Visualisation

Dmitry Adamskiy, David Barber

What is Data Visualisation?

- Data is often very high dimensional meaning that we can't directly 'see' the data.
- It's difficult to get intuitions about the data since we can't 'see it'.
- In Data Visualisation we try to find a low dimensional representation (2 or 3 dimensions) so that we 'see something'.
- There is no 'correct' or 'perfect' visualisation. Every low dimensional representation will lose some information contained in the original high dimensional data.
- Such visualisations can be useful to see whether there might be clusters of datapoints, or which datapoints are in some sense 'similar' to another.
- Historically, methods such as PCA or Sammon 'mappings' were popular, but they are now less preferred.
- We tend to heuristically prefer representations that better preserve neighbourhood structure.
- Some visualisation methods (autoencoders) can be good but they are very expensive to train.

Setup

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' visualisation methods (Sammon mapping, Isomap, Locally Linear Embedding) don't work that well on real-world data sets.
- We will focus on Stochastic Neighbour Embedding (SNE) and its 'robust' variant t-SNE which is one of the most popular current approaches.

Stochastic Neighbour Embedding (SNE)

We define an $N \times N$ Markov transition matrix:

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

Similarly we define

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

- The transition p describes the neighbourbood structure – how easily can one jump to other points from a given point.
- If we want to preserve this structure, we need to find $\mathcal{Y}$ such that q is approximately the same as p.
- Note that the $\mathbf{y}_n$ scale is arbitrary, so we have 'fixed' the variance to $1/\sqrt{2}$ in the y space.

SNE

- SNE minimises the KL divergence between each conditional distribution $p_i \equiv p_{\cdot|i}$:

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

- The optimisation is performed using gradient descent with parameters $\mathcal{Y}$.
- One can show (exercise) that

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

- Criticisms about this approach are that the KL divergence is not symmetric.
- For example 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. We therefore need to make such points have an influence on the objective.

Symmetric SNE

There are many potential ways to 'robustify' the SNE procedure. The t-SNE approach, which seems to work reasonably well, is:

- Symmetric SNE uses a 'symmetric' loss

$$E = \mathrm{KL}(p|q) = \sum_i \sum_j p_{i,j} \log \frac{p_{i,j}}{q_{i,j}}$$

  where the 'joint' distribution is defined

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

- Note that 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.
- The gradient is (exercise)

$$\frac{\partial E}{\partial \mathbf{y}_i} = 4 \sum_j (p_{i,j} - q_{i,j}) (\mathbf{y}_i - \mathbf{y}_j)$$

- This works quite well, but there is an additional step that is also useful.

