i - x_j||$$

- Then a low-dimensional embedding y is optimized to closely mimic the pair-wise distances:

$$L(y) = \sum_{i \neq j} (d_{ij} - ||y_i - y_j||)^2$$

- The minimization of loss L can be performed using gradient descent. Generally different solutions are obtained on each run, but as distances are approximated, they usually correspond to rotations of the low-dimensional space.

t-SNE vs MDS

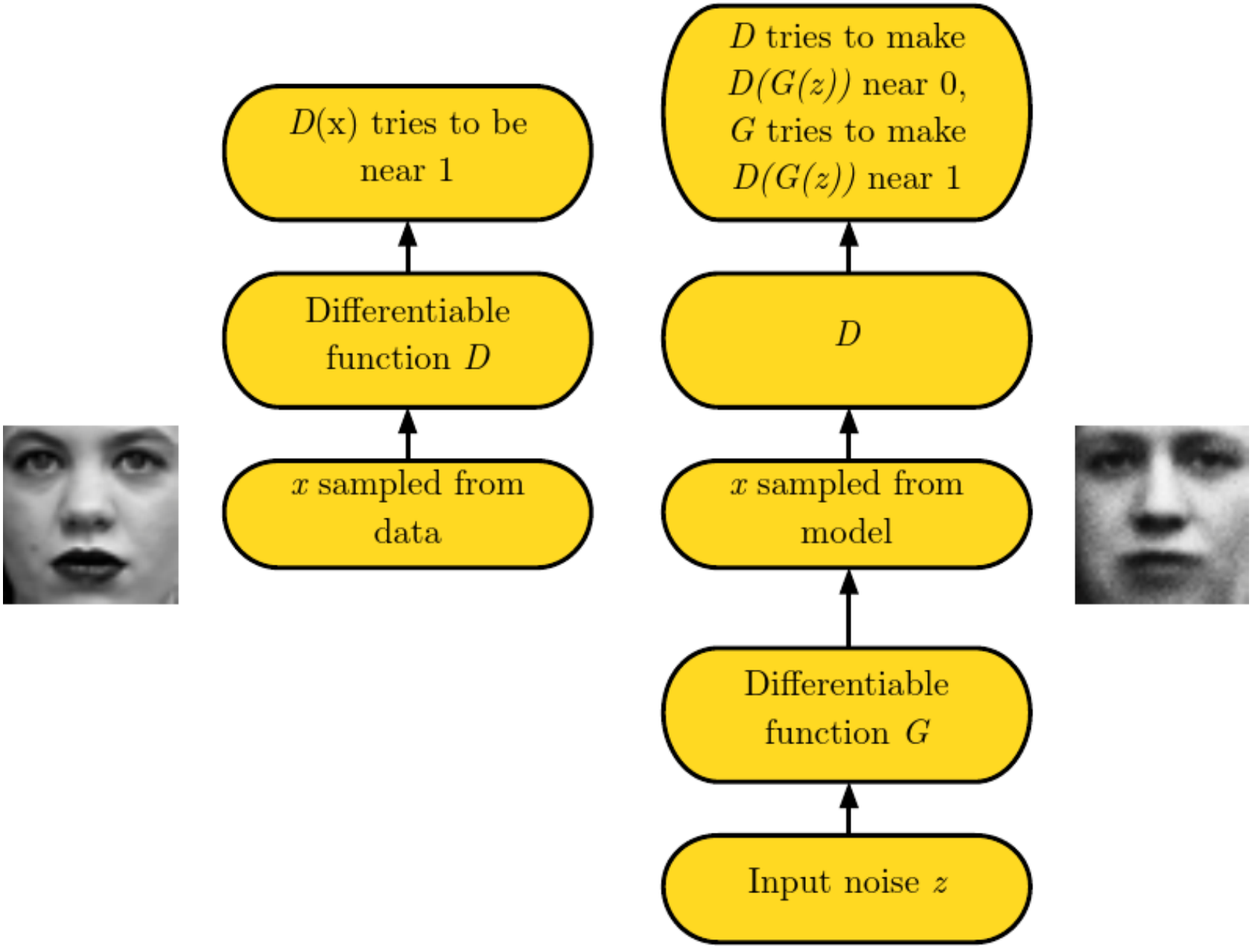

Manifold Learning with 1000 points, 10 neighbors
MDS (2.7 sec)
t-SNE (17 sec)
2
1
0
−1
−2
−1.0
−0.5
0.0
0.5
1.0
0.0
0.5
1.0
1.5
2.0

Generative Adversarial Networks

Generative Adversarial Networks

- A Generative Adversarial Network is a special kind of generative model.
- The idea is to **learn** a probability distribution directly from data generated by that distribution.
- For example, given a dataset containing images, learn the high-dimensional distribution that generated those images.
- And then use it to generate new images from a noise source.
- The key concept is adversarial training.

