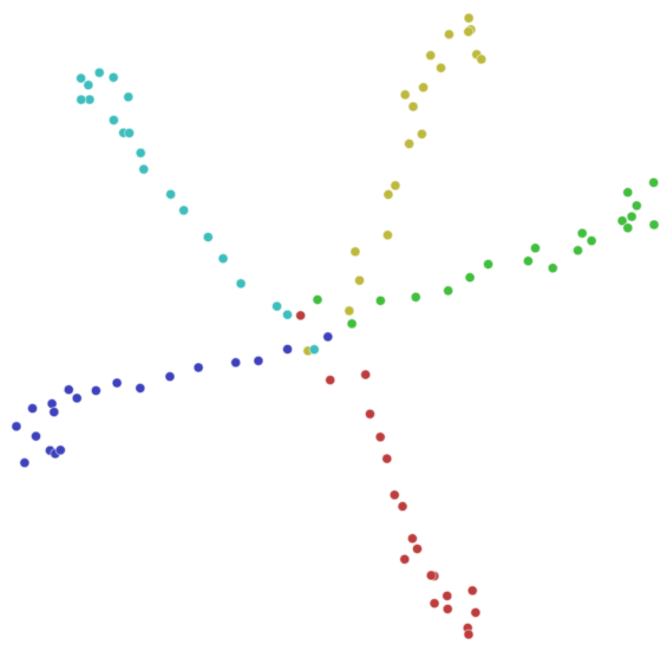
Understanding UMAP

Andy Coenen, Adam Pearce | *Google PAIR*

Dimensionality reduction is a powerful tool for machine learning practitioners to visualize and understand large, high dimensional datasets. One of the most widely used techniques for visualization is t-SNE, but its performance suffers with large datasets and using it correctly can be challenging.

UMAP is a new technique by McInnes et al. that offers a number of advantages over t-SNE, most notably increased speed and better preservation of the data's global structure. In this article, we'll take a look at the theory behind UMAP in order to better understand how the algorithm works, how to use it effectively, and how its performance compares with t-SNE.



Points arranged in a radial star pattern

Step

400

Dataset Parameters

Number of points 100

Number of arms 5

Dimensions 17

UMAP Parameters

n_neighbors 44

min_dist 0.15

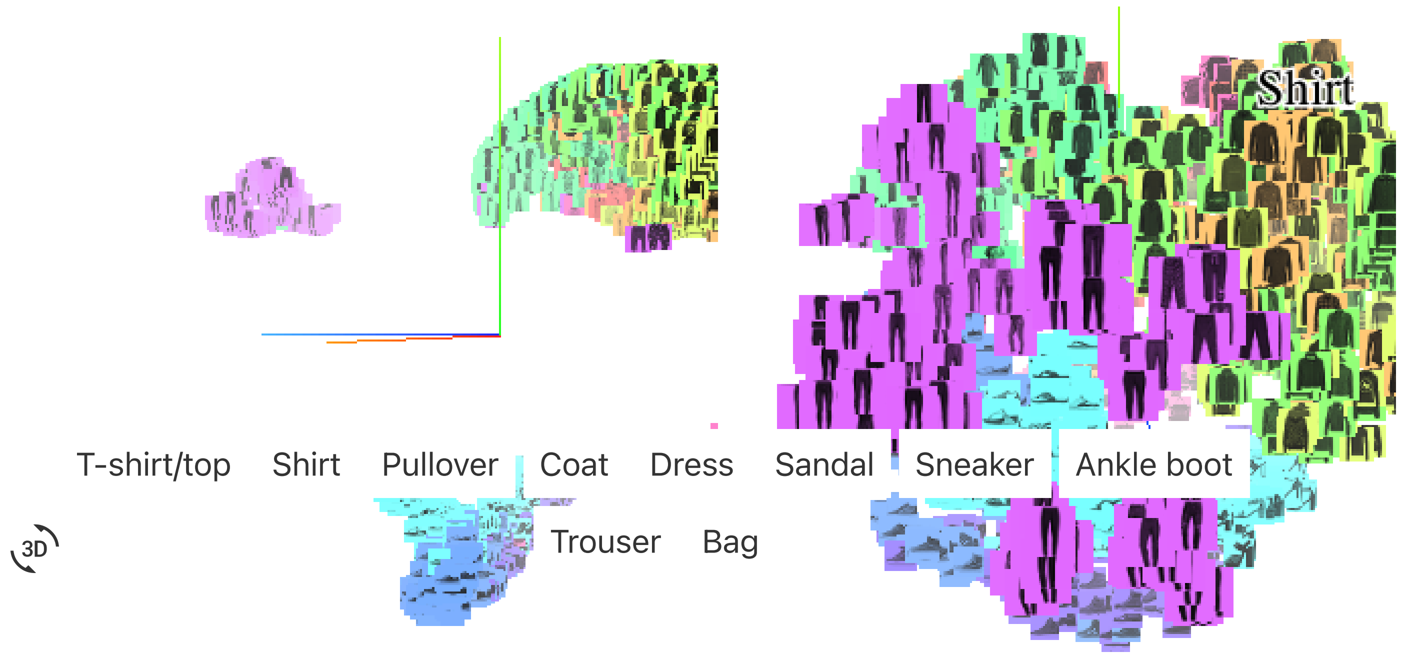
Figure 1: *Apply UMAP projection to various toy datasets, powered by umap-js.*

