Imputation reveals gene-gene correlation patterns
Integrating multiple single-cell datasets
Canonical Correlation Analysis (CCA)

From: Butlet et al., 2018
Mutual nearest neighbors (MNN) correction

From: Haghverdi et al., 2018
Summary and Method comparison
Comparison of differential expression methods

Found Limma-trend, MAST, edgeR, also t-test and Wilcoxon to perform well.
Summary

- **Normalization**
  - Scran and Linnorm
- **Imputation**
  - SAVER
- **Batch-correction**
  - [fast]MNN and Harmony
- **Clustering**
  - ACTIONet and Seurat
- **Trajectory detection**
  - Monocle3 and Slingshot
- **Differential expression**
  - Limma-trend
Deep Learning methods for scRNA-seq
Autoencoder ResNet arch. | Train ResNet with loss MMD score function
---|---
MMD (Gretton et al., 2006, 2012) is a measure for distance between two probability distributions $p, q.$ It is defined with respect to a function class $\mathcal{F}$ by

\[
\text{MMD}(\mathcal{F}, p, q) \equiv \sup_{f \in \mathcal{F}} \left( \mathbb{E}_{x \sim p}(f(x)) - \mathbb{E}_{y \sim q}(f(y)) \right).
\]

When $\mathcal{F}$ is a reproducing kernel Hilbert space with kernel $k,$ the MMD can be written as the distance between the mean embeddings of $p$ and $q$

\[
\text{MMD}^2(\mathcal{F}, p, q) = \| \mu_p - \mu_q \|^2,
\]

where $\mu_p(t) = \mathbb{E}_{x \sim p} k(x, t).$ Equation (1) can be written as

\[
\text{MMD}^2(\mathcal{F}, p, q) = \mathbb{E}_{x \sim p, x' \sim p} k(x, x') - 2 \mathbb{E}_{x \sim p, y \sim q} k(x, y) + \mathbb{E}_{y \sim q, y' \sim q} k(y, y').
\]

where $x$ and $x'$ are independent, and so are $y$ and $y'.$ Importantly, if $k$ is a universal kernel, then $\text{MMD}(\mathcal{F}, p, q) = 0$ iff $p = q.$ In practice, the distributions $p, q$ are unknown, and instead we are given observations $X = \{x_1, \ldots, x_n\}, Y = \{y_1, \ldots, y_m\},$ so that the (biased) sample version of (2) becomes

\[
\text{MMD}^2(\mathcal{F}, X, Y) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n k(x_i, x_j) - \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m k(x_i, y_j) + \frac{1}{m^2} \sum_{j=1}^m \sum_{j'=1}^m k(y_j, y_{j'}).
\]

MMD-ResNet P-value distribution shifted to high KS P-values (more similar distributions)

Fig. 5. Histograms of the 25 P-values of Kolmogorov-Smirnov tests, comparing the distributions of the calibrated data with the target distribution of each of the 25 markers

MMD-ResNet outperforms PCA, Combat, and

Removal of batch effects using distribution-matching residual networks

Uri Shaham$^{1,\dagger},$ Kelly P. Stanton$^{2,3,\dagger},$ Jun Zhao$^3,$ Huamin Li$^4,$ Khadir Raddassi$^5,$ Ruth Montgomery$^6$ and Yuval Kluger$^{2,3,4,*}$

Shaham et al., Bioinf, 2017

MMD-ResNet, Autoencoder for batch correction

MMD-ResNet

Prkca-high cells

Combat

MMD-ResNet

t-SNE plots before (left) and after (right) calibration
DESC: Deep embedding for cell-type-specific batch correction

DESC (Deep Embedding for single-cell clustering):
- Stacked auto-encoder learns cluster-specific gene expression representation and cluster assignments for scRNA-seq data clustering
- Initialize clustering obtained from autoencoder
- Learn non-linear mapping from original space to a low-dimensional space
- iteratively optimize clustering objective function
  - Move each cell to nearest cluster
  - balance biological and technical differences between clusters
  - reduce influence of batch effect
- Enables soft clustering by assigning cluster-specific probabilities to each cell
- Facilitates clustering of cells with high confidence

DESC avoids cluster-specific batch effects found in other methods

Iterative approach progressively removes cluster-specific batch effects

Rand Index (RI) = measure of the similarity between two data clusterings
ARI = Adjusted Rand Index, adjusted for the chance grouping of elements
**AutoImpute: Overcomplete autoencoder for filling in zeros**

AutoImpute filters raw gene expression data for bad genes (normalize by library size, prune by gene-selection, log transform). Feed processed matrix to AutoImpute model:
- Learn expression data representation
- Reconstruct imputed matrix

Use overcomplete autoencoders to capture distribution of sparse gene expression data, and regenerate complete version of it:
- Feeding sparse gene expression matrix as input to autoencoder
- Train it to learn the encoder and decoder functions that best regenerate imputed expression with no dropouts
- Back-propagating errors only for non-zero counts in sparse matrix.