Generative Adversarial Networks



Generative Adversarial Networks

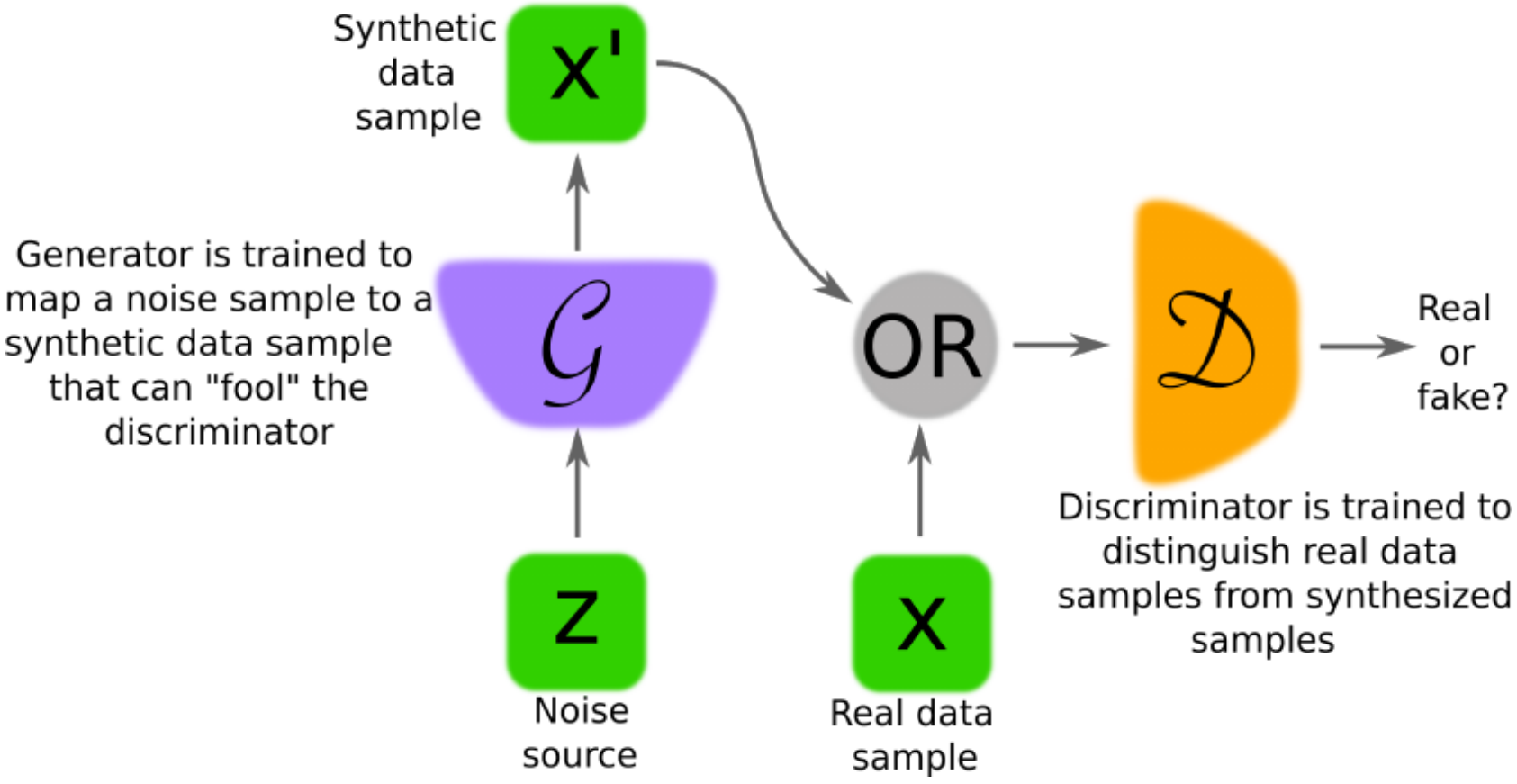
Hard to train.