So what does UMAP bring to the table? Most importantly, UMAP is fast, scaling well in terms of both dataset size and dimensionality. For example, UMAP can project the 784-dimensional, 70,000-point MNIST dataset in less than 3 minutes, compared to 45 minutes for scikit-learn's t-SNE implementation. Additionally, UMAP tends to better preserve the global structure of the data. This can be attributed to UMAP's strong theoretical foundations, which allow the algorithm to better strike a balance between emphasizing local versus global structure.

UMAP vs t-SNE, at first glance

Before diving into the theory behind UMAP, let's take a look at how it performs on real-world, high-dimensional data. The following visualization shows a comparison between using UMAP and t-SNE to project a subset of the 784-dimensional Fashion MNIST dataset down to 3 dimensions. Notice how well clustered each different category is (local structure), while similar categories (such as sandal, sneaker, and ankle boot) tend to colocate (global structure).

UMAP **t-SNE**



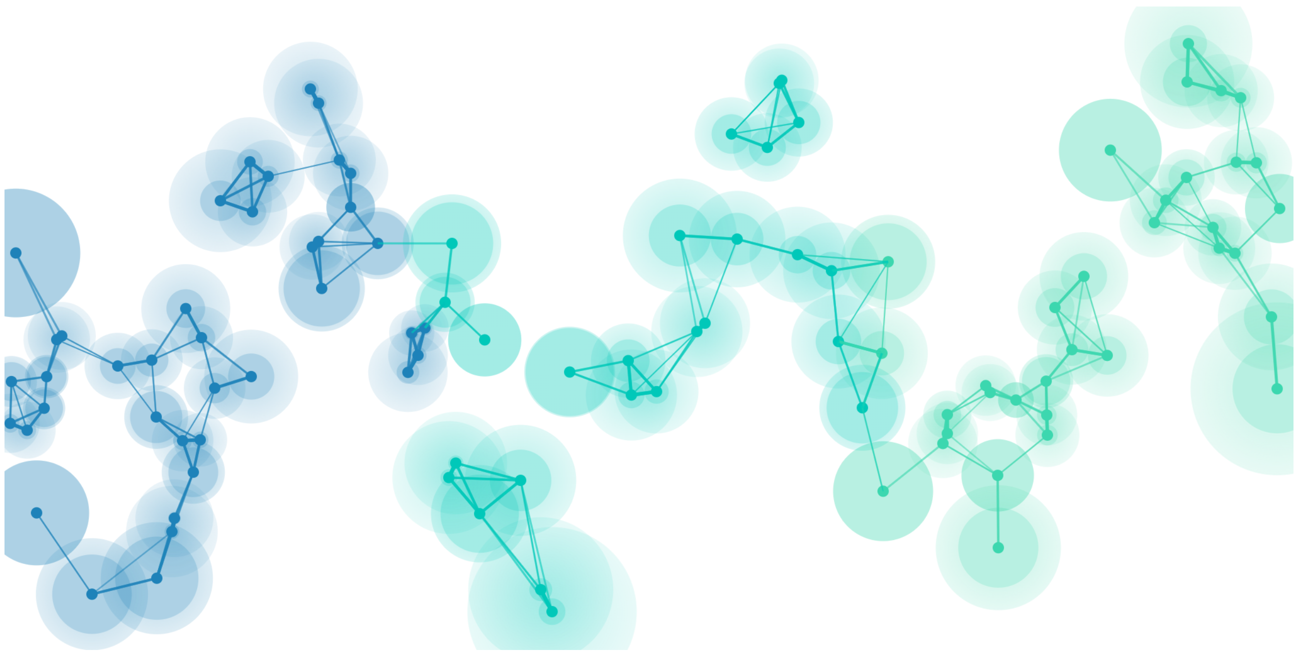
Whil

together, UMAP much more clearly separates these groups of similar categories from each other. It's also worth noting that UMAP projection of the dataset took 4 minutes in comparison to 27 minutes with multicore t-SNE.