**AutoImpute captures more non-zero values for highly-expressed genes**

**Training and Hyper-parameter Selection.** The autoencoder network consists of a fully-connected multi-layer perceptron (MLP), with three layers: input, hidden and the output layer. It is trained using gradient descent with gradients computed by backpropagation to reach the minimum of the cost function (equation 8). RAISE Prop Optimizer was used to adjust the learning rate, such that, we avoid getting stuck at local minima and reach the minimum of the cost function faster. Both $E$ - encoder matrix and $D$ - decoder matrix were initialized from a random normal distribution.

The hyper-parameter selection was done after doing an extensive grid search on the following hyper-parameters:

- $\lambda$ - the regularization coefficient, to control the contribution of the regularization term in the loss or cost function.
- Size of the hidden layer or latent space dimensionality.
- Initial value of learning rate.
- Threshold value - We stop the gradient descent after the change in loss function value in consecutive iterations is less than the threshold value, implying convergence.

The best results were observed on the hyper-parameter choices shown in Table 2.
**scVI:** Use NN to estimate params in variational inference

**scVI:** Learn non-linear embedding of cells for multiple analysis tasks
NN=Neural networks used to compute embedding and expr. distribution $f_w, f_h$: functional representations NN5,6 to capture parameters of Gaussians

Modeled observed expression $x_{ng}$ (gene g, cell n) as sample
Drawn from zero-inflated negative binomial (ZINB) distribution
Conditioned on the batch annotation $s_n$ of each cell (if available)
And on two additional, unobserved random variables:
- $\rho_{gn}$ **nuisance variation**, 1-D Gaussian, model differences in capture efficiency & sequencing depth, cell-specific scaling factor
- $z_n$, **remaining variation**, 10-D Gaussian, model biological differences between cells.

Represent each cell as point in low-dimensional latent space (for visualization and clustering).
Neural network maps the latent variables to ZINB distribution parameters (Fig. 1a, neural networks 5 and 6).
This mapping goes through intermediate variables:
- batch-corrected, normalized estimate of the percentage of transcripts in each cell n that originate from each gene g
Use these estimates for differential expression analysis
Use scaled version (multiplying by estimated library size) for imputation.
Derived approximation for posterior distribution of latent variables $q$ by training another neural network using variational inference and a scalable stochastic optimization procedure (NN1-NN4).
Non-Linear Dimensionality Reduction
Dimensionality reduction: Some Assumptions

- High-dimensional data often lies on or near a much lower dimensional, curved manifold.
- A good way to represent data points is by their low-dimensional coordinates.
- The low-dimensional representation of the data should capture information about high-dimensional pairwise distances.
The basic idea of non-parameteric dimensionality reduction

- Represent each data-point by a point in a lower dimensional space.
- Choose the low-dimensional points so that they optimally represent some property of the data-points (e.g. the pairwise distances).
  - Many different properties have been tried.
- Do not insist on learning a parametric “encoding” function that maps each individual data-point to its low-dimensional representative.
- Do not insist on learning a parametric “decoding” function that reconstructs a data-point from its low dimensional representative.
Two types of dimensionality reduction

- **Global methods** assume that all pairwise distances are of equal importance.
  - Choose the low-D pairwise distances to fit the high-D ones (using magnitude or rank order).

- **Local methods** assume that only the local distances are reliable in high-D.
  - Put more weight on modeling the local distances correctly.
Linear methods of reducing dimensionality

- **PCA** finds the directions that have the most variance.
  - By representing where each datapoint is along these axes, we minimize the squared reconstruction error.
  - Linear autoencoders are equivalent to PCA.

