

Default values for these arguments: `pca=0.95` (run PCA on the raw dataset to keep 95% of the variance), `seed=2020` (if not set manually), `margin=0.5` (relative margin of 0.5).

For testing the method only, consider to run with `--no-score` option (without calculating the quality scores `AUC[R_NX]` and `AUC[G_NN]` since they are $O(N^2 \log N)$).

Example: Run Hct-SNE for a subset of 10K of the CIFAR10 dataset

```
# run Hct-SNE with CIFAR10 (set -n 50000 for running with full dataset)
python main.py -d cifar10 -n 10000 --margin 0.5 --rerun0 --rerun1 --seed 2020
```

Run other methods (UMAP, cat-SNE) for comparison (using `compare.py` script)

```
# run UMAP with CIFAR10 (set -n 50000 for running with full dataset)
python compare.py -d cifar10 -n 10000 --method umap --seed 2020

# run cat-SNE for CIFAR10. This method is not optimized for large dataset,
# It can just be run with a subset of 10K points
# (and it takes around 1h30 in the colab environment.)
python compare.py -d cifar10 -n 10000 --method catsne --seed 2020
```

Hyper-parameters

Hct-SNE uses the same hyper-parameter of tSNE (for creating the initial embedding) without exaggeration phase.

```
tsne_kwargs = dict(
    perplexity=50,
    random_state=2020,
    initialization = "pca", # change it to `random` for running different times
    negative_gradient_method = "bh", # use Barnes-Hut acceleration
    learning_rate = "auto", # will be set to N/12
    early_exaggeration = 12,
    early_exaggeration_iter = 250,
    theta=0.5, # parameters for Barnes-Hut tree
)
```

Hct-SNE implements a new objective function `my_kl_divergence_bh` and injects into the optimization loop of openTSNE.

```
hctsne_kwargs = dict(
    ..., # the same arguments as t-SNE as above

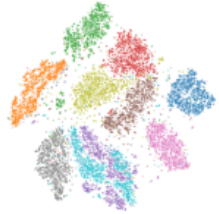
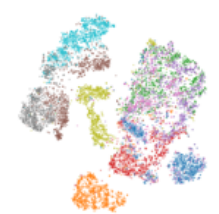

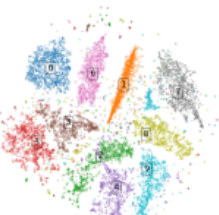



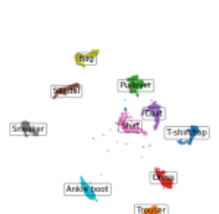

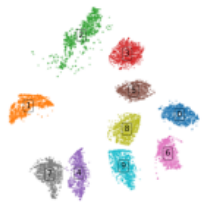


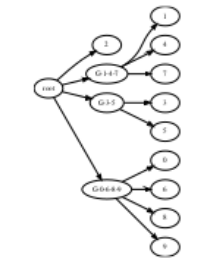


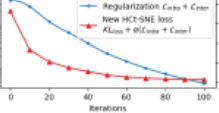
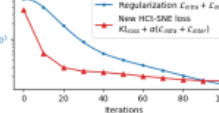
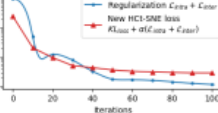
    # disable exaggeration
    early_exaggeration = 0,
    early_exaggeration_iter = 0,

    # new objective function that takes
    # the hierarchical constraints in `tree_regularizer`
    negative_gradient_method=partial(
        my_kl_divergence_bh,
        list_regularizer=[(alpha, tree_regularizer)],
        logger=loss_logger,
    )
)
```

The `alpha` hyper-parameter for each dataset can be easily tuned by choosing the value that makes the new objective function decrease. In the paper, `alpha=7.5e-4` for MNIST and Fashion-MNIST and `alpha=5e-3` for CIFAR10.


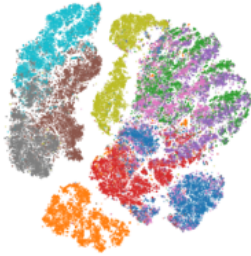



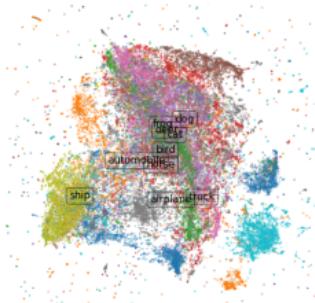


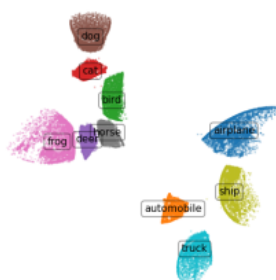
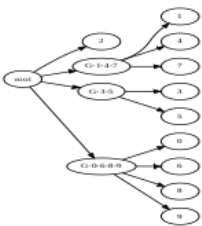