t-SNE

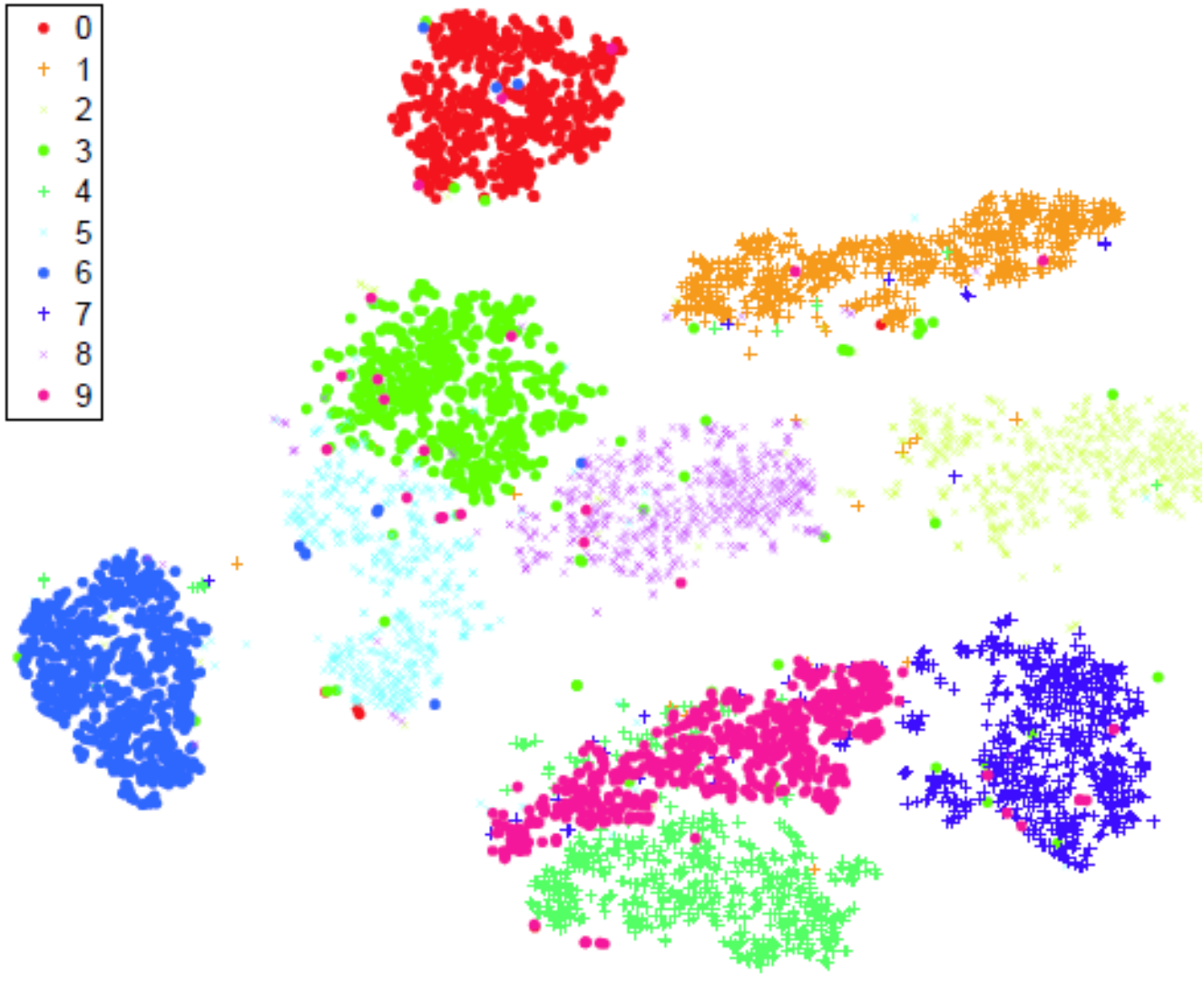
- If we use a student t-distribution (rather than a Gaussian) this has heavier tails and can therefore assign non-negligible mass to y points that are quite far apart. Defining a student t-distribution with a single degree of freedom,

