Mid Term Exam
IST 597
Physics-Informed Machine Learning

Subarna Pudasaini (sfp5828@psu.edu)

Question 1

Implement a t-SNE algorithm from scratch and visualize the embedding of the MNIST dataset.

Ans:

- No. of datapoints: 1000
- No. of optimization iterations: 1000

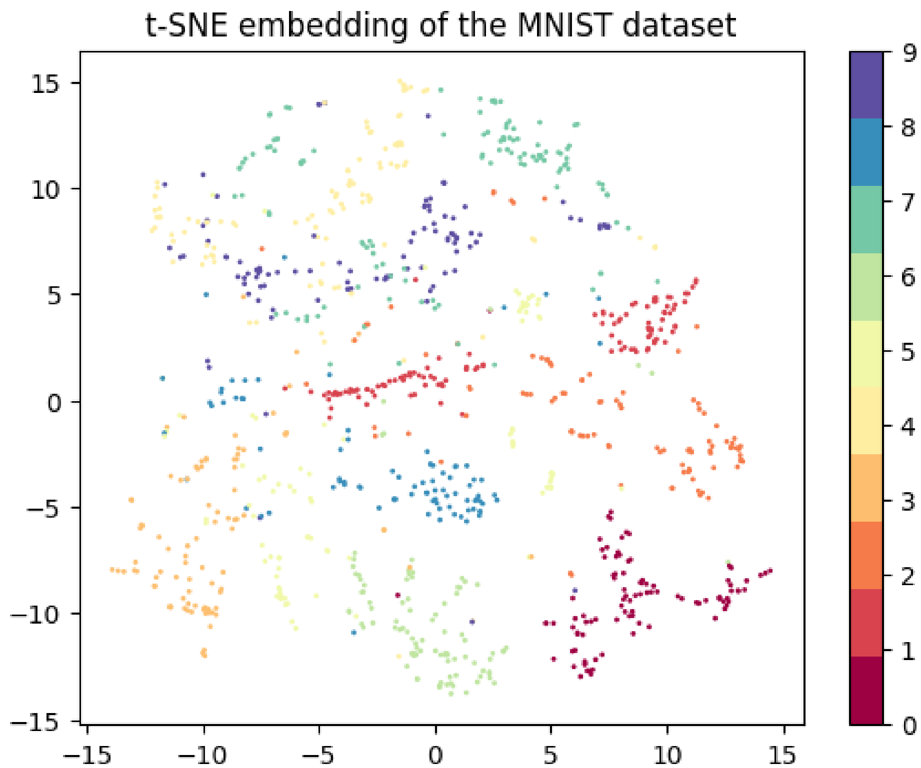

Figure 1: MNIST Data Visualization using t-SNE

Question 2

Derive the computational complexity for computing the gradient for t-sne using the closed-form formulation.

Ans:

The formula for the gradient of the KL divergence is:

$$\frac{\partial \text{KL}}{\partial y_i} = 4 \sum_j (P_{ij} - Q_{ij})(y_i - y_j) Q_{ij}$$

where each term inside the summation is:

$$(P_{ij} - Q_{ij})(y_i - y_j) Q_{ij}$$

Assuming n is the number of data points and k is the number of dimensions.

Complexity for Each Term Inside the Summation

1. **Calculating** $P_{ij} - Q_{ij}$:
   - **Operation**: Subtraction of two scalars.
   - **Complexity**: $O(1)$
2. **Calculating** $y_i - y_j$:
   - **Operation**: Subtraction of two vectors of dimension k.
   - **Complexity**: $O(k)$ (specifically k subtractions)
3. **Calculating** $(y_i - y_j)Q_{ij}$:
   - **Operation**: Scaling the vector $(y_i - y_j)$ by the scalar Q_{ij}.
   - **Complexity**: $O(k)$ (specifically k multiplications)
4. **Calculating** $(P_{ij} - Q_{ij})(y_i - y_j)Q_{ij}$:
   - **Operation**: Multiplying the scalar $(P_{ij} - Q_{ij})$ with the vector $(y_i - y_j)Q_{ij}$.
   - **Complexity**: $O(k)$ (specifically k multiplications)

Total Complexity of One Summation Term

Combining the above steps gives:

$$O(1) + O(k) + O(k) + O(k) = O(3k + 1)$$

Complexity for One i

Since we sum over $j = 1, \ldots, n$ for each i (there are $n - 1$ terms in the summation), the complexity for the entire summation for each i is:

$$(n - 1) \times O(3k + 1) = O(3nk + n - 3k - 1)$$

Total Complexity for All n Gradients

This calculation is performed for each point i (there are n points), so we multiply by n:

$$n \times O(3nk + n - 3k - 1) = O(3n^2k + n^2 - 3nk - n)$$

Question 3

Consider methods to accelerate t-SNE beyond its standard formulation. Provide implementations and statistical evidence of speed-up.

Ans:

Before Optimization:

Experiment Parameter:

- No. of runs: 1
- No. of datapoints: 1000
- No. of optimization iterations: 1000

Experiment Result:

Average Time (s): 3170.8262 (gradient descent with momentum)

Only one run was carried out because the implementation was very slow.
Code: t_sne_serial.ipynb

After Optimization:

Optimization: Used the Python package 'numba' to generate machine-level code and to parallelize the implementations.

Formulations:

1. tsne_base : Simple gradient descent (no momentum).
2. tsne_momentum : Gradient descent with momentum.
3. tsne_pca : The data is first compressed using PCA into 30 dimensions. Then, gradient descent with momentum is applied.

Experiment Parameters:

- No. of runs: 10
- No. of datapoints: 1000
- No. of optimization iterations: 1000

Experiment Result:

Implementation	**Average Time (s)**
tsne_base	33.3966
tsne_momentum	33.0514
tsne_pca	33.2430

Table 1: Comparison of Average Time for Different Implementations

The 3rd implementation might be slower due to slow PCA implementation.
Code: t_sne.ipynb

Speed-up Factor

The speed-up factor (for tsen_momentum implementation) is calculated as follows:

$$\text{Speed-Up Factor} = \frac{3170.8262}{33.0514} \approx 95.94$$

Thus, the optimized code runs approximately 95.94 times faster.

References

1. Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(11).

2. Implementing t-SNE in Python with Optimized Code and Examples (https://tinyurl.com/te4m9kzn)