- **Multi-Dimensional Scaling** arranges the low-dimensional points so as to minimize the discrepancy between the pairwise distances in the original space and the pairwise distances in the low-D space.
Metric Multi-Dimensional Scaling

• Find low dimensional representatives, \( y \), for the high-dimensional data-points, \( x \), that preserve pairwise distances as well as possible.

• An obvious approach is to start with random vectors for the \( y \)'s and then perform steepest descent by following the gradient of the cost function.

• Since we are minimizing squared errors, maybe this has something to do with PCA?
  
  – If so, we don’t need an iterative method to find the best embedding.

\[
\text{Cost} = \sum_{i<j} (d_{ij} - \hat{d}_{ij})^2
\]

\[
d_{ij} = \| x_i - x_j \|^2
\]

\[
\hat{d}_{ij} = \| y_i - y_j \|^2
\]
Converting metric MDS to PCA

• If the data-points all lie on a hyperplane, their pairwise distances are perfectly preserved by projecting the high-dimensional coordinates onto the hyperplane.
  – So in that particular case, PCA is the right solution.

• If we “double-center” the data, metric MDS is equivalent to PCA.
  – Double centering means making the mean value of every row and column be zero.
  – But double centering can introduce spurious structure.
Other non-linear methods of reducing dimensionality

- Non-linear autoencoders with extra layers are much more powerful than PCA but they can be slow to optimize and they get different, locally optimal solutions each time.

- **Multi-Dimensional Scaling** can be made non-linear by putting more importance on the small distances. A popular version is the Sammon mapping:

  \[
  \text{Cost} = \sum_{i \neq j} \left( \frac{\| x_i - x_j \| - \| y_i - y_j \|}{\| x_i - x_j \|} \right)^2
  \]

- Non-linear MDS is also slow to optimize and also gets stuck in different local optima each time.
Problems with Sammon mapping

• It puts too much emphasis on getting very small distances exactly right.
• It produces embeddings that are circular with roughly uniform density of the map points.
IsoMap: Local MDS without local optima

• Instead of only modeling local distances, we can try to measure the distances along the manifold and then model these intrinsic distances.
  – The main problem is to find a robust way of measuring distances along the manifold.
  – If we can measure manifold distances, the global optimisation is easy: It’s just global MDS (i.e. PCA)

If we measure distances along the manifold, \(d(1,6) > d(1,4)\)
How Isomap measures intrinsic distances

• Connect each datapoint to its K nearest neighbors in the high-dimensional space.
• Put the true Euclidean distance on each of these links.
• Then approximate the manifold distance between any pair of points as the shortest path in this “neighborhood graph”.
Using Isomap to discover the intrinsic manifold in a set of face images
Linear methods cannot interpolate properly between the leftmost and rightmost images in each row.

This is because the interpolated images are NOT averages of the images at the two ends.

Isomap does not interpolate properly either because it can only use examples from the training set. It cannot create new images.

But it is better than linear methods.
Maps that preserve local geometry

• The idea is to make the local configurations of points in the low-dimensional space resemble the local configurations in the high-dimensional space.
• We need a coordinate-free way of representing a local configuration.
• If we represent a point as a weighted average of nearby points, the weights describe the local configuration.

\[ x_i \approx \sum_j w_{ij} x_j \]
Finding the optimal weights

- This is easy.
- Minimize the squared “construction” errors subject to the sum of the weights being 1.

\[
Cost = \sum_{i} \left\| x_i - \sum_{j \in N(i)} w_{ij} x_j \right\|^2, \quad \sum_{j \in N(i)} w_{ij} = 1
\]

- If the construction is done using less neighbors than the dimensionality of \( x \), there will generally be some construction error
  - The error will be small if there are as many neighbors as the dimensionality of the underlying noisy manifold.
A sensible but inefficient way to use the local weights

- Assume a low-dimensional latent space.
  - Each datapoint $x_i$ has latent coordinates $y_i$.

- Find a set of latent points that minimize the construction errors produced by a two-stage process:
  1. First use the latent points to compute the local weights that construct $y_i$ from its neighbors.
  2. Use those weights to construct the high-dimensional coordinates of a datapoint $x_i$ from the high-dimensional coordinates of its neighbors.