- G is called the generator and it is a neural network that takes a noise vector z and transforms it into a sample from the real distribution.
- z can be considered as a learned latent space that can be used to explore it.
- D is called the discriminator and it is also a neural network, but it discriminates real samples from ones generated by the discriminator.
- One important property of GANs is that they are **unsupervised**.

You create a batch of real images and of generated images using the GAN.

Generative Adversarial Networks

G and D are trained together. At some point G gets really good that it gets really hard to D to discriminate the images. When this happens we reach an equilibrium.



Generative Adversarial Networks

GANs are trained as a mini-max game:

$$\max_D \min_G V(G, D)$$

Where:

$$V(G, D) = \mathbb{E}_{p_{\text{data}}(x)} \log D(x) + \mathbb{E}_{p_g}(x) \log(1 - D(x))$$

The update rules for G and D are different. Training G means performing gradient descent on V, while training D is done by gradient ascent on V. This shows the adversarial training game.

Initial GAN results (2014)

a)

b)

c)

d)

Training GANs

- Training a GAN is not easy, because there are two players playing versus each other.
- A common simplification is to train the Discriminator while holding the Generator constant (non-trainable), and then train the Generator while holding the Discriminator as constant.
- Common problems while training is that one player overpowers the other.
- If the Discriminator is "too good", it will produce values close to 0 and 1, and the gradient of the Generator will be very low, leading to vanishing gradients.
- Similarly, if the Generator is "too good" it can exploit structural weaknesses in the Discriminator, leading to non-realistic outputs.

Mode Collapse

- Training GANs is hard due to the instability of the training process.
- Unfortunately there are no rules to identify when GAN training has converged to a good point, unlike most classification and regression models.
- The Mode Collapse problem is when the generator collapses into only predicting the mean/median/mode of the training data, instead of fully covering the whole probability distribution domain.
- In that case one has to restart the training process.

Mode Collapse

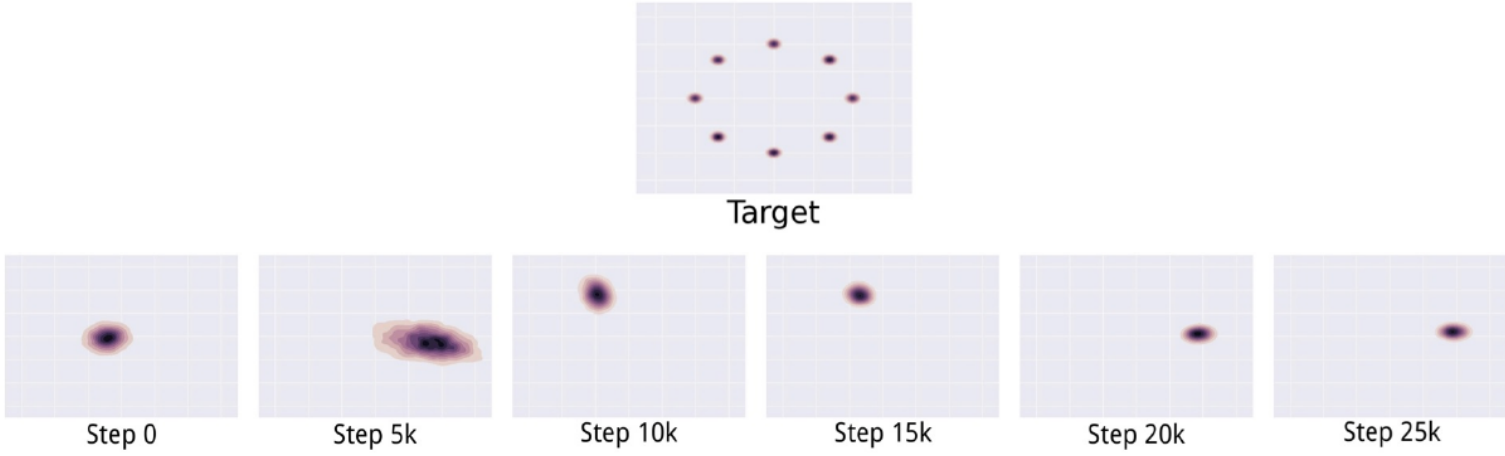


Figure 22: An illustration of the mode collapse problem on a two-dimensional toy dataset. In the top row, we see the target distribution p_{data} that the model should learn. It is a mixture of Gaussians in a two-dimensional space. In the lower row, we see a series of different distributions learned over time as the GAN is trained. Rather than converging to a distribution containing all of the modes in the training set, the generator only ever produces a single mode at a time, cycling between different modes as the discriminator learns to reject each one. Images from Metz *et al.* (2016).