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

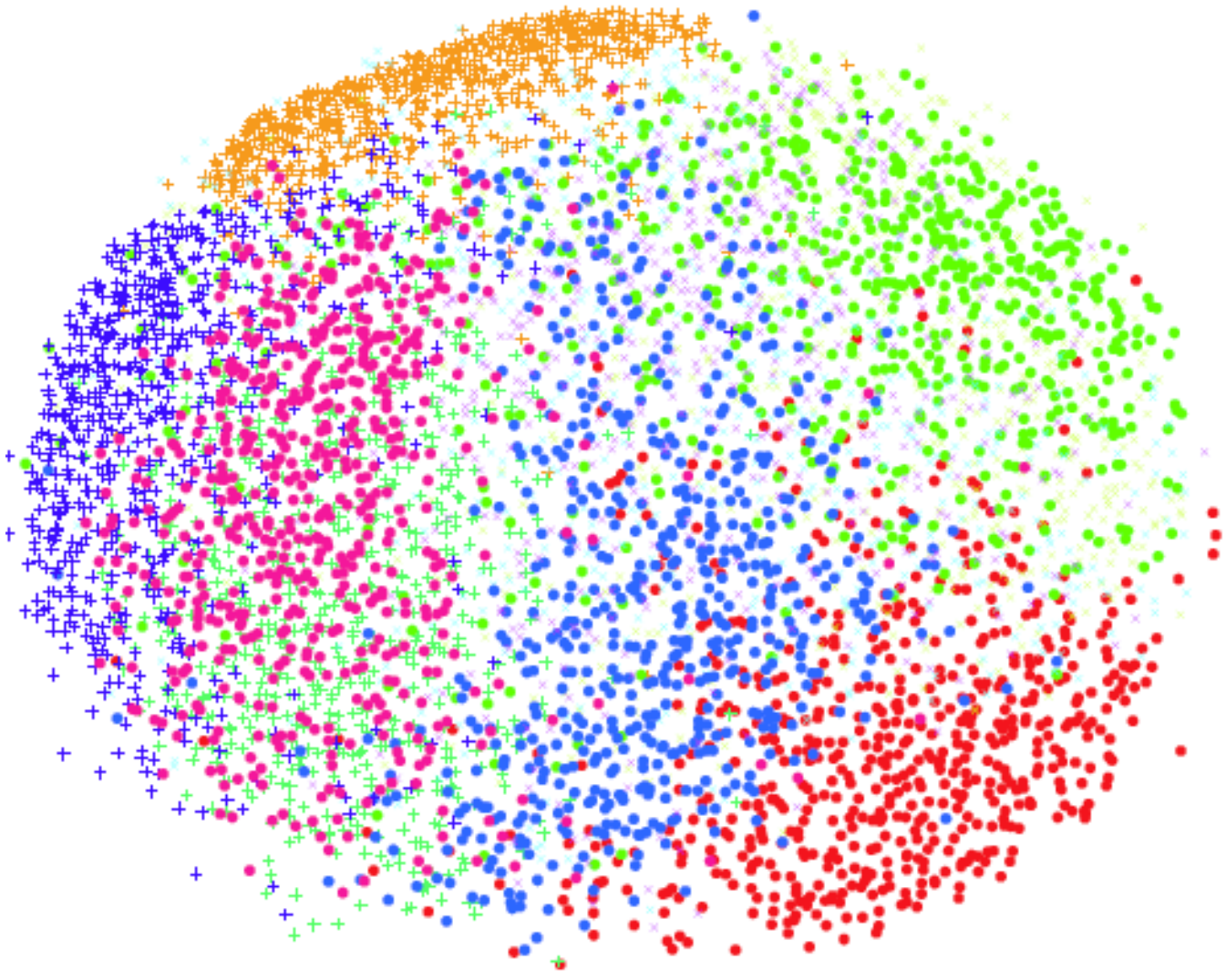
  Note that the sum in the denominator is over all pairs of distinct points.
- When $(\mathbf{y}_i - \mathbf{y}_j)^2$ is large, the '1' term becomes irrelevant and the q distribution will be essentially invariant with respect to the overall length scale.
- Hence, for all but the finest length scales, pairs of points that are very far apart will have a similar contribution to points that are reasonably far apart.
- The gradient is (exercise)

$$\frac{\partial E}{\partial \mathbf{y}_i} = 4 \sum_j \frac{(p_{i,j} - q_{i,j})}{1 + (\mathbf{y}_i - \mathbf{y}_j)^2} (\mathbf{y}_i - \mathbf{y}_j)$$