- Unfortunately, this is a hard optimization problem.
  - Iterative solutions are expensive because they must repeatedly measure the construction error in the high-dimensional space.
Local Linear Embedding: A less sensible but more efficient way to use local weights

• Instead of using the latent points plus the other datapoints to construct each held-out datapoint, do it the other way around.
• Use the datapoints to determine the local weights, then try to construct each latent point from its neighbors.
  – Now the construction error is in the low-dimensional latent space.
• We only use the high-dimensional space once to get the local weights.
  – The local weights stay fixed during the optimization of the latent coordinates.
  – This is a much easier search.
The convex optimization

\[ \text{Cost} = \sum_{i} \| y_i - \sum_{j \in N(i)} w_{ij} y_j \|^2 \]

- Find the \( y \)'s that minimize the cost subject to the constraint that the \( y \)'s have unit variance on each dimension.
  - Why do we need to impose a constraint on the variance?
The collapse problem

• If all of the latent points are identical, we can construct each of them perfectly as a weighted average of its neighbors.
  – The root cause of this problem is that we are optimizing the wrong thing.
  – But maybe we can fix things up by adding a constraint that prevents collapse.

• Insist that the latent points have unit variance on each latent dimension.
  – This helps a lot, but sometimes LLE can satisfy this constraint without doing what we really intend.
Failure modes of LLE

- If the neighborhood graph has several disconnected pieces, we can satisfy the unit variance constraint and still have collapses.

- Even if the graph is fully connected, it may be possible to collapse all the densely connected regions and satisfy the variance constraint by paying a high cost for a few outliers.
A typical embedding found by LLE

• LLE embeddings often look like this.
• Most of the data is close to the center of the space.
• A few points are far from the center to satisfy the unit variance constraint.
A comment on LLE

• It has two very attractive features
  – 1. The only free parameters are the dimensionality of the latent space and the number of neighbors that are used to determine the local weights.
  – 2. The optimization is convex so we don’t need multiple tries and we don’t need to fiddle with optimization parameters.

• It has one bad feature:
  – It is not optimizing the right thing!
  – One consequence is that it does not have any incentive to keep widely separated datapoints far apart in the low-dimensional map.
Maximum Variance Unfolding

- This fixes one of the problems of LLE and still manages to be a convex optimization problem.
- Use a few neighbors for each datapoint and insist that the high-dimensional distances between neighbors are exactly preserved in the low-dimensional space.
  - This is like connecting the points with rods of fixed lengths.
- Subject to the rigid rods connecting the low-dimensional points, maximize their squared separations.
  - This encourages widely separated datapoints to remain separated in the low-dimensional space.
How to solve many problems in AI

• 1. Map from the data domain to a domain of feature vectors in which the important relationships can be modeled by linear operations.

• 2. Do some linear operations.

• 3. Map the answer back to the data domain.
Modeling relational data

- Suppose we have a set of facts of the form ARB
  - i.e. The relation R maps A to B as in Allan has-mother Beatrice
- We could model the facts using matrix algebra.
  - Learn a vector for each object
  - Learn a matrix for each relation
  - The aim is to make A*R=B
- This doesn't work because all the vectors learn to be zero.
  - We need A*R to be closer to B than to C.
A discriminative cost function

\[ p_{B|AR} = \frac{e^{\|AR-B\|^2}}{\sum_{C} e^{\|AR-C\|^2}} \]
Applying the idea to dimensionality reduction

• 0. Compute a big probability table that contains the probability that each high-dimensional data-point, \( i \), would pick another data-point, \( j \), as its “neighbor”.

• 1. Use a learned look-up table to convert each high-dimensional data-point to a 2-D feature vector.

• 2. Multiply by the identity matrix.

• 3. Compare the resulting 2-D feature vector with all the other 2-D feature vectors to get a predicted distribution over data-points in the original data-space.

• Learn the look-up tables so that the probabilities computed in the 2-D space match the probabilities computed in the original space.
A probabilistic version of local MDS

• It is more important to get local distances right than non-local ones, but getting infinitesimal distances right is not infinitely important.
  – All the small distances are about equally important to model correctly.
  – Stochastic neighbor embedding has a probabilistic way of deciding if a pairwise distance is “local”.
Stochastic Neighbor Embedding

• First convert each high-dimensional similarity into the probability that one data point will pick the other data point as its neighbor.

• To evaluate a map:
  – Use the pairwise distances in the low-dimensional map to define the probability that a map point will pick another map point as its neighbor.
  – Compute the Kullback-Leibler divergence between the probabilities in the high-dimensional and low-dimensional spaces.
A probabilistic local method

- Each point in high-D has a conditional probability of picking each other point as its neighbor.

