Visualization

Brooks Paige

Week 8

Based on slides from Dmitry Adamskiy and David Barber

What is Data Visualization?

- Data is often very high dimensional meaning that we can't directly "see" the data — in turn makes it difficult to get intuitions about the data
- For data visualization, we try to find a low dimensional representation (2 or 3 dimensions) so that we "see something".
- There is no "correct" or "perfect" visualization. Every low dimensional representation will lose some information contained in the original high dimensional data.
- Intuitively, we would hope these visualizations would allow us to see potential clusters of datapoints, or find datapoints which are "similar" to each other
- PCA was used for this historically, but now heuristic representations that better preserve neighborhood structure are generally preferred
- Deep learning (autoencoders) can work, but also are difficult to train

Visualization problem formulation

Each data vector $\mathbf{x}_n$ is in a high dimensional space. Given the set of datapoints

$$\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$$

we want to find a corresponding low dimensional (2 or 3) vector representation $\mathbf{y}_n$ for each $\mathbf{x}_n$ to give

$$\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$$

- We would like $\mathcal{Y}$ to preserve both the local and global structure in $\mathcal{X}$.
- Unfortunately, many "classical" visualization methods (Sammon mapping, Isomap, Locally Linear Embedding) don't work that well on real-world data.
- We will focus on **Stochastic Neighbor Embedding** (SNE) and its "robust" variant t-SNE, one of the most popular current approaches.

Stochastic Neighbor Embedding (SNE)

We define an $N \times N$ Markov transition matrix p which describes the local neighborhood structure — intuitively, how easily could we "jump" from one data point to another, with

$$p_{j|i} = \frac{\exp\left(-\left(\mathbf{x}_i - \mathbf{x}_j\right)^2 / (2\sigma_i^2)\right)}{\sum_{j \neq i} \exp\left(-\left(\mathbf{x}_i - \mathbf{x}_j\right)^2 / (2\sigma_i^2)\right)}, \qquad p_{i|i} = 0.$$

Stochastic Neighbor Embedding (SNE)

We define an $N \times N$ Markov transition matrix p which describes the local neighborhood structure — intuitively, how easily could we "jump" from one data point to another, with

$$p_{j|i} = \frac{\exp\left(-\left(\mathbf{x}_i - \mathbf{x}_j\right)^2 / (2\sigma_i^2)\right)}{\sum_{j \neq i} \exp\left(-\left(\mathbf{x}_i - \mathbf{x}_j\right)^2 / (2\sigma_i^2)\right)}, \qquad p_{i|i} = 0.$$

We then define the same transition matrix in our low-dimensional space as

$$q_{j|i} = \frac{\exp\left(-\left(\mathbf{y}_i - \mathbf{y}_j\right)^2\right)}{\sum_{j \neq i} \exp\left(-\left(\mathbf{y}_i - \mathbf{y}_j\right)^2\right)}, \qquad q_{i|i} = 0.$$

Stochastic Neighbor Embedding (SNE)

We define an $N \times N$ Markov transition matrix p which describes the local neighborhood structure — intuitively, how easily could we "jump" from one data point to another, with

$$p_{j|i} = \frac{\exp\left(-\left(\mathbf{x}_i - \mathbf{x}_j\right)^2 / (2\sigma_i^2)\right)}{\sum_{j \neq i} \exp\left(-\left(\mathbf{x}_i - \mathbf{x}_j\right)^2 / (2\sigma_i^2)\right)}, \qquad p_{i|i} = 0.$$

We then define the same transition matrix in our low-dimensional space as

$$q_{j|i} = \frac{\exp\left(-\left(\mathbf{y}_i - \mathbf{y}_j\right)^2\right)}{\sum_{j \neq i} \exp\left(-\left(\mathbf{y}_i - \mathbf{y}_j\right)^2\right)}, \qquad q_{i|i} = 0.$$

If we want to preserve this structure, we need to find $\mathcal{Y}$ such that q is approximately the same as p.