A dip into UMAP theory

UMAP, at its core, works very similarly to t-SNE - both use graph layout algorithms to arrange data in low-dimensional space. In the simplest sense, UMAP constructs a high dimensional graph representation of the data then optimizes a low-dimensional graph to be as structurally similar as possible. While the mathematics UMAP uses to construct the high-dimensional graph is advanced, the intuition behind them is remarkably simple.

In order to construct the initial high-dimensional graph, UMAP builds something called a "fuzzy simplicial complex". This is really just a representation of a weighted graph, with edge weights representing the likelihood that two points are connected. To determine connectedness, UMAP extends a radius outwards from each point, connecting points when those radii overlap. Choosing this radius is critical - too small a choice will lead to small, isolated clusters, while too large a choice will connect everything together. UMAP overcomes this challenge by choosing a radius locally, based on the distance to each point's *n*th nearest neighbor. UMAP then makes the graph "fuzzy" by decreasing the likelihood of connection as the radius grows. Finally, by stipulating that each point must be connected to at least its closest neighbor, UMAP ensures that local structure is preserved in balance with global structure.



extent: 50%
n_nearest: 4

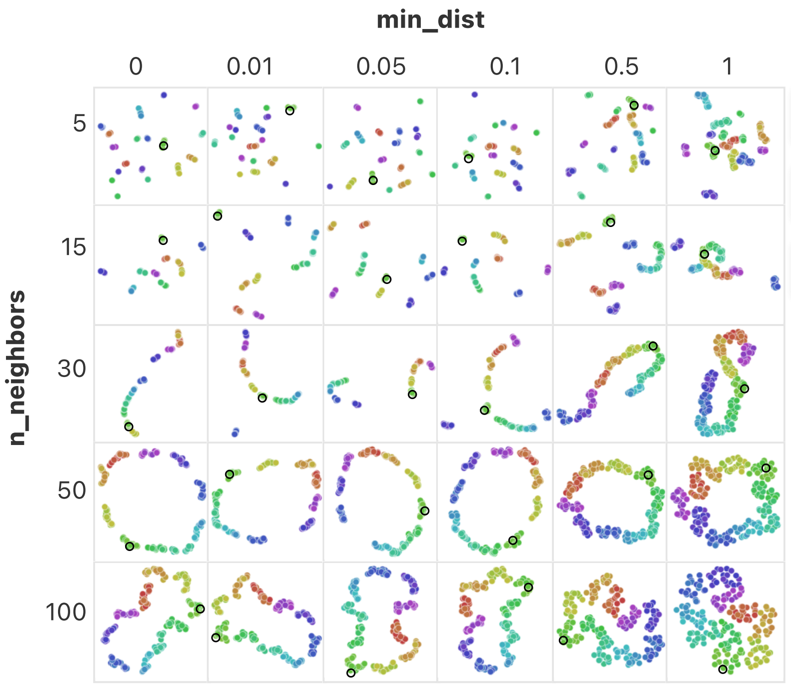
Figure 3: *Adjust the slider to extend a radius outwards from each point, computed by the distance to its nth nearest neighbor. Notice that past the intersection with the first neighbor, the radius begins to get fuzzy, with subsequent connections appearing with less weight;*

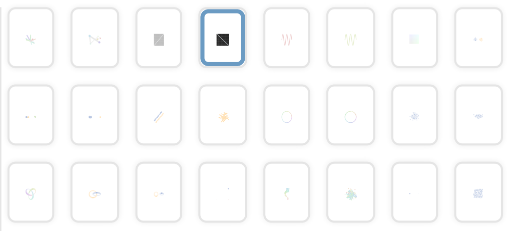
Once the high-dimensional graph is constructed, UMAP optimizes the layout of a low-dimensional analogue to be as similar as possible. This process is essentially the same as in t-SNE, but using a few clever tricks to speed up the process.

