California Institute of Technology (TAPIR)
ARC Seminar
Nicholas Z. Rui

Dimensionality Reduction
3 April 2023

Outline *These Notes summarize various computational methods for dimensionality reduction. The concept of dimensionality reduction is introduced and motivated, and then linear and nonlinear techniques are reviewed. We conclude with a discussion of autoencoders.*

Contents

1 Introduction and Motivations

Real-life datasets often consist of high-dimensional data which is constrained to lie on a much lower-dimensional manifold by common physical origins. Methods which embed the majority of the natural variation in the data into a space of (possibly much) smaller dimension fall into the purview of **dimensionality reduction**.

One very simple example of this phenomenon is the **Hertzsprung–Russell diagram (HRD)** in astrophysics, which plots stars and star-like objects on a two-dimensional plane spanned by their luminosity and effective (surface) temperature (e.g., Figure 1).

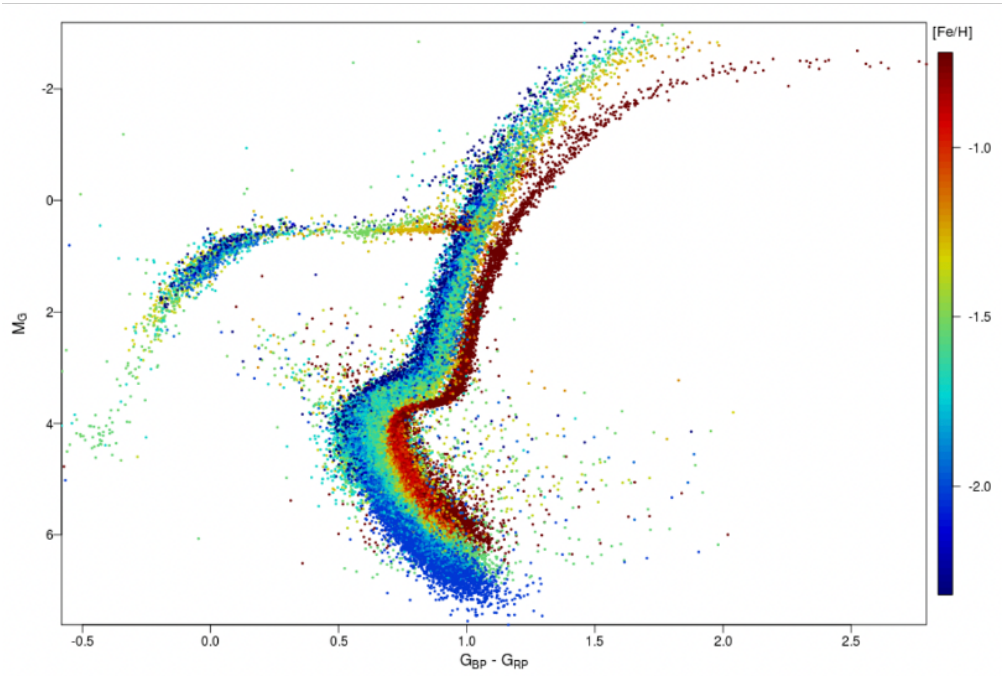

Figure 1: A Hertzsprung–Russell diagram showing fourteen globular clusters (of comparable age) colored by metallicity, reproduced from Figure 3 of [1]. At fixed metallicity, we see that the stars lie on an approximately one-dimensional subset of the whole space.

While naïvely a star may lie anywhere on the Hertzsprung–Russell diagram, in reality stars of a common age lie on a one-dimensional curve in this space. The reason for this is simple: the variation in luminosity and effective temperature is *dominated* by the initial mass of the star only (for fixed age), two variables which are only a function of a single variable will have an intrinsic dimension of only one. Stars may lie off of this curve, but they only do so because there is some other variable besides initial mass that is playing a role. For example, stars which are observed to be over-bright are likely to be binaries. In this way, examining the dimensionality of the space and observing outliers can at once inform the physics that we know and also point to physics that we might not.

As another example, [2] find that, over a selected set of independently measurable structural properties, **globular clusters** lie on a three-dimensional manifold called the **King manifold**. Since

the majority of the variation in the morphology of globular clusters is due to three variables, the initial mass, size, and dynamical age of the cluster, this is hardly unexpected.