DCGAN (2014)

Figure 18: Samples of images of bedrooms generated by a DCGAN trained on the LSUN dataset.

DCGAN (2014)

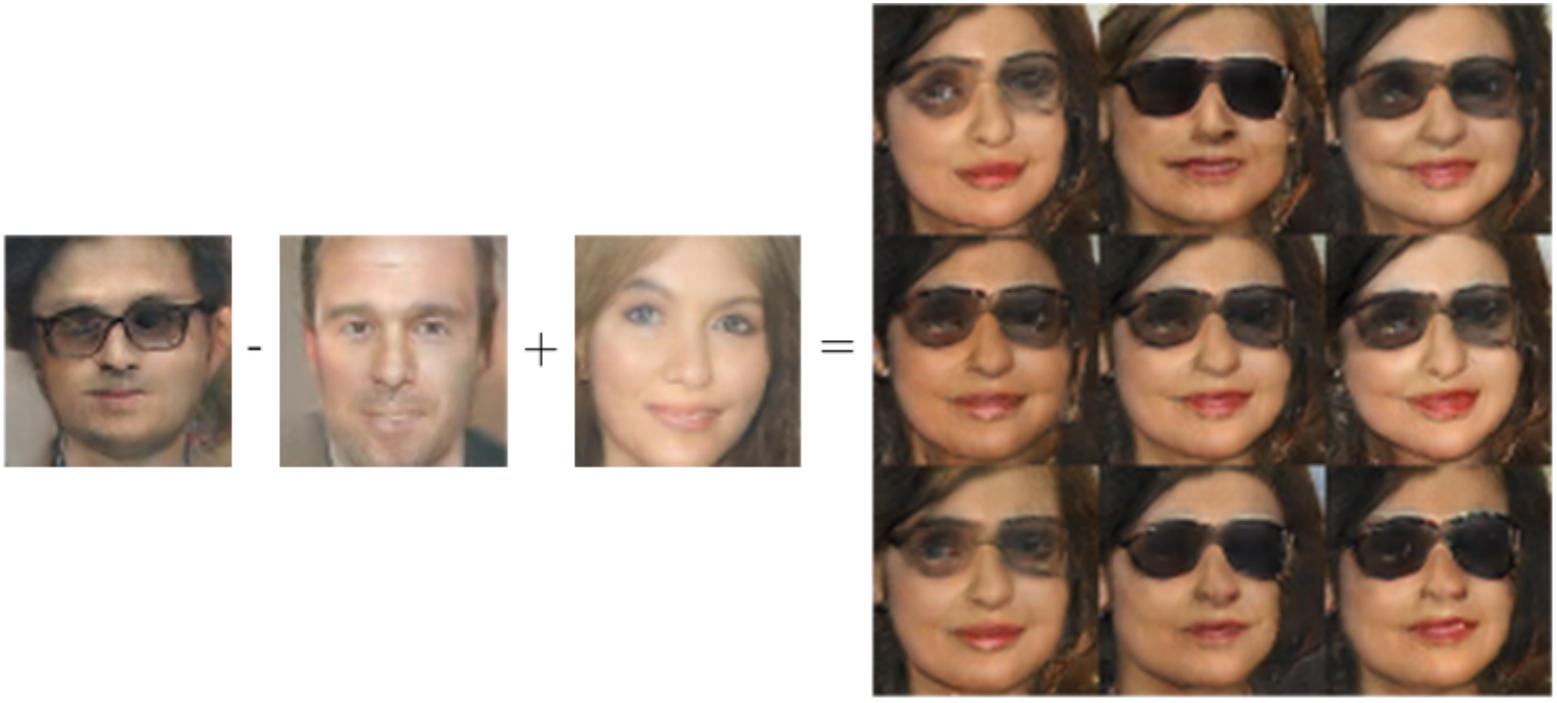
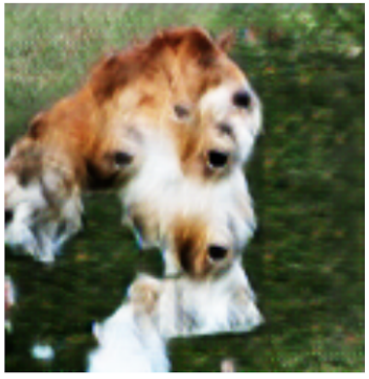