Stochastic Neighbor Embedding (SNE)

Stochastic Neighbor Embedding minimizes the KL divergence between each conditional distribution $p_i \equiv p_{\cdot|i}$, i.e. with a loss

$$L(\mathcal{Y}) = \sum_i D_{KL}(p_i \| q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}.$$

Stochastic Neighbor Embedding (SNE)

Stochastic Neighbor Embedding minimizes the KL divergence between each conditional distribution $p_i \equiv p_{\cdot|i}$, i.e. with a loss

$$L(\mathcal{Y}) = \sum_i D_{KL}(p_i \| q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}.$$

Note that the parameters are the actual *locations* of the points $\mathcal{Y}$.

They can be optimized by gradient descent, following

$$\frac{\partial L}{\partial \mathbf{y}_i} = 2 \sum_j \left(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}\right) \left(\mathbf{y}_i - \mathbf{y}_j\right).$$

Stochastic Neighbor Embedding (SNE)

Dangers:

- KL divergence is not symmetric — there is a large cost for using widely separated y points (small $q_{j|i}$) to represent nearby x points (large $p_{j|i}$).
- SNE therefore focuses on making sure that the local structure is correct, but loses fidelity in retaining the global structure.
- Another problem is that the Gaussian form of $p_{j|i}$ means that points which are far away will have negligible impact on the objective.

Stochastic Neighbor Embedding (SNE)

Dangers:

- KL divergence is not symmetric — there is a large cost for using widely separated y points (small $q_{j|i}$) to represent nearby x points (large $p_{j|i}$).
- SNE therefore focuses on making sure that the local structure is correct, but loses fidelity in retaining the global structure.
- Another problem is that the Gaussian form of $p_{j|i}$ means that points which are far away will have negligible impact on the objective.

Also, note that there is no way to embed *new* datapoints, or to project back from the embedded space into the data space — this is purely a means for visualization.

Symmetric SNE

There are a few steps that can be taken to make SNE more robust. The first one is to use a **symmetric** loss, by defining a symmetric transition

$$p_{i,j} = \frac{p_{j|i} + p_{i|j}}{2N}, \qquad p_{i,i} = 0$$

and using this in the objective

$$L^{\mathrm{sym}}(\mathcal{Y}) = D_{KL}(p \| q) = \sum_i \sum_j p_{i,j} \log \frac{p_{i,j}}{q_{i,j}}.$$

Symmetric SNE

There are a few steps that can be taken to make SNE more robust. The first one is to use a **symmetric** loss, by defining a symmetric transition

$$p_{i,j} = \frac{p_{j|i} + p_{i|j}}{2N}, \qquad p_{i,i} = 0$$

and using this in the objective

$$L^{\text{sym}}(\mathcal{Y}) = D_{KL}(p\|q) = \sum_i \sum_j p_{i,j} \log \frac{p_{i,j}}{q_{i,j}}.$$

This definition ensures that $p_i = \sum_j p_{i,j} > 1/(2N)$ which encourages each datapoint to have a significant effect on the cost function. This has a gradient

$$\frac{\partial L^{\text{sym}}}{\partial \mathbf{y}_i} = 4 \sum_j (p_{i,j} - q_{i,j}) \left(\mathbf{y}_i - \mathbf{y}_j\right)$$

t-SNE

A Student t distribution has heavier tails than a Gaussian and can therefore assign non-negligible mass to y points that are quite far apart.

t-SNE uses a Student t-distribution with a single degree of freedom for q, with

$$q_{i,j} = \frac{\left(1 + \left(\mathbf{y}_i - \mathbf{y}_j\right)^2\right)^{-1}}{\sum_{i \neq j} \left(1 + \left(\mathbf{y}_i - \mathbf{y}_j\right)^2\right)^{-1}}, \qquad q_{i,i} = 0$$

t-SNE

A Student t distribution has heavier tails than a Gaussian and can therefore assign non-negligible mass to y points that are quite far apart.