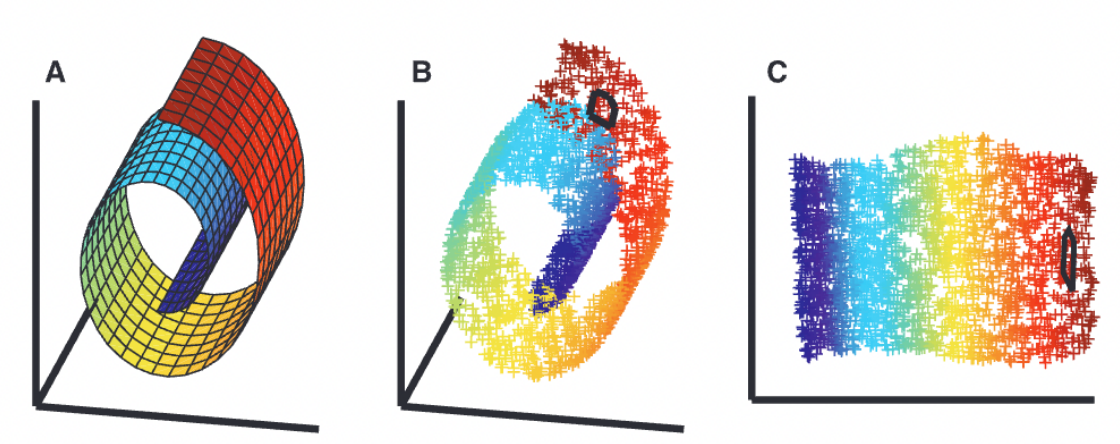

Figure 2: An example of dimensionality reduction on a highly nonlinear **Swiss roll** dataset, reproduced from Figure 1 of [3].

There are many reasons to perform dimensionality reduction. Large dimensional spaces (such as those spanned by images) are incredibly unwieldy, and suffer from the **curse of dimensionality** which often makes sampling such spaces near impossible. Datasets with reduced dimensionality are easier to **visualize** (Figure 2) and **store**, and a proper understanding of the subset of the full space actually populated by data can even inform **generative models**, which can *simulate* data (such as images, which are contemporary interest). Finally, because data points which resist dimensionality reduction may do so for some physical reason, dimensionality reduction is a strong tool for **anomaly detection**.

These Notes review a few prominent methods in dimensionality reduction. They are by no means comprehensive.

2 Principal component analysis (PCA)

Consider n data points $\{\vec{X}^{(i)}\}$ comprising an N-dimensional dataset. One can define the **covariance matrix** of the data as follows:

$$\mathbf{\Sigma} = \frac{1}{n-1}\sum_{j=1}^{n}\left(\vec{X}^{(i)} - \langle\vec{X}\rangle\right)\left(\vec{X}^{(i)} - \langle\vec{X}\rangle\right)^{\top} \tag{1}$$

where $\langle\vec{X}\rangle$ is the mean data point. Note that the diagonal components of this matrix are the **variances** in each of the basis directions:

$$\Sigma_{ii} = \frac{1}{n-1}\sum_{j=1}^{n}\left|X_i^{(j)} - \langle X_i\rangle\right|^2 = \sigma_i^2 \tag{2}$$

The off-diagonal elements (called **covariances**) reflect the (linear) correlations between the values of the data points in two different basis directions.

Note that $\mathbf{\Sigma}$ is a **real symmetric** matrix (as can be seen in Equation 1). It is also **positive semidefinite** (in the sense that $\vec{X}^\top \mathbf{\Sigma} \vec{X} \geq 0$ for all vectors $\vec{X}$). These two faacts mean that it is guaranteed to have an orthogonal eigenbasis with non-negative eigenvalues.

One very clever thing that we can do is change the basis of our data points, $\vec{X}' = \mathbf{U}\vec{X}$, to the eigenbasis of $\mathbf{\Sigma}$. Then, in this basis, $\mathbf{\Sigma}' = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$ is diagonal, and its diagonal elements are the variances along each eigendirection. In other words, the size of each element reflects how much the data points vary amongst each other in a given eigendirection.

Because the *whole idea* of dimensionality reduction is to capture as much of the variation of the data as possible with as few numbers as possible, it is natural to *project out* all of the components of each data point in the eigenbasis except for those associated with the largest few eigenvalues. If the data is intrinsically d-dimensional (where perhaps $d \ll N$), all except n of these eigenvalues will be small, and so you do not really lose much information by discarding the others (since these components will be close to zero in most/all of the data points).

This procedure is called **principal component analysis (PCA)**. The visual interpretation of this method is to "rotate" to a perspective where the data are uncorrelated, and to only record the components in the directions of maximum variation.

Note that the covariance matrix only properly captures linear correlations, and so principal component analysis is a **linear** method. More sophisticated methods are needed to reduce the dimensionality of a dataset which has more complicated variations. However, many datasets have correlations which are close to linear, and PCA may be a good preprocessing step even if this is not the case.

3 Nonlinear techniques

3.1 Kernel PCA

Suppose that we have a function $k(\vec{x}, \vec{y})$ (called a **kernel**), which should be understood as intuitively measuring the "closeness" of two data points. This need not be the actual proximity of two data points in the original space, since this may well not be true for data with nonlinear correlations.

A kernel must be **symmetric** ($k(\vec{x}, \vec{y}) = k(\vec{y}, \vec{x})$) and positive semidefinite ($k(\vec{x}, \vec{y}) \geq 0$, $\forall\, \vec{x}, \vec{y}$). This should comport with our intuitive understanding of what it should mean for two data points to be close. These conditions are collectively called **Mercer's condition**, and are the conditions of validity for **Mercer's theorem**.