The key to effectively using UMAP lies in understanding the construction of the initial, high-dimensional graph. Though the ideas behind the process are very intuitive, the algorithm relies on some advanced mathematics to give strong theoretical guarantees about how well this graph actually represents the data. Interested readers can dive deeper into the entire process in the supplementary section: A deeper dive into UMAP theory.

UMAP Parameters

By understanding the theory behind UMAP, it becomes much easier to understand the algorithm's parameters, especially compared with the `perplexity` parameter in t-SNE. We'll consider the two most commonly used parameters: `n_neighbors` and `min_dist`, which are effectively used to control the balance between local and global structure in the final projection.





nxn images of a line rotated around the center, represented as an n*n dimensional vector. Grouped by similar angles.

Number of lines: 200
Number of clusters: 10
Noise: 8
Pixels per side: 28

Figure 4: *UMAP projection of various toy datasets with a variety of common values for the* `n_neighbors` *and* `min_dist` *parameters.*

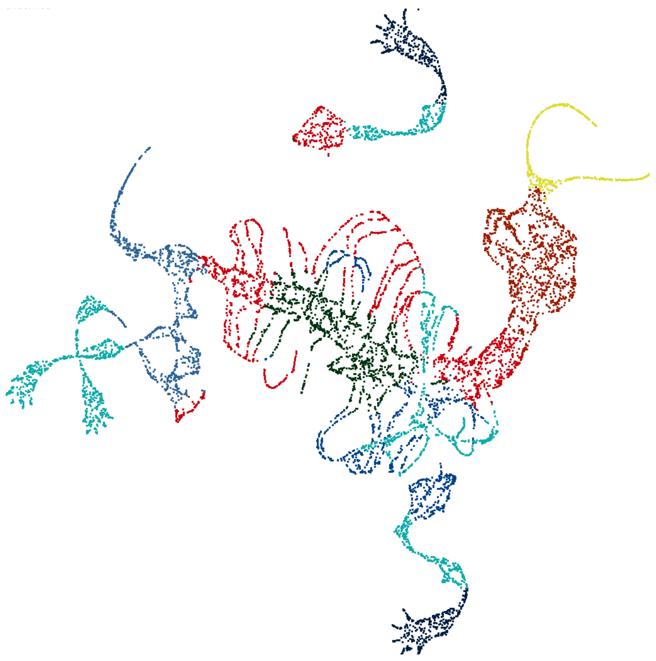
The most important parameter is `n_neighbors` - the number of approximate nearest neighbors used to construct the initial high-dimensional graph. It effectively controls how UMAP balances local versus global structure - low values will push UMAP to focus more on local structure by constraining the number of neighboring points considered when analyzing the data in high dimensions, while high values will push UMAP towards representing the big-picture structure while losing fine detail.

The second parameter we'll investigate is `min_dist`, or the minimum distance between points in low-dimensional space. This parameter controls how tightly UMAP clumps points together, with low values leading to more tightly packed embeddings. Larger values of `min_dist` will make UMAP pack points together more loosely, focusing instead on the preservation of the broad topological structure.

The following visualization - extended from excellent work by Max Noichl - is an exploration of the impact of UMAP parameters on a 2D projection of 3D data. By changing the `n_neighbors` and `min_dist` parameters, you can explore their impact on the resulting projection.