Figure 19: DCGANs demonstrated that GANs can learn a distributed representation that disentangles the concept of gender from the concept of wearing glasses. If we begin with the representation of the concept of a man with glasses, then subtract the vector representing the concept of a man without glasses, and finally add the vector representing the concept of a woman without glasses, we obtain the vector representing the concept of a woman with glasses. The generative model correctly decodes all of these representation vectors to images that may be recognized as belonging to the correct class. Images reproduced from Radford *et al.* (2015).

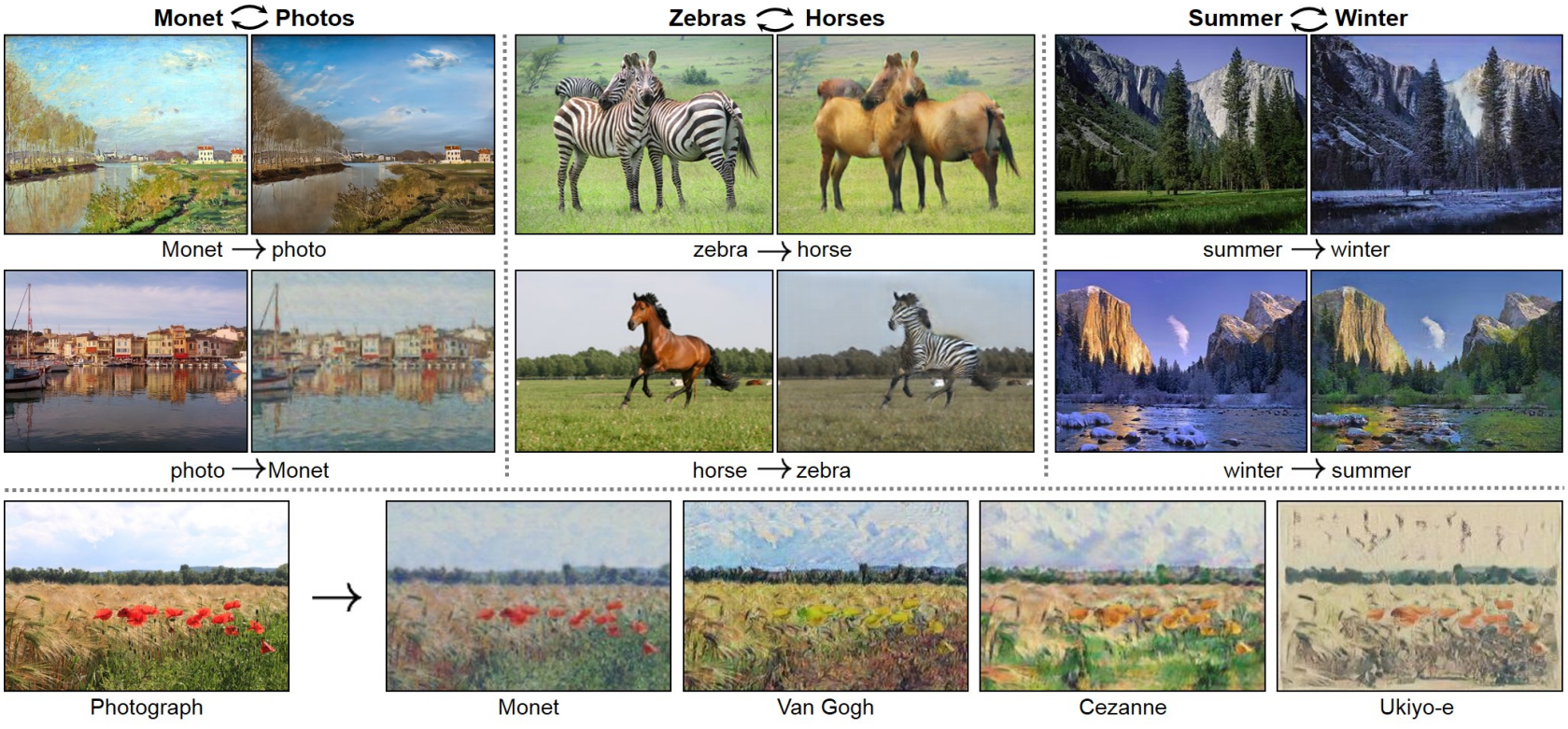
Failure Cases





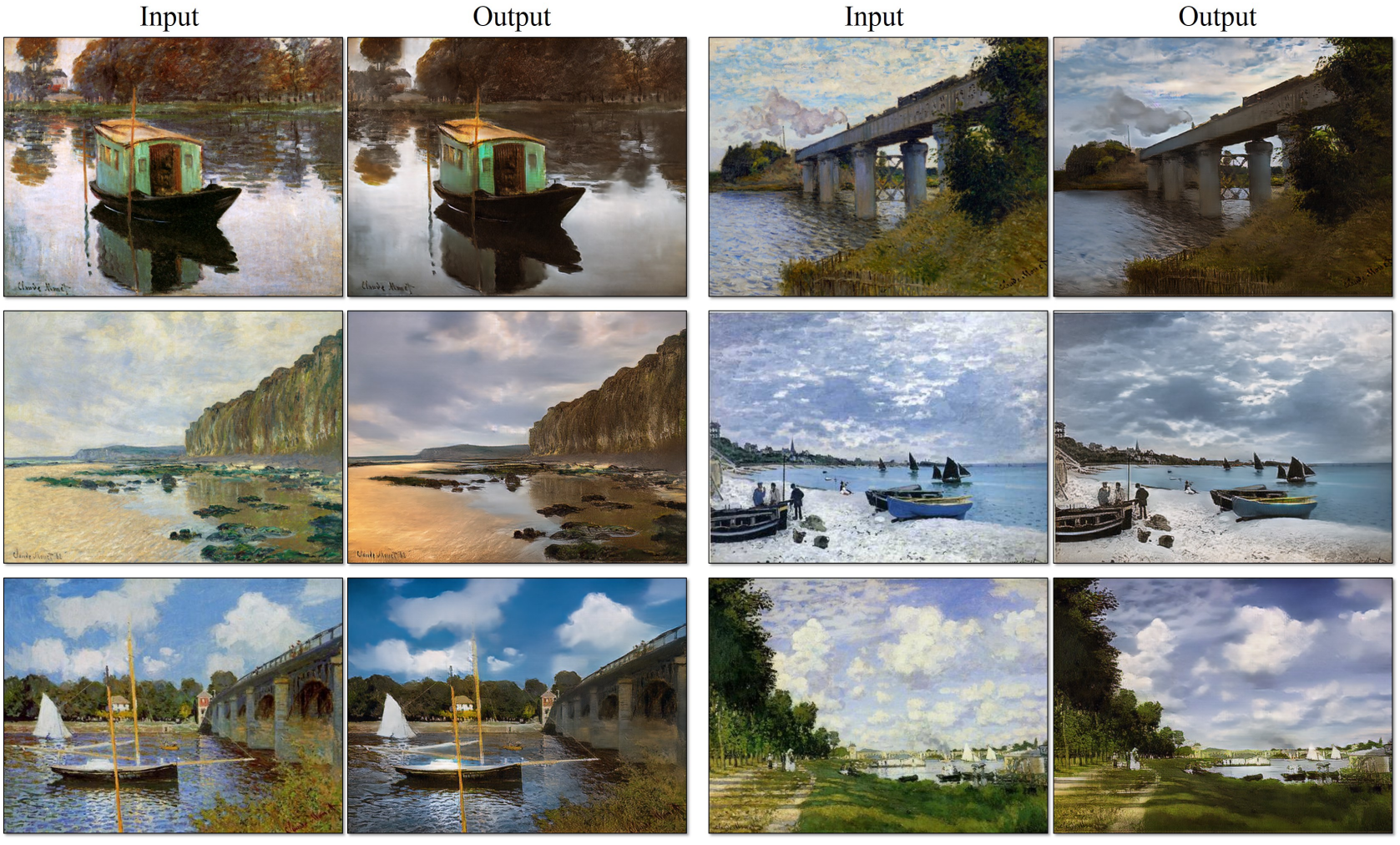

Failure Cases

CycleGAN (2017)