Mercer's theorem states that a kernel k (together with some data points $\{\vec{X}^{(i)}\}$) is associated to an *inner product* matrix (called a **Gramian**, $\mathbf{G}$) in *some* higher-dimensional space. The Gramian is related to the kernel by

$$G_{ij} = k(\vec{X}^{(i)}, \vec{X}^{(j)}) \tag{3}$$

One can imagine that there is some (generally nonlinear map) $\varphi : \mathbb{R}^N \to \mathbb{R}^D$ which takes data points from their native space to some D-dimensional **feature space**. Then the Gramian represents the

inner product in this feature space, as

$$G_{ij} = \varphi(\vec{X}^{(i)})\varphi(\vec{X}^{(j)})^\top = \vec{Y}^{(i)}\vec{Y}^{(j)\top} \tag{4}$$

where $\vec{Y}^{(i)} = \varphi(\vec{X}^{(i)})$. One can then find the feature-space data points $\{\vec{Y}^{(i)}\}$ by writing

$$\mathbf{G} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \mathbf{U}\mathbf{\Lambda}^{1/2}\left(\mathbf{U}\mathbf{\Lambda}^{1/2}\right)^\top \tag{5}$$

where $\mathbf{U}$ is the change-of-basis matrix which represents the Gramian as a diagonal matrix $\mathbf{\Lambda}$. We see that $\vec{Y}^{(i)}$ is just the ith column of $(\mathbf{U}\mathbf{\Lambda}^{1/2})^\top$, i.e., the jth component of $\vec{Y}^{(i)}$ is

$$\vec{Y}^{(i)}_j = \sqrt{\lambda_j}\vec{e}^{(j)}_i \tag{6}$$

where $\vec{e}^{(j)}$ is the jth eigenvector of $\mathbf{G}$, and λ_j is its associated eigenvector. This latter procedure is representative of **classical multidimensional scaling (MDS)**.

This doesn't seem inherently useful yet because we have just mapped our N-dimensional data to an N-dimensional space, which probably doesn't reduce the dimension. However, once again, we can discard all the components of $\vec{Y}^{(i)}$ except those associated to the largest eigenvalues λ_i. We see that the others will be small anyway, since the components are proportional to $\sqrt{\lambda_i}$. This whole procedure is called **kernel PCA**.

There is some flexibility in the choice of kernel. An example kernel (which satisfies Mercer's condition) might be a **Gaussian kernel**, parameterized by some σ:

$$k(\vec{x}, \vec{y}) = \exp\left(-\frac{1}{2\sigma^2}\left|\vec{x} - \vec{y}\right|^2\right) \tag{7}$$

Note that, throughout this whole process, we did not need to know the map φ from the native to the feature space at all. By specifying a kernel, we were able to get around doing this. This is called the **kernel trick**.

3.2 Local linear embedding (LLE)

If one looks closely enough at a data manifold, one should in principle observe that close-by data points will seem to be arranged on a plane: the tangent plane of the manifold. If this is the case, it should be possible to reconstruct any given data point as a linear combination of its neighboring data points. Leveraging this idea for dimensionality reduction is the basic idea behind **local linear embedding (LLE)** (Figure 3). This algorithm was introduced by [3], and is discussed further by [4].

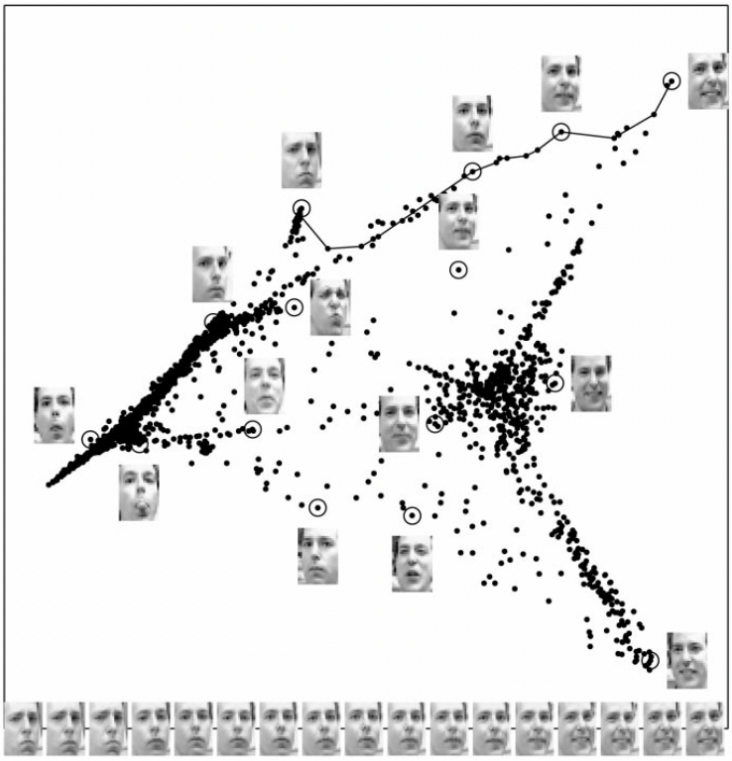

Figure 3: The output of local linear embedding on a dataset of faces, reproduced from Figure 3 of [3].