Original 3D Data

2D UMAP Projection



n_neighbors: 100

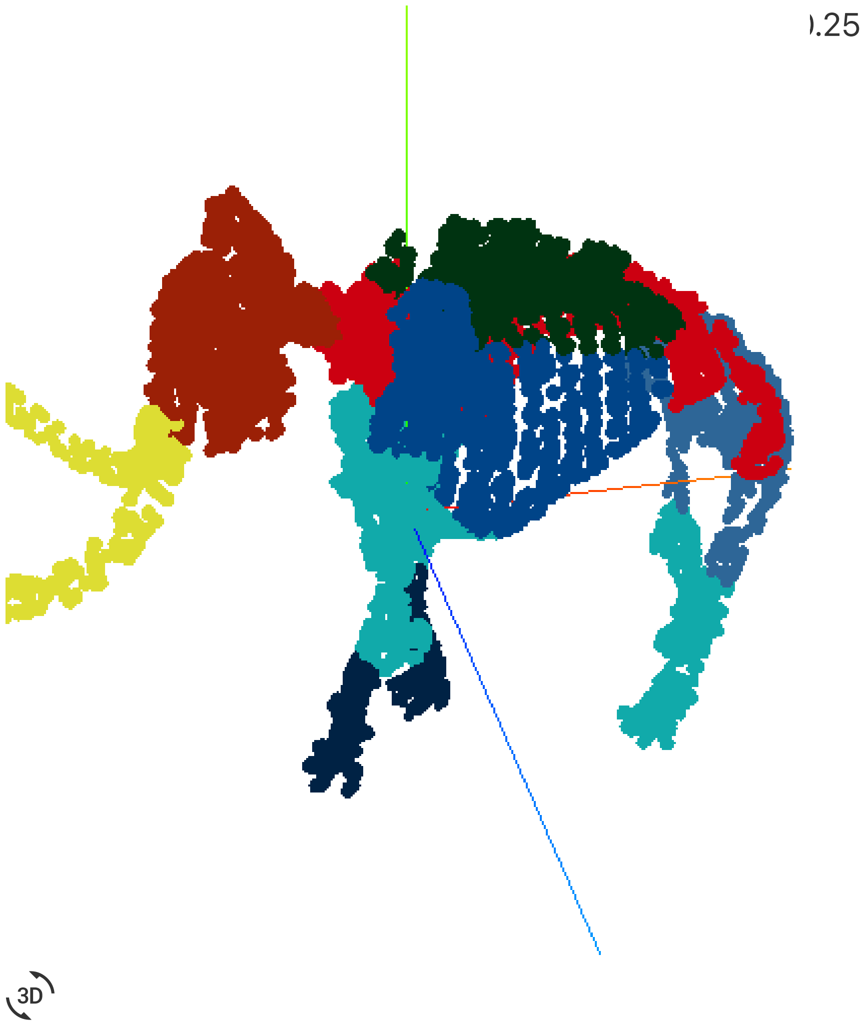


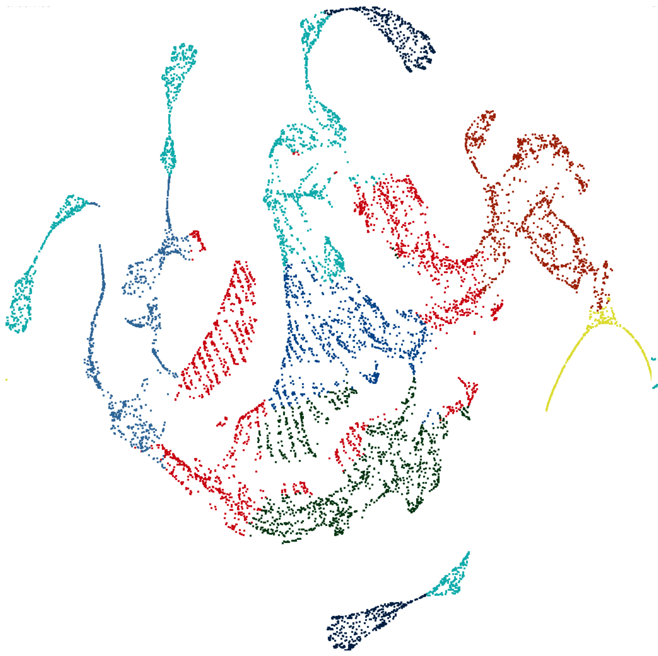
Figure 5: *UMAP projections of a 3D woolly mammoth skeleton (50k points, 10k shown) into 2 dimensions, with various settings for the* `n_neighbors` *and* `min_dist` *parameters.*

While most applications of UMAP involve projection from high-dimensional data, the projection from 3D serves as a useful analogy to understand how UMAP prioritizes global vs local structure depending on its parameters. As `n_neighbors` increases, UMAP connects more and more neighboring points when constructing the graph representation of the high-dimensional data, which leads to a projection that more accurately reflects the global structure of the data. At very low values, any notion of global structure is almost completely lost. As the `min_dist` parameter increases, UMAP tends to "spread out" the projected points, leading to decreased clustering of the data and less emphasis on global structure.