- The distribution over neighbors is based on the high-D pairwise distances.
  - If we do not have coordinates for the datapoints we can use a matrix of dissimilarities instead of pairwise distances.

\[
p_{j|i} = \frac{e^{-\frac{d_{ij}^2}{2\sigma_i^2}}}{\sum_k e^{-\frac{d_{ik}^2}{2\sigma_i^2}}}
\]

probability of picking j given that you start at i
Throwing away the raw data

- The probabilities that each points picks other points as its neighbor contains all of the information we are going to use for finding the manifold.
  - Once we have the probabilities $P_{j|i}$ we do not need to do any more computations in the high-dimensional space.
  - The input could be “dissimilarities” between pairs of datapoints instead of the locations of individual datapoints in a high-dimensional space.
Evaluating an arrangement of the data in a low-dimensional space

• Give each datapoint a location in the low-dimensional space.
  – Evaluate this representation by seeing how well the low-D probabilities model the high-D ones.

\[ q_{j|i} = \frac{e^{-d_{ij}^2}}{\sum_k e^{-d_{ik}^2}} \]

probability of picking \( j \) given that you start at \( i \)
The cost function for a low-dimensional representation

\[
Cost = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}
\]

- For points where \( p_{ij} \) is large and \( q_{ij} \) is small we lose a lot.
  - Nearby points in high-D really want to be nearby in low-D
- For points where \( q_{ij} \) is large and \( p_{ij} \) is small we lose a little because we waste some of the probability mass in the \( Q_i \) distribution.
  - Widely separated points in high-D have a mild preference for being widely separated in low-D.
The forces acting on the low-dimensional points

\[ \frac{\partial \text{Cost}}{\partial y_i} = 2 \sum_j (y_j - y_i) (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}) \]

- Points are pulled towards each other if the p’s are bigger than the q’s and repelled if the q’s are bigger than the p’s
Unsupervised SNE embedding of the digits 0-4. Not all the data is displayed.
Picking the radius of the gaussian that is used to compute the p’s

• We need to use different radii in different parts of the space so that we keep the effective number of neighbors about constant.
• A big radius leads to a high entropy for the distribution over neighbors of i.
• A small radius leads to a low entropy.
• So decide what entropy you want and then find the radius that produces that entropy.
• Its easier to specify $2^{\text{entropy}}$
  – This is called the perplexity
  – It is the effective number of neighbors.
Symmetric SNE

• There is a simpler version of SNE which seems to work about equally well.
• Symmetric SNE works best if we use different procedures for computing the $p$’s and the $q$’s
  – This destroys the nice property that if we embed in a space of the same dimension as the data, the data itself is the optimal solution.
Computing the p’ s for symmetric SNE

- Each high dimensional point, i, has a **conditional** probability of picking each other point, j, as its neighbor.

- The conditional distribution over neighbors is based on the high-dimensional pairwise distances.

\[
p_{j|i} = \frac{e^{-d_{ij}^2/2\sigma_i^2}}{\sum_k e^{-d_{ik}^2/2\sigma_i^2}}
\]

probability of picking j given that you start at i
Turning conditional probabilities into pairwise probabilities

To get a symmetric probability between \( i \) and \( j \) we sum the two conditional probabilities and divide by the number of points (points are not allowed to choose themselves).

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

This ensures that all the pairwise probabilities sum to 1 so they can be treated as probabilities.

\[
\sum_{i,j} p_{ij} = 1
\]
Evaluating an arrangement of the points in the low-dimensional space

- Give each data-point a location in the low-dimensional space.
  - Define low-dimensional probabilities symmetrically.
  - Evaluate the representation by seeing how well the low-D probabilities model the high-D affinities.

\[
q_{ij} = \frac{e^{-d_{ij}^2}}{\sum_{k<l} e^{-d_{kl}^2}}
\]
The cost function for a low-dimensional representation

\[ \text{Cost} = KL(P \parallel Q) = \sum_{i<j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \]

- It’s a single KL instead of the sum of one KL for each datapoint.
The forces acting on the low-dimensional points

\[
\frac{\partial KL(P \parallel Q)}{\partial y_i} = 2 \sum_j (y_i - y_j) (p_{ij} - q_{ij})
\]

- Points are pulled towards each other if the \( p \)'s are bigger than the \( q \)'s and repelled if the \( q \)'s are bigger than the \( p \)'s
  - It's equivalent to having springs whose stiffnesses are set dynamically.
SNE applied to 30-dimensional PCA codes of 5000 MNIST digits
Optimization methods for SNE

• We get much better global organization if we use annealing.
  – Add Gaussian noise to the y locations after each update.
  – Reduce the amount of noise on each iteration.
  – Spend a long time at the noise level at which the global structure starts to form from the hot plasma of map points.
• It also helps to use momentum (especially at the end).
• It helps to use an adaptive global step-size.
More optimization tricks for SNE

• Anneal the perplexity.
  – This is expensive because it involves computing distances in the high-dimensional data-space.