To be quantitative, we might like to minimize the cost function

$$J_1(\mathbf{W}) = \sum_i \left| \vec{X}^{(i)} - \sum_j W_{ij} \vec{X}^{(j)} \right|^2 \tag{8}$$

where the optimization is over the matrix of weights $\mathbf{W}$ required to reconstruct all the data points from their neighbors. We enforce that $W_{ii} = 0$, and also that $W_{ij} \neq 0$ only for the top k nearest neighbors (k is the only free parameter in this method).

One then uses those same weights to optimize over data embeddings in the smaller space:

$$J_2(\{\vec{Y}^{(i)}\}) = \sum_i \left| \vec{Y}^{(i)} - \sum_j W_{ij} \vec{Y}^{(j)} \right|^2 \tag{9}$$

More formally, this method works because the action of encoding our data as a low-dimensional space will *locally* be to rotate, translate, and rescale a patch of data points, and this is reconstruction is exactly respected by a linear combination of those data points. The weights should therefore be the same in both the original and final spaces.

There are a few more considerations before this method will fully work.

First, we must enforce that

$$\sum_i W_{ij} = 1 \tag{10}$$

in the first optimization. This is required by translational symmetry in $J_1(\mathbf{W})$: it can be seen that this must be true if we impose invariance of this cost function under shifts of all the data points by a uniform vector $\vec{A}$.

Second, suppose for practicality that we impose a centering constraint

$$\sum_i \vec{Y}^{(i)} = 0 \tag{11}$$

In the final optimization, it is required to normalize the variance of $\vec{Y}^{(i)}$ to (customarily) one, i.e.,

$$\frac{1}{n}\sum_i \vec{Y}^{(i)}\vec{Y}^{(i)\top} = 1 \tag{12}$$

This is because, if we do not do this, then placing all $\vec{Y}^{(i)} = 0$ (the "degenerate solution") will totally minimize the cost function $J_2(\{\vec{Y}^{(i)}\})$, which is clearly undesirable.

Subject to these considerations, LLE is in principle a very effective dimensionality reduction method. However, it is somewhat susceptible to "cheating" the variance constraint (Equation 12), since the optimization may decide to place most of the data points at the origin while creating long, finger-like structures away from the origin which are made to contain most of the variance of the dataset.

3.3 Isomap

In the original space, the Euclidean distance is not necessarily a good representation of similarity. However, **geodesic distance** probably is. Geodesic distance is, intuitively, the distance between two points *along* the manifold. Because dimensionality reduction can roughly be thought of as "flattening out" a manifold, the geodesic distance should be preserved by the procedure.

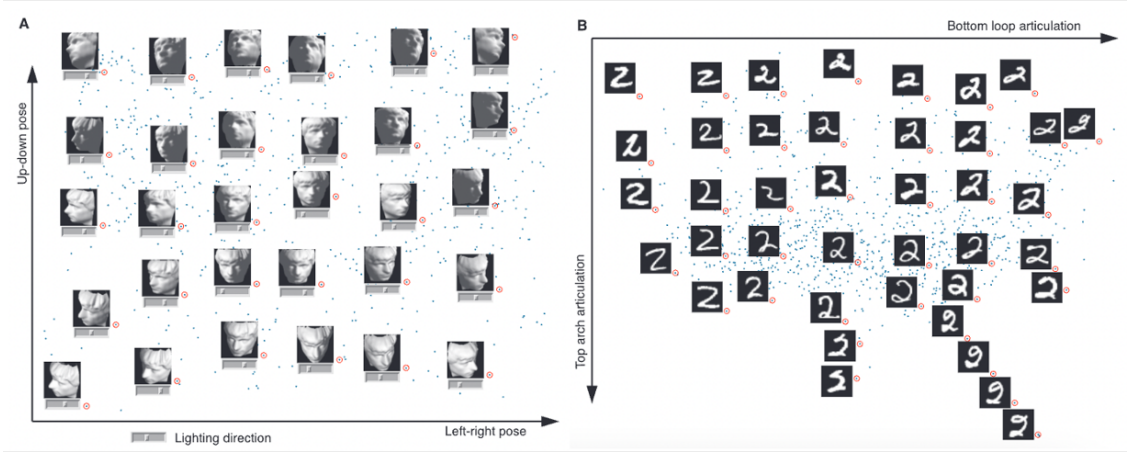

Figure 4: The output of the Isomap algorithm on datasets of images of statues (*left*) and handwritten digits from the MNIST dataset (*right*), reproduced from Figure 1 of [5].