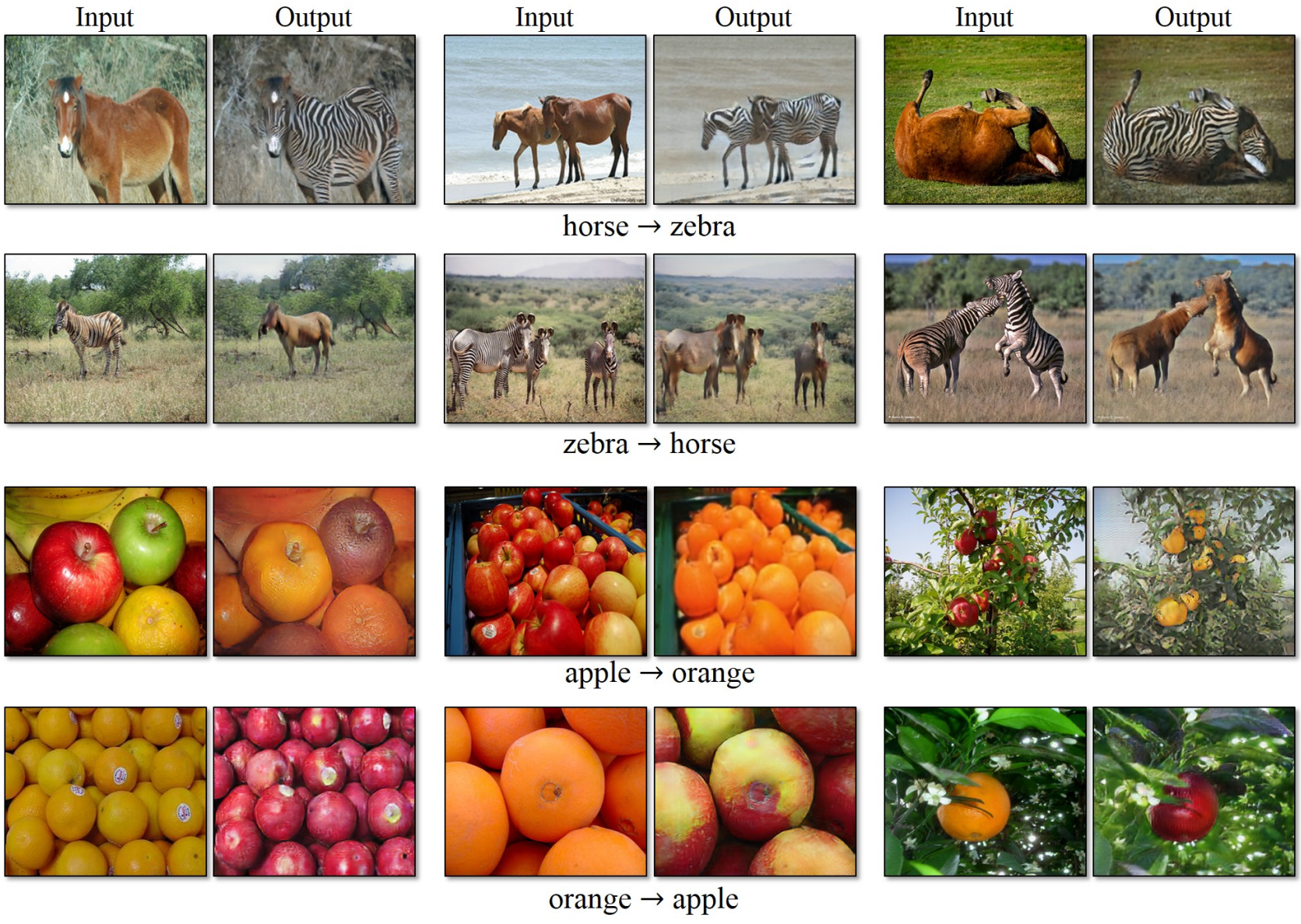


CycleGAN Style Transfer (2017)