t-SNE uses a Student t-distribution with a single degree of freedom for q, with

$$q_{i,j} = \frac{\left(1 + (\mathbf{y}_i - \mathbf{y}_j)^2\right)^{-1}}{\sum_{i \neq j}\left(1 + (\mathbf{y}_i - \mathbf{y}_j)^2\right)^{-1}}, \qquad q_{i,i} = 0$$

- When $(\mathbf{y}_i - \mathbf{y}_j)^2$ is large, the '1' term is negligible and the q distribution will be essentially invariant with respect to the overall length scale.
- Hence, for all but the finest length scales, pairs of points that are very far apart will have a similar contribution as points that are "reasonably" far apart.

t-SNE

A Student t distribution has heavier tails than a Gaussian and can therefore assign non-negligible mass to y points that are quite far apart.
t-SNE uses a Student t-distribution with a single degree of freedom for q, with

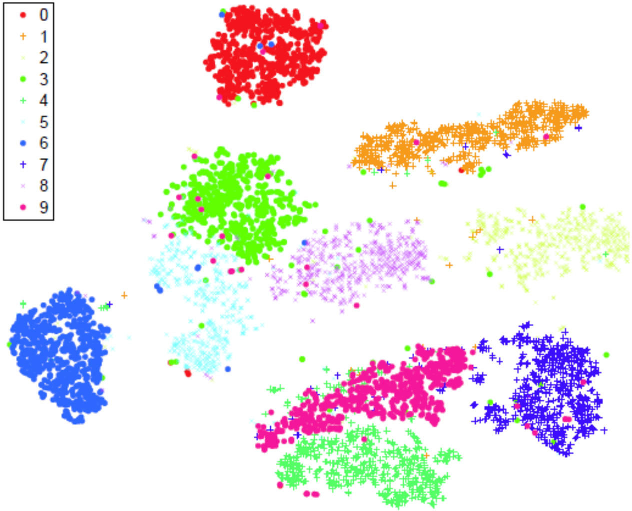
$$q_{i,j} = \frac{\left(1 + \left(\mathbf{y}_i - \mathbf{y}_j\right)^2\right)^{-1}}{\sum_{i \neq j} \left(1 + \left(\mathbf{y}_i - \mathbf{y}_j\right)^2\right)^{-1}}, \qquad q_{i,i} = 0$$

- When $\left(\mathbf{y}_i - \mathbf{y}_j\right)^2$ is large, the '1' term is negligible and the q distribution will be essentially invariant with respect to the overall length scale.
- Hence, for all but the finest length scales, pairs of points that are very far apart will have a similar contribution as points that are "reasonably" far apart.

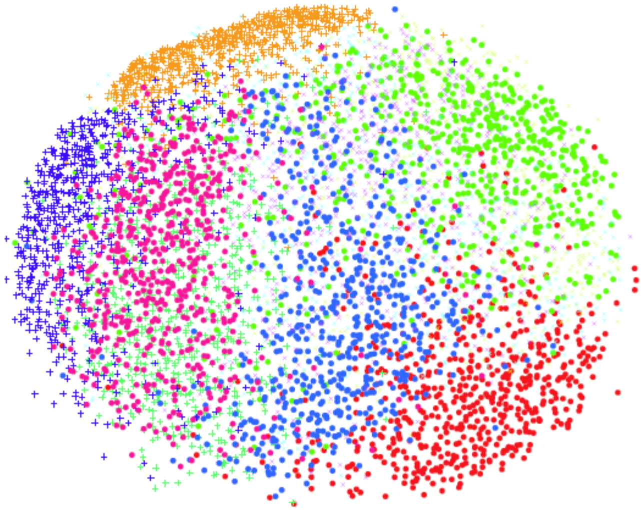
$$\frac{\partial L^{\mathrm{t-SNE}}}{\partial \mathbf{y}_i} = 4 \sum_j \frac{\left(p_{i,j} - q_{i,j}\right)}{1 + \left(\mathbf{y}_i - \mathbf{y}_j\right)^2} \left(\mathbf{y}_i - \mathbf{y}_j\right)$$