Geodesic distances lie at the heart of the **Isomap** algorithm, first introduced by [5] (Figure 4). First, a **neighboorhood graph** (connecting points considered to be "close together") is constructed, typically either by connecting each point to its k closest neighbors or, alternatively, all points within some Euclidean distance of it. Next, the shortest distance between every pair of points is computed as the length of the shortest graph path between them, as computed by, e.g., **Dijkstra's algorithm**. These distances (which can be encoded in a squared distance matrix $\mathbf{D}^2$) can be converted to a Gramian $\mathbf{G}$ using a "centering matrix" $\mathbf{H}$ (with $H_{ij} = \delta_{ij} - 1/n$) as

$$\mathbf{G} = -\frac{1}{2}\mathbf{H}\mathbf{D}^2\mathbf{H} \tag{13}$$

Finally, classical MDS is applied to turn the Gramian into a lower-dimensional embedding.

Isomap can be a very effective tool for dimensionality reduction. However, it can be computationally expensive for large datasets (due to its shortest path graph search), and does not deal well with manifolds with "holes." The first of these issues can be ameliorated somewhat by using an adaptation of Isomap called **Landmark Isomap** (e.g., [6]), which computes the distance from every data point to some subset of $q < n$ data points, called **landmarks**, which sacrifices some fidelity for computational speedup.

3.4 Maximum variance unfolding (MVU)

Dimensionality reduction can be thought of as "unfolding" a manifold. The action of unfolding is, roughly speaking, to try and pull a surface apart while not ripping or stretching it. **Maximum variance unfolding (MVU)** [7] is an algorithm based on this intuition.

As in Isomap, a neighborhood graph is constructed, either using the k closest neighbors or some threshold distance. The embedded data points $\{\vec{Y}^{(i)}\}$ are initialized in the original space at the same locations as the original data $\{\vec{X}^{(i)}\}$, and an optimization is then done to maximize the cost

function

$$J(\{\vec{Y}^{(i)}\}) = \frac{1}{2n}\sum_{ij}|\vec{Y}^{(i)} - \vec{Y}^{(j)}|^2 = \frac{1}{n}\operatorname{Tr}\mathbf{G} \tag{14}$$

where $\mathrm{G} = \vec{Y}^{(i)} \cdot \vec{Y}^{(j)}$ is the Gramian matrix again. This optimization is done while fixing the distances between connected neighbors on the neighborhood graph as constraints. By maximizing the variance of the data subject to fixing close distances, MVU "unwraps" the manifold until it becomes flat.

The optimization in MVU is convex, implying a unique maximum, and falls into a family of **semidefinite programs (SDPs)** (for this reason, MVU is sometimes referred to as **semidefinite embedding (SDE)**). However, because MVU requires an optimization involving a matrix of size $n \times n$, it quickly becomes infeasible for $n \gtrsim \text{few} \times 10^3$. Runtime improvements are implemented in methods such as **Landmark MVU** which, like Landmark Isomap, uses a subset of points as *landmarks* to trade off fidelity against speed [8].

3.5 *t*-Distributed stochastic neighbor embedding (*t*-SNE)

One major motivation for dimensionality reduction is that humans have major trouble visualizing spaces in dimensions $d > 3$. Dimensionality reduction techniques can thus serve as helpful visualization tools that somehow preserve "similarity" through proximity in an embedding, *even if* the original data is not intrinsically low-dimensional.

Using similar intuition to MVU, we would like to preserve the distances between nearest neighbors during our optimization procedure. However, we may seek to do this probabilistically, since iut may not be incredibly important to map extremely close neighbors extremely close to each other (since neighbors closer than a certain threshold to a point are probably comparably close to each other, with closer neighbors probably being flukes). We can associate to each point some spread σ_i, and then define a normalized probability that a Gaussian distribution of width σ_i centered around point i will draw point j (as opposed to others, excluding i):

$$p_{j|i} = \frac{\exp\left(-|\vec{X}^{(j)} - \vec{X}^{(i)}|^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-|\vec{X}^{(k)} - \vec{X}^{(i)}|^2/2\sigma_i^2\right)} \tag{15}$$

In practice, the spreads σ_i are not specified directly. Instead, they are specified using an information-theoretic quantity called the **perplexity** $\mathcal{P}$, which, roughly speaking, fixes σ_i so that point i has roughly $\mathcal{P}$ nearest neighbors.