• Dimension decay
  – Use additional dimensions to avoid local optima, then penalize the squared magnitudes of the map points on the extra dimensions.
    • Turn up the penalty coefficient until all of the map points have very small values on those extra dimensions.

• Neither of these tricks is a big win in general.
A more interesting variation that uses the probabilistic foundation of SNE

- All other dimensionality reduction methods assume that each data point is represented by ONE point in the map.
- But suppose we had several different maps.
  - Each map has a representative of each datapoint and the representative has a mixing proportion.
  - The overall \( q_{ij} \) is a sum over all maps

\[
\begin{align*}
q_{ij} &= \sum_m q_{ij}^m \\
q_{ij}^m &= \pi_i^m \pi_j^m \exp(-d_{ij}^m)
\end{align*}
\]
A nice dataset for testing “Aspect maps”

• Give someone a word and ask them to say the first other word they associate with it.
  – Different senses of a word will have different associations and so they should show up in different aspect maps.
Two of the 50 aspect maps for the Florida word association data
The relationship between aspect maps and clustering

• If we force all of the locations in each map to be the same, it’s a form of spectral clustering!

• Putting a point into a map is not just based on the affinities it has to the other points in the map.
  – It depends on whether it can find a location in the map that allows it to match the pattern of affinities.
  – It has a very abstract resemblance to mixtures of experts vs ordinary mixtures.
A weird behaviour of aspect maps

- If we use just 2 aspect maps, one of them collapses all of the map points to the same location.
  - It’s trying to tell us something!
- It wants to use a uniform background probability for all pairs

\[ q_{ij} = \pi_1 q_{ij}^1 + \frac{1 - \pi_1}{N^2} \]
Why SNE does not have gaps between classes

• In the high-dimensional space there are many pairs of points that are moderately close to each other.
  – The low-D space cannot model this. It doesn’t have enough room around the edges.
• So there are many $p_{ij}$’s that are modeled by smaller $q_{ij}$’s.
  – This has the effect of lots of weak springs pulling everything together and crushing different classes together in the middle of the space.
• A uniform background model eliminates this effect and allows gaps between classes to appear.
  – It is quite like Maximum Variance Unfolding
SNE then UNI-SNE applied to 30-D PCA codes of 5000 MNIST digits
From UNI-SNE to t-SNE

• Laurens van der Maaten started experimenting on UNI-SNE and we soon realised that it was easier to replace the mixture of a Gaussian and a uniform by an infinite mixture of Gaussians:

\[ q_{ij} \propto \frac{1}{1 + d_{ij}^2} \]

• By using a Gaussian to compute \( p_{ij} \) and a heavy-tailed student’s t to compute \( q_{ij} \) we partially compensate for the different rates of growth of volume as you move away from a point in 2-D and in N-D.
t-SNE

• Instead of using a gaussian plus a uniform, why not use gaussians at many different spatial scales?
  – This sounds expensive, but if we use an infinite number of gaussians, it's actually cheaper because we avoid exponentiating.

\[ q_{ij} \propto \frac{1}{1 + d_{ij}^2} \]
Optimization hacks

• **Reputable hack:** Introduce a penalty term that keeps all the map-points close together.
  – Then gradually relax the penalty to break symmetry slowly.