UMAP vs t-SNE, revisited

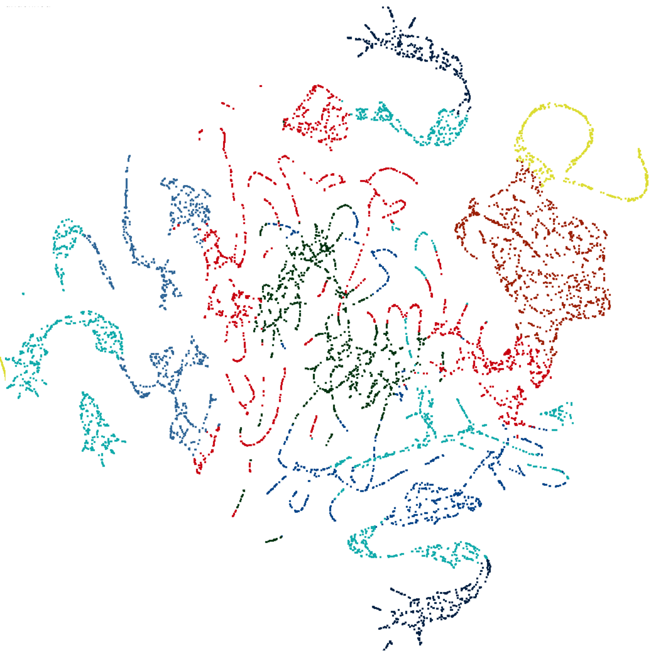
The biggest difference between the the output of UMAP when compared with t-SNE is this balance between local and global structure - UMAP is often better at preserving global structure in the final projection. This means that the inter-cluster relations are potentially more meaningful than in t-SNE. However, it's important to note that, because UMAP and t-SNE both necessarily warp the high-dimensional shape of the data when projecting to lower dimensions, any given axis or distance in lower dimensions still isn't directly interpretable in the way of techniques such as PCA.

2D t-SNE projection



perplexity: 500
time: 42m 24s

2D UMAP projection



n_neighbors: 10
min_dist: 0.1
time: 51s

Figure 6: *A comparison between UMAP and t-SNE projections of a 3D woolly mammoth skeleton (50,000 points) into 2 dimensions, with various settings for parameters. Notice how much more global structure is preserved with UMAP, particularly with larger values of* `n_neighbors` .

Going back to the 3D mammoth example, we can easily see some big differences between the two algorithms' output. For lower values of the `perplexity` parameter, t-SNE tends to "spread out" the projected data with very little preservation of the global structure. In contrast, UMAP tends to group adjacent pieces of the higher-dimensional structure together in low dimensions, which reflects an increased preservation of global structure. Note that, using t-SNE, it takes an extremely high perplexity (~1000) to begin to see the global structure emerge, and at such large perplexity values the time to compute is dramatically longer. It's also notable that t-SNE projections vary widely from run to run, with different pieces of the higher-dimensional data projected to different locations. While UMAP is also a stochastic algorithm, it's striking how similar the resulting projections are from run to run and with different parameters. This is due, again, to UMAP's increased emphasis on global structure in comparison to t-SNE.

It's worth noting that t-SNE and UMAP wind up performing very similarly on the toy examples from earlier figures, with the notable exception of the following example: A dense, tight cluster inside of a wide, sparse cluster. Interestingly, UMAP is unable to separate the two nested clusters, especially when dimensionality is high.