A lower-dimensional embedding should ideally preserve the probabilities $p_{j|i}$. More specifically, we can define the probabilities

$$q_{j|i} = \frac{\exp\left(-|\vec{Y}^{(j)} - \vec{Y}^{(i)}|^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-|\vec{Y}^{(k)} - \vec{Y}^{(i)}|^2/2\sigma_i^2\right)} \tag{16}$$

which are defined similarly to $p_{j|i}$ but for the embedded data points $\{\vec{Y}^{(i)}\}$, which lie in a smaller dimensional space $d < N$, typically $d = 2$ or 3. Enforcement of similarity of these two sets of

probabilities is enforced by minimizing the **Kullback–Leibler (KL) divergence**, which is an information-theoretic measure of the distance between probability distributions:

$$C \equiv \mathrm{KL}(Q|P) = \sum_{i \neq j} p_{i|j} \log\left(p_{i|j}/q_{i|j}\right) \tag{17}$$

The gradient of C is given by

$$\frac{\partial C}{\partial \vec{Y}^{(i)}} = 2 \sum_j (p_{i|j} - q_{i|j} + p_{j|i} - q_{j|i})(\vec{Y}^{(i)} - \vec{Y}^{(j)}) \tag{18}$$

which strongly resembles the equations for springs (with spring constants dependent on the probability dissimilarities) connecting all the data points together (here, C has the interpretation of an energy). Because of its resemblance with N-body dynamics, this optimization often borrows the **Barnes–Hut approximation** [9] from stellar dynamics.

The aforementioned technique is called **stochastic neighbor embedding (SNE)** [10]. However, in recent years, a vastly more popular variant, called ***t*-distributed stochastic neighbor embedding (*t*-SNE)**[1] [11], redefines $q_{j|i}$ (but not $p_{j|i}$) to follow a (thick-tailed) Student's t-distribution, i.e.,

$$q_{j|i} = \frac{(1 + |\vec{Y}^{(j)} - \vec{Y}^{(i)}|^2)^{-1}}{\sum_{k \neq l} (1 + |\vec{Y}^{(k)} - \vec{Y}^{(l)}|^2)^{-1}} \tag{19}$$

The reason for this is, if the data are not intrinsically low-dimensional, then you want the freedom to map points which are far away from each other also far away from each other in the flattened space. Because higher-dimensional spaces are "bigger," it is sensible to relax the $q_{j|i}$ values to a fat-tailed distribution, so that the optimizer is not penalized for overestimating how far apart two distant data points are.

In t-SNE, [11] also symmetrize and normalize the probabilities $p_{j|i}$ and $q_{j|i}$

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n} \tag{20a}$$

$$q_{ij} = \frac{q_{i|j} + q_{j|i}}{2n} \tag{20b}$$

Then the minimization is over the KL divergence with respect to the symmetrized probabilities:

$$C \equiv \mathrm{KL}(Q|P) = \sum_{i \neq j} p_{ij} \log(p_{ij}/q_{ij}) \tag{21}$$

which yields as its equations of motion

$$\frac{\partial C}{\partial \vec{Y}^{(i)}} = 4 \sum_j (p_{ij} - q_{ij})(\vec{Y}^{(i)} - \vec{Y}^{(j)}) \tag{22}$$

[1] A helpful toy which shows t-SNE in action can be found here: https://distill.pub/2016/misread-tsne/

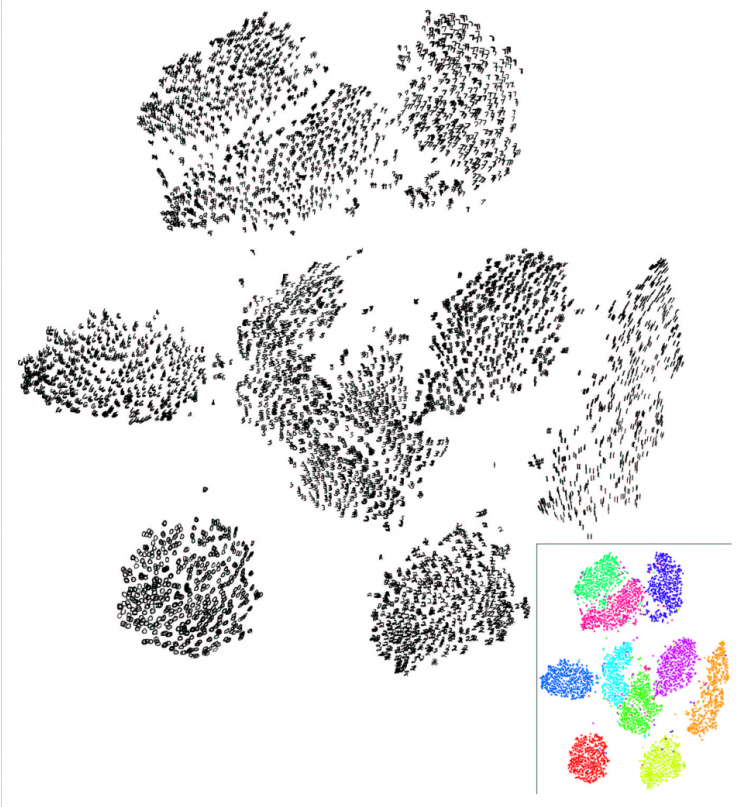

Figure 5: The output of t-SNE on 6,000 handwritten digits from the MNIST dataset, reproduced from Figure 7 of [11].