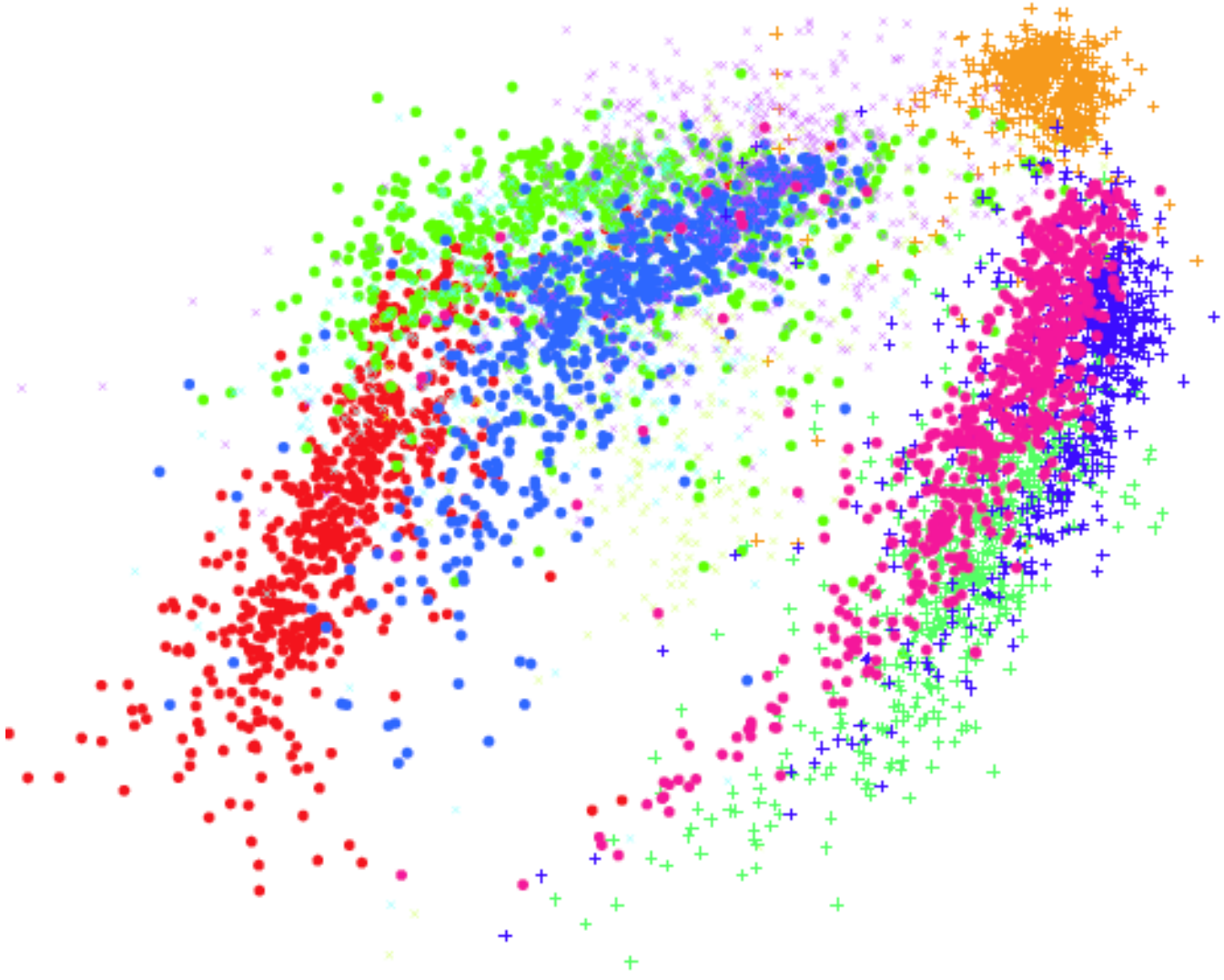
MNIST 2D visualisation: t-SNE



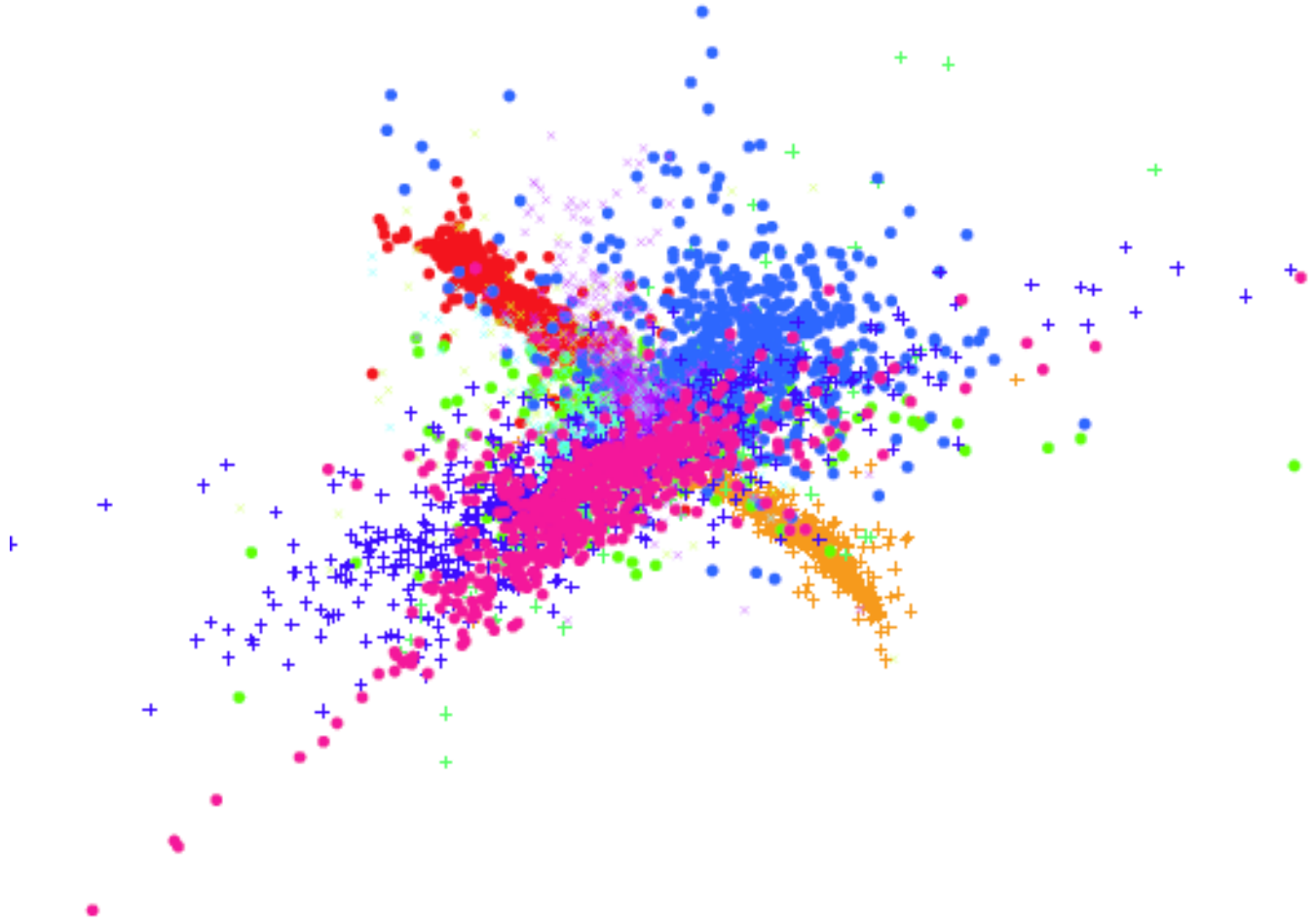
MNIST 2D visualisation: Sammon Mapping



MNIST 2D visualisation: Isomap



MNIST 2D visualisation: LLE



Large Datasets

- Like most visualisation methods, t-SNE has an $O(N^2)$ cost just to calculate the objective function since the distance between all pairs of points is calculated.
- For large datasets, this means that is very expensive to train t-SNE and related methods since each iteration of gradient descent requires an $O(N^2)$ calculation.
- A cheaper alternative is to first define a desired number of neighbours and calculate a graph of which are the nearest neighbours of each node (x datapoint).
- 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 visualisation for the chosen landmark points.
- Whilst it is expensive to calculate $p(j'|i')$, this only needs to be done once.