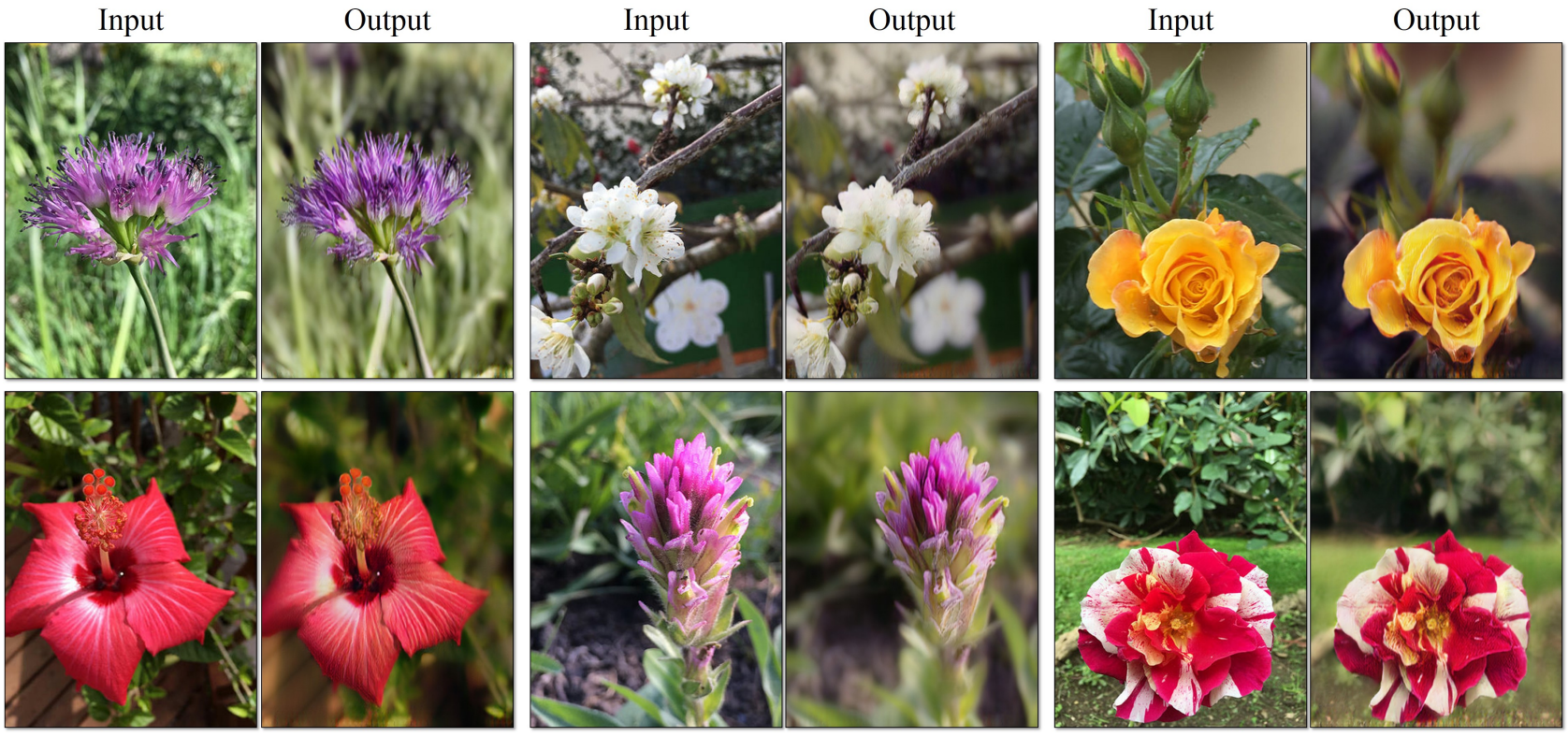
CycleGAN Money to Photo (2017)



CycleGAN Object Transfer (2017)



CycleGAN Depth of Field (2017)



Readings & References

- "Dimensionality Reduction by Learning an Invariant Mapping" by Hadsell et al. 2006
- "FaceNet: A Unified Embedding for Face Recognition and Clustering" by Schroff et al. 2015
- "Visualizing Data using t-SNE" by van der Maaten and Hinton. 2008.
- "Tutorial on Variational Autoencoders" by Doersch, 2016.
- "NIPS 2016 Tutorial: Generative Adversarial Networks" by Goodfellow. 2016.

Questions?