Note that t-SNE does not necessarily have a unique minimum, so Gaussian noise is typically introduced at some stages of the evolution to move the dynamical system from local minima.

The use of the fat-tailed distribution allows greater freedom to "fit" intrinsically larger-dimensional spaces into a flat map for visualization. However, as a consequence, while short distances in t-SNE are broadly trustworthy, large distances do not mean much (besides reflecting dissimilarity), either as a direct measure of distance or for comparison purposes.

4 Autoencoders

An **autoencoder** is a neural network whose goal is to return its output as its input. Because the training data do not need to have any sort of labels attached to them (which specify, e.g., classifications), autoencoders are a form of **unsupervised learning**.

At a glance, it seems like this would be a trivial task, since a neural network could very easily learn the identity map. However, we can get around the autoencoder doing something trivial by imposing restrictions on its architecture. For example, if the data are N-dimensional, we can force the data to pass through a layer which is only d-dimensional, where $d < N$ (and possibly $d \ll N$).

More precisely, we can think of an autoencoder as two neural networks in series. First, an **encoder**

takes the raw, N-dimensional data as its input and returns some d-dimensional data (which is said to reside in the **latent space**). Then, a **decoder** takes the "encoded" data and attempts to reconstruct the full N-dimensional vector. The hope is that, because the data is forced to pass through a dimensional bottleneck, the encoder and decoder will work together during training to find the most efficient way to encode the information content within the dataset in a lower-dimensional latent space. The encodings of the data in this latent space then become the lower-dimensional embedding that we desire.

It may be tricky in practice to coax the autoencoder to behave in a desirable way. As an example of how this could go wrong, we can consider an encoder which maps a data point to its index (which typically has no physical significance) and a decoder which maps it back:

$$\vec{X}^{(i)} \to i \to \vec{X}^{(i)} \tag{23}$$

In other words, formally it is possible to map any data (of arbitrarily large dimension *and* intrinsic dimension) to a one-dimensional space (of indices) and then back. This is undesirable, since we would like proximity in our lower-dimensional embedding to say something about the similarity of our data points, which is clearly not the case here. This is conceptually similar to **overfitting**: the one-dimensional embedding here basically goes out of its way to pass through every single data point in a very contrived way.

There are many ways to construct and modify autoencoders towards the end of creating a good low-dimensional embedding. One survey of these is given by [12].

4.1 Denoising autoencoders

One way to coax an autoencoder out of overfitting is to *add noise* to the input before giving it to the encoder during training. Then the cost function is constructed towards the goal of recovering the *original* (non-noisy) input. This sort of setup is called a **denoising autoencoder**.

The idea behind adding noise in the training is simple: in a proper lower-dimensional embedding, we would prefer for two points which are close together to be "similar" in some sense. However, if the encoding is overfit, this is not necessarily the case. To solve this problem, we can "perturb" the original data point (by adding noise) and then reward the autoencoder for recovering a "nearby" point. The autoencoder now cannot get away with creating a totally contrived encoding, because such encodings will not be robust to perturbing the input by a little bit. It will thus be forced to create an embedding which respects proximity in some sense.

4.2 Variational autoencoders

Variational autoencoders have a similar motivation to denoising autoencoders, namely, enforcing that the latent space encode data in a smooth way. However, rather than adding noise in the input, variational autoencoders make their probabilistic nature more explicit by having the encoder output to a mean $\vec{\mu}$ and standard deviation σ. The decoder is then given a vector $\vec{z}$ sampled from a Gaussian centered around $\vec{\mu}$ with standard deviation σ from the latent space.

The loss function (to be minimized) is spiritually similar, although a little more complicated[2]:

$$J(\theta, \phi) = -\mathbb{E}_{\vec{z}\sim q_\theta(\vec{z}|\vec{x})} \left[\log p_\phi(\vec{x}|\vec{z})\right] + \mathrm{KL}(q_\theta(\vec{z}|\vec{x})|p_\phi(\vec{x})) \tag{24}$$

The first term denotes the expectation value of the log-likelihood of the decoder $p_\phi(\vec{x}|\vec{z})$ (with weights and biases ϕ) to produce the desired output $\vec{x}$ (matching the input) given a latent space vector $\vec{z}$ drawn from the encoder's output probability distribution $q_\theta(\vec{z}|\vec{x})$ (with weights and biases θ). This is the standard autoencoder term that checks how similar the input and output are.

The second term is a regularization term which compares the output distribution of the *encoder* to a prior $p_\phi(\vec{x})$, typically a unit normal Gaussian. This term incentivizes the optimization to keep the distribution as close to a Gaussian as possible (so that deviations from a Gaussian on the latent space must be compensated by significant improvements in the first term).

The optimization during training is, of course, over the encoder and decoder weights and biases θ and ϕ, respectively.