• **Disreputable hack:** Allow the probabilities to add up to 4.
  – This causes the map-points to curdle into small clusters leaving lots of space for clusters to move past each other.
  – Then make the probabilities add up to 1.
Two other state-of-the-art dimensionality reduction methods on the 6000 MNIST digits

Isomap

Locally Linear Embedding
t-SNE on the 6000 MNIST digits
The COIL20 dataset

Each object is rotated about a vertical axis to produce a closed one-dimensional manifold of images.
Isomap & LLE for COIL20 dataset

Isomap

Locally Linear Embedding
t-SNE for COIL20 dataset
Show the map of 2000 English words produced by Joseph Turian using t-SNE on the feature vectors learned by Colobert and Weston (ICML 2008)
Using t-SNE to see what you are thinking
Using t-SNE to see how a DBN that recognizes phonemes deals with speaker variation

• Current speech recognizers try to preprocess the input to eliminate differences between speakers.
• We get very good performance from a DBN without doing this.
  – It's very difficult to improve the DBN by adding speaker information.
• Maybe the DBN is using its hidden layers to get rid of speaker variation.
  – This would explain why speaker information is not much help.
t-SNE applied to windows of 15 input frames for 6 speakers saying two sentences
t-SNE applied to the first hidden layer
t-SNE applied to the 8\textsuperscript{th} hidden layer
Some recent developments

• Miguel Carreira-Perpinan (ICML 2010) showed that the original SNE cost function can be rewritten so that it is equivalent to Laplacian Eigenmaps with an extra repulsion term that spreads out the map points (like in MVU).

• This led to a much faster optimization method. The fast code is now on the t-SNE webpage.
Combining non-parametric dimensionality reduction with neural networks

• If we have a smooth objective function that assigns a value to a set of codes, we can backpropagate its derivatives through a feedforward neural net.
  – The neural net is like the “encoder” part of an autoencoder.

• We could combine this objective function with the objective of getting good reconstructions from the codes.
Evaluating the codes found by an autoencoder

- Use 3000 images of handwritten digits from the USPS training set.
  - Each image is 16x16 and fairly binary.
- Use a highly non-linear autoencoder
  - Use logistic output units and linear code units.
Does code space capture the structure of the data?

- We would like the code space to model the underlying structure of the data.
  - Digits in the same class should get closer together in code space.
  - Digits of different classes should get further apart.

- We can use k nearest neighbors to see if this happens.
  - Hold out each image in turn and try to label it by using the labels of its k nearest neighbors.

- In pixel space we get 5.9% errors. In code space we get about 12% errors. Why doesn’t it work?
How it goes wrong

• PCA is not powerful enough to really mangle the data.

• Non-linear auto-encoders can fracture a manifold into many different domains.
  – This can lead to very different codes for nearby data-points.
How to fix it

• We need a regularizer that will make it costly to fracture the manifold.
  – There are many possible regularizers.

• Stochastic neighbor embedding can be used as a regularizer.
  – It's like putting springs between the codes to prevent the codes for similar datapoints from being too far apart.
How the gradients are combined

Forces generated by springs attaching this code to the codes for all the other data-points. The stiffness of each spring is dynamically set to be: $p_{ij} - q_{ij}$
How well does it work?

• Instead of rising from 5.9% to 12%, the hold-one-out KNN error falls to about 2%
  – The strength of the regularizer must be chosen sensibly.
  – The SNE regularizer alone gives about 4.5% hold-one-out KNN errors.

• Can we visualize the codes that are produced using the regularizer?
Learning codes with some pairwise information about labels

• If we pair each digit with another of the same class, there are very nice category boundaries between digits.
A more efficient version

• The derivatives that come from the autoencoder will stop the codes from all collapsing to a point.
  – So we don’t need the quadratically expensive normalization term that is used in computing the qij’s
• We should be able to just add attractive forces between codes that correspond to similar inputs.
  – The similarity could be measured in the input space or it could be based on extra information (e.g. identity of class labels).
Non-linear Neighborhood Components Analysis (Salakhutdinov and Hinton, 2007)

• Use a feed forward neural net to learn codes that make nearest neighbor classification work well.
• In code space, we can predict the class of a point by summing the probabilities assigned to other points of that class when we use the stochastic neighbor probabilities.

\[ p(\text{class}(i) = c) = \sum_{j \in c} p_{j|i} \]

• So we do gradient ascent in the log probabilities of getting the correct class for each point when it is held-out
What NNCA achieves

- Linear Discriminant Analysis tries to find a projection of the data that makes each point be close to other points of the same class and far from other points of different classes.
  - This is a bad objective function if the classes form long curved manifolds
NCA

- NCA is the linear version of NNCA. It aims to find projections in which each datapoint is close to some other datapoints of the same class and not too close to datapoints of other classes.
  - This does not force all points of the same class to be similar.
  - But it can allow manifolds to become disconnected.