Speeding things up

Two main components:

- Approximating imput similarities using vantage point trees
  - We want a sparse distribution P with uN non-zero components.
  - Probabilities p_{ij} for dissimilar objects are negligible so this is possible.
- Approximating the gradient using Barnes-Hut algorithm.
  - The gradient components admit representation as sums of attractive and repulsive forces.
  - This allows us to borrow an algorithm from physics to approximate the dynamics of N-body problem.

Approximating input similarities

- We redefine the pairwise input similarities like this:

$$p_{j|i} = \begin{cases} \frac{\exp\left(-(\mathbf{x}_i - \mathbf{x}_j)^2/(2\sigma_i^2)\right)}{\sum_{j \neq i} \exp\left(-(\mathbf{x}_i - \mathbf{x}_j)^2/(2\sigma_i^2)\right)}, \text{if} j \in N_i \\ 0, \text{otherwise} \end{cases}$$

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

- Here N_i is the set of $3u$ nearest neighbours of x_i.
- They are found in $O(uN \log N)$ using a vantage-point tree.

Vantage-Point trees

Vantage-point tree is a binary tree where each node stores

1. an input object,
2. radius of a ball centered on this object.

For each non-leaf node, the set of objects associated with the tree rooted at this node is partitioned into those located inside the ball and outside. They are stored in the left and right subtrees respectively.

VP-tree: construction

In t-SNE implementation, the vp-tree is constructed as follows:

Input: set of input objects
function BUILDTREE(P: set of points)
if P is empty **then return** Null
else if P has one point p **then**
return node = Node(p)
else
Select random point $p \in P$, node = Node(p).
Find the median distance d from p to points in $P \setminus p$
Partition $P \setminus p$ into P_l and P_r such that $dist(p, q) < d$ for $q \in P_l$ and $dist(p, q) \geq d$ for $q \in P_r$.
node $\rightarrow$ left = BuildTree(P_l)
node $\rightarrow$ right = BuildTree(P_r)
return node
end if
end function

VP-tree: search

Now suppose we need to find k nearest neighbours for a query point q. We are going to start at the root of the tree and maintain the radius of interest τ around the query point (the distance from q to the furthest among current k candidates).

- Perform depth-first search of VP-tree
- Before going into any subtree, compute whether the τ-ball centered at q is completely inside/outside relevant ball. If it is, do not go into the corresponding subtree.
- When choosing which child to explore first, go to the one where the query point q would be placed.

VP-tree: references

- t-SNE follow-up paper, describing VP-tree usage.
- Nice overview of VP-tree
- Original Yanilos paper

Approximating the gradient

- Recall that the gradient is:

$$\frac{\partial E}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} \frac{(p_{i,j} - q_{i,j})}{1 + (\mathbf{y}_i - \mathbf{y}_j)^2} (\mathbf{y}_i - \mathbf{y}_j) = 4 \sum_{j \neq i} (p_{i,j} - q_{i,j}) \, q_{ij} Z \, (\mathbf{y}_i - \mathbf{y}_j)$$

- We can split it as:

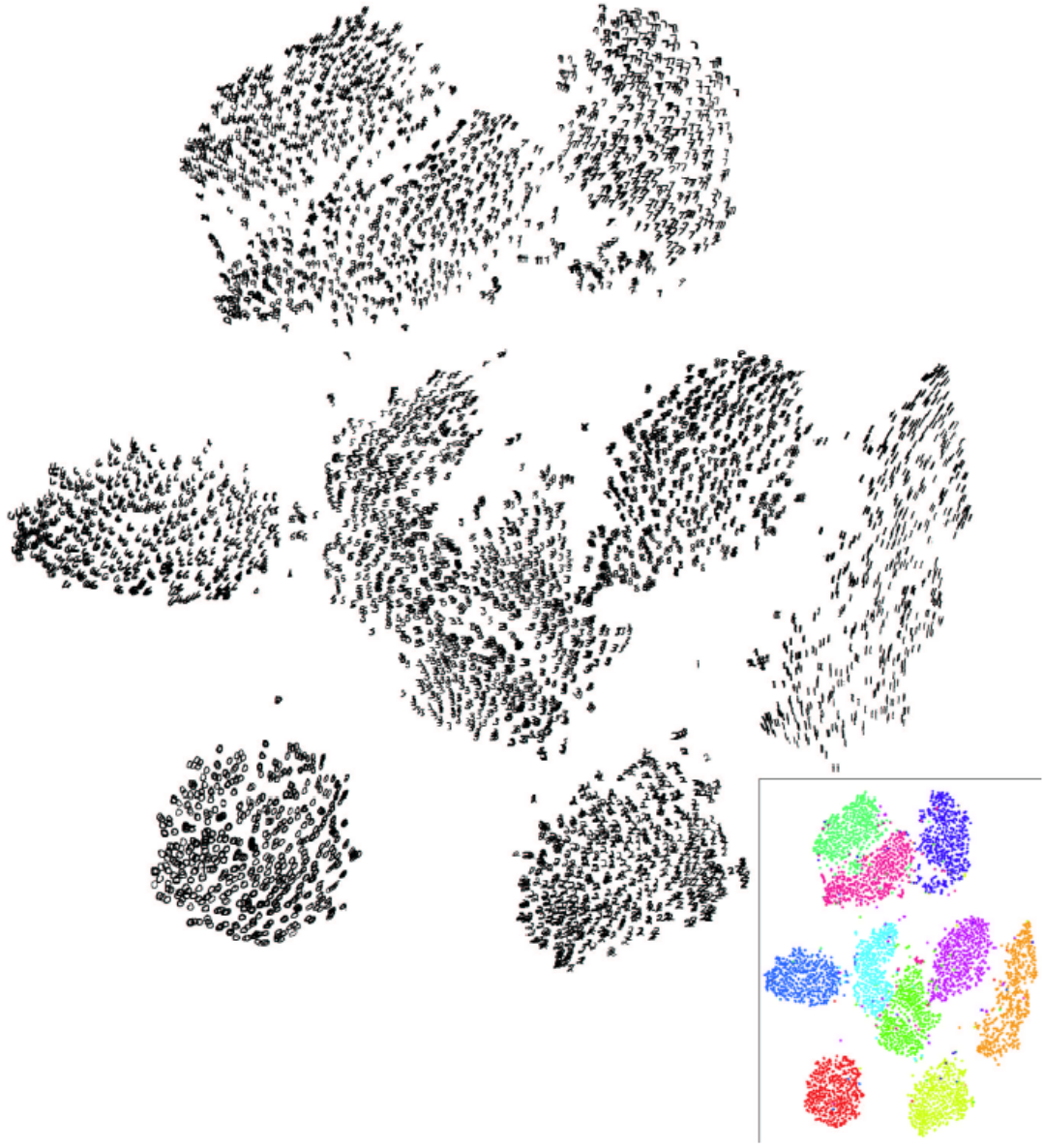
$$\frac{\partial E}{\partial \mathbf{y}_i} = 4(F_{attr} + F_{rep}) = 4\Big(\sum_{j \neq i} p_{i,j} \, q_{i,j} Z \, (\mathbf{y}_i - \mathbf{y}_j) - \sum_{j \neq i} q_{i,j}^2 Z \, (\mathbf{y}_i - \mathbf{y}_j)\Big)$$

- Intuition: we can think of the components of the gradient as attractive and repulsive forces acting on y_i.
- Computing F_{attr} is efficient: it needs summing over non-zero elements in P in $O(uN)$.
- However, computing F_{rep} is still $O(N^2)$.

Barnes-Hut approximation

- Consider three points y_i, y_j, y_k, such that y_j and y_k are close to each other, but are both far away from y_k. In this case the contrbutions of y_j and y_k to F_{rep} is roughly the same.
- We construct a quadtree (or octtree) on the current embedding (see next lecture), storing the centre of masses at each cell.
- Traverse the quadtree, at every node deciding whether the corresponding cell could be used as a summary for the contributions of points in it.
- Great visualisation of Barnes-Hut using d3

MNIST 2D visualisation: t-SNE



Visualisation of 6000 landmark points (using the random walk transition).