A version of variational autoencoders adds a hyperparameter to the loss function, β, which regulates the strength of the regularization:

$$J(\theta, \phi) = -\mathbb{E}_{\vec{z}\sim q_\theta(\vec{z}|\vec{x})} \left[\log p_\phi(\vec{x}|\vec{z})\right] + \beta\, \mathrm{KL}(q_\theta(\vec{z}|\vec{x})|p_\phi(\vec{x})) \tag{25}$$

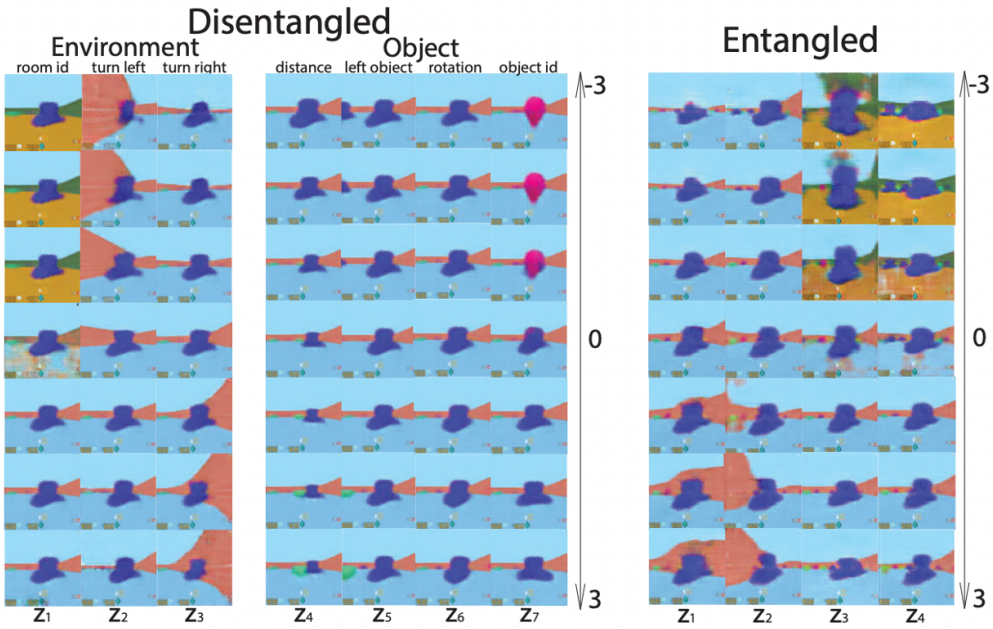

Figure 6: A comparison of the outputs of entangled and disentangled autoencoders, reproduced from Figure 3 of [13]. The directions in the latent space of the latter more clearly correspond to single, physically interpretable variables.

[2]Some helpful notes which clearly lay out the variables which appear in this complicated loss function can be found here: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

Such autoencoders are called **disentangled variational autoencoders (β-VAE)**[3]. The purpose of increasing the strength of this regularization is to coax the autoencoder into only using more directions in the latent space if it has to do so. This usually encourages the physical directions in the latent space to take physical interpretations (e.g., Figure 6), which is highly desirable in a lower-dimensional embedding.

5 Concluding remarks

The problem of dimensionality reduction, especially as it relates to machine learning, is of considerable interest in computer science and has wide ranging applications. There are thus many helpful online resources and reviews (e.g., [14]).

These Notes are by no means comprehensive, and quite a few more dimensionality reduction techniques can be found in the MATLAB toolbox of Laurens van der Maaten[4], although there are doubtless many more.

References

[1] Carine Babusiaux, Floor van Leeuwen, MA Barstow, C Jordi, Antonella Vallenari, D Bossini, Alessandro Bressan, T Cantat-Gaudin, M Van Leeuwen, AGA Brown, et al. Gaia data release 2-observational hertzsprung-russell diagrams. *Astronomy & Astrophysics*, 616:A10, 2018.

[2] S Djorgovski and G Meylan. The galactic globular cluster system. *The Astronomical Journal*, 108:1292–1311, 1994.

[3] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[4] Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Locally linear embedding and its variants: Tutorial and survey. *arXiv preprint arXiv:2011.10925*, 2020.

[5] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[6] Vin De Silva and Joshua B Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. *Advances in neural information processing systems*, pages 721–728, 2003.

[7] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[8] Kilian Weinberger, Benjamin Packer, and Lawrence Saul. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In *International Workshop on Artificial Intelligence and Statistics*, pages 381–388. PMLR, 2005.

[9] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.

[3] A good introduction to variational autoencoders (including disentangled variational autoencoders) can be found here: https://www.youtube.com/watch?v=9zKuYvjFFS8

[4] https://lvdmaaten.github.io/drtoolbox/

[10] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15, 2002.

[11] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[12] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[13] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pages 1480–1490. PMLR, 2017.

[14] Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Unified framework for spectral dimensionality reduction, maximum variance unfolding, and kernel learning by semidefinite programming: Tutorial and survey. *arXiv preprint arXiv:2106.15379*, 2021.