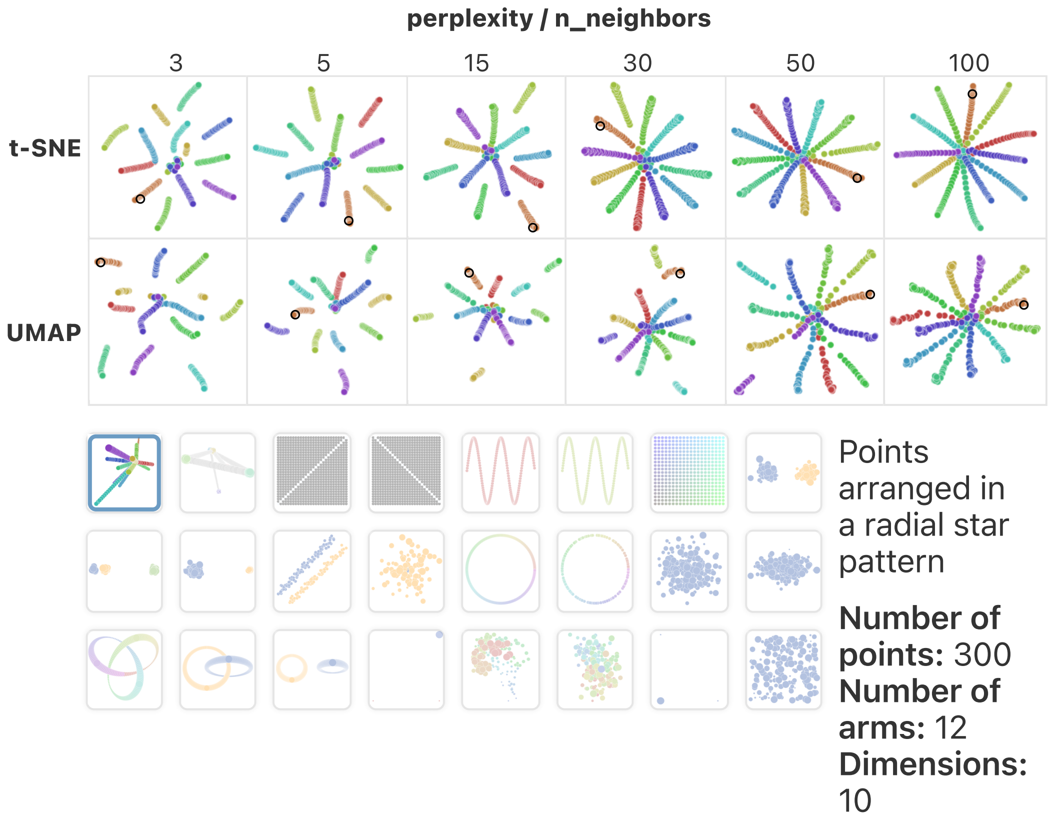


Figure 7: *Comparison between UMAP and t-SNE projecting various toy datasets.*

The failure of the algorithm to handle this case of "containment" may be explained by UMAP's use of local distances in the initial graph construction. Since distances between points in high dimensions tend to be very similar (the curse of dimensionality), UMAP seems to be connecting the outer points of the inner cluster to those of the outer cluster. This, in effect, blends the two clusters together.

How to (mis)read UMAP

While UMAP offers a number of advantages over t-SNE, it's by no means a silver bullet - and reading and understanding its results requires some care. It's worth revisiting our previous work on (mis)reading t-SNE, since many of the same takeaways apply to UMAP:

1. Hyperparameters really matter

Choosing good values isn't easy, and depends on both the data and your goals (eg, how tightly packed the projection ought to be). This is where UMAP's speed is a big advantage - By running UMAP multiple times with a variety of hyperparameters, you can get a better sense of how the projection is affected by its parameters.

2. Cluster sizes in a UMAP plot mean nothing

Just as in t-SNE, the size of clusters relative to each other is essentially meaningless. This is because UMAP uses local notions of distance to construct its high-dimensional graph representation.

3. Distances between clusters might not mean anything

Likewise, the distances between clusters is likely to be meaningless. While it's true that the global positions of clusters are better preserved in UMAP, the distances between them are not meaningful. Again, this is due to using local distances when constructing the graph.

4. Random noise doesn’t always look random.

Especially at low values of `n_neighbors`, spurious clustering can be observed.

5. You may need more than one plot

Since the UMAP algorithm is stochastic, different runs with the same hyperparameters can yield different results. Additionally, since the choice of hyperparameters is so important, it can be very useful to run the projection multiple times with various hyperparameters.

Conclusion

UMAP is an incredibly powerful tool in the data scientist's arsenal, and offers a number of advantages over t-SNE. While both UMAP and t-SNE produce somewhat similar output, the increased speed, better preservation of global structure, and more understandable parameters make UMAP a more effective tool for visualizing high dimensional data. Finally, it's important to remember that no dimensionality reduction technique is perfect - by necessity, we're distorting the data to fit it into lower dimensions - and UMAP is no exception. However, by building up an intuitive understanding of how the algorithm works and understanding how to tune its parameters, we can more effectively use this powerful tool to visualize and understand large, high-dimensional datasets.

Acknowledgements

We'd like to thank Max Noichl for the delightful mammoth visualization idea, and the Smithsonian Institute for the 3D mammoth model. We'd also like to thank Leland McInnes for his excellent feedback, and of course the UMAP library itself.

Source code available on github. Interactive UMAP projections powered by umap-js. 3D Scatter plots powered by scatter-gl.