MNIST Data

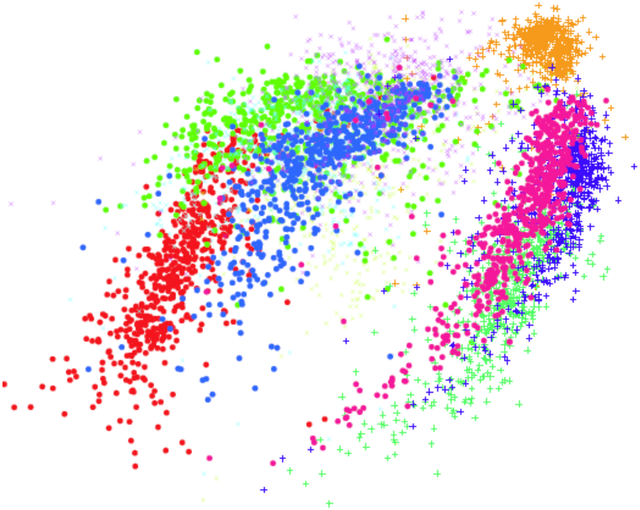
MNIST 2D visualisation: t-SNE



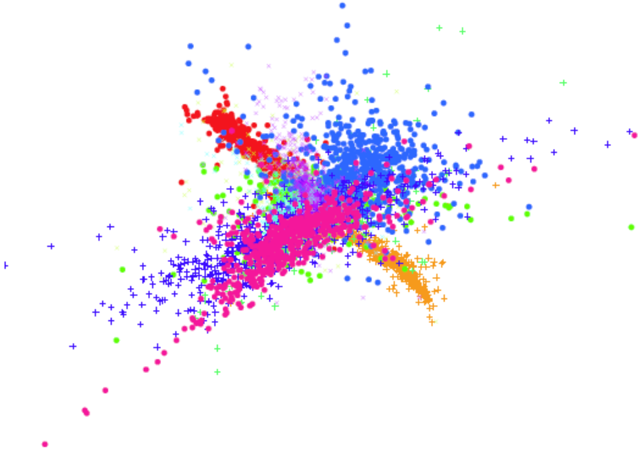
MNIST 2D visualisation: Sammon Mapping



MNIST 2D visualisation: Isomap



MNIST 2D visualisation: LLE



Large Datasets

- Like most visualization methods, t-SNE has an $O(N^2)$ cost to calculate the objective function (it computes distances between all pairs of points).
- For large datasets, it is very expensive to train t-SNE and related methods since each iteration of gradient descent requires an $O(N^2)$ calculation.

One option for scaling

- A cheaper alternative is to first define a desired number of neighbors and calculate a graph of which are the nearest neighbors of each $\mathbf{x}_i$
- We then select (randomly) a small set of "landmark" datapoints in x, indexed by i'. We can then calculate a new transition matrix for these datapoints as follows. Starting from i' we randomly sample j according to $p(j|i = i')$. We repeat sampling from this Markov chain until we land on another landmark $j' \neq i'$. We then repeat this procedure many times for landmark i' and then normalise to obtain the transition $p(j'|i')$. We then repeat this for each landmark i'.
- We then use $p(j'|i')$ in place of the original full $p(j|i)$ transition to find a visualization for the chosen landmark points. Though this is expensive to compute, it only needs happen once.

Other options for scaling

Approximate!

- Only compute similarities over a set of nearest neighbors, and use a fast approach for finding nearest neighbors?
- Accelerate / approximate the gradient computations (Barnes-Hut)?
- Estimate by sampling from approximate nearest neighbor graph (LargeVis)? `https://github.com/elbamos/largeVis`