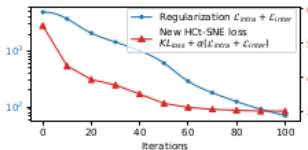
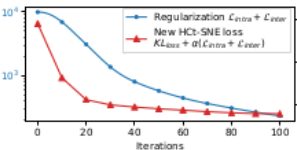
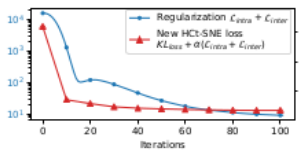
Results

Visualization with the subset of 10K for each dataset

	MNIST (10K)	Fashion-MNIST (10K)	CIFAR10 (10K)
Unsupervised <i>t</i> -SNE			
<i>cat</i> -SNE			
Supervised UMAP			
H <i>Ct</i> -SNE (ours)			
Hierarchy for H <i>Ct</i> -SNE			
Loss values of H <i>Ct</i> -SNE			

Visualization with full datasets

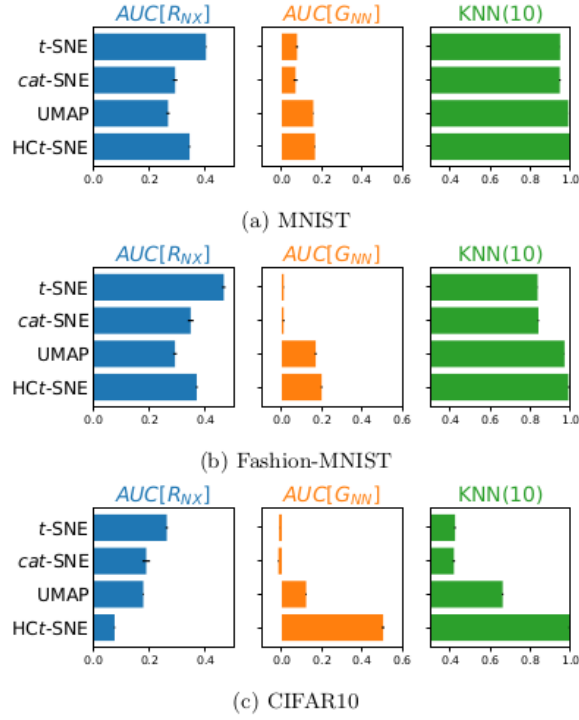
We use the full training set of each dataset (60K for MNIST, 60K for Fashion-MNIST and 50K for CIFAR10). (The test set is reserved for testing the embedding with new points -- not presented in this version). cat-SNE cannot run with the full dataset.

	MNIST (60K)	Fashion-MNIST (60K)	CIFAR10 (50K)
Unsupervised <i>t</i> -SNE			
	KNN score 0.97	KNN score 0.87	KNN score 0.45
Supervised UMAP			
	KNN score 0.99	KNN score 0.98	KNN score 0.63
HCT-SNE (Ours)			
	KNN score 1.0	KNN score 1.0	KNN score 1.0
Hierarchy			
Loss values			

Quality Metrics

Average scores of different metrics for three datasets (on the subset of 10K data points) are shown in the following figure.

Each methods are run 10 times with different random initializations. The scores are calculated for each resulting visualizations. The mean values are reported in the bar chart, the standard deviations are shown in the black error bars. Note that the scores are stable and thus the standard deviations are small.



The two quality metrics AUC[R_{NX}] and AUC[G_{NN}] are detailed here:

Neighborhood preserving	KNN gain
$Q_{NX}(k) = \frac{1}{Nk} \sum_{i=1}^n \nu_i^k \cap \rho_i^k $ $R_{NX}(k) = \frac{(N-1)Q_{NX}(k) - k}{N-1-k}$ $AUC[R_{NX}] = \left(\sum_{k=1}^{N-2} \frac{R_{NX}(k)}{k} \right) / \left(\sum_{k=1}^{N-2} \frac{1}{k} \right)$	$\hat{\rho}_i^k = \{j \in \rho_i^k : c_j = c_i\} , \quad \hat{\nu}_i^k = \{j \in \nu_i^k : c_j = c_i\} $ $G_{NN}(k) = \frac{1}{Nk} \sum_{i=1}^n (\hat{\rho}_i^k - \hat{\nu}_i^k)$ $AUC[G_{NN}] = \left(\sum_{k=1}^{N-2} \frac{G_{NN}(k)}{k} \right) / \left(\sum_{k=1}^{N-2} \frac{1}{k} \right)$

Effect of Relative Margin

With different values of the relative margin, we obtain different visualizations. Large margin forces the groups to be more concentrated. (In the paper, we use the relative margin of 0.5.)

