Input: Number of random restarts I, number of hill-climbing steps J, length of random walks T.
Initialize set $U = \{\}$
for $i := 1, \ldots, I$ **do**
  Initialize $x =$ random point in S.
  $x' =$ SMOOTHED-SIMULATED-ANNEALING(x, T, J)
  $U = U \cup \{x'\}$
end for
Return the best element in U

Figure 4: The Optimization Method with Random Restarts

Input: Starting point $x \in S$, number of hill-climbing steps J, length of random walks T.
for $j := 1, \ldots, J$ **do**
  Perform a random walk of length T from x. Let y be the stopping state.
  Let $\widehat{f}(x, s) = f(y)$
  Let u be a neighbor of x chosen uniformly at random.
  Perform a random walk of length T from u. Let v be the stopping state.
  Let $\widehat{f}(u, s) = f(v)$.
  if $f(v) \leq f(y)$ **then**
    Update $x = u$
  else
    Temperature $\tau = 1 - j/J$
    With probability $e^{(f(y)-f(v))/\tau}$, update $x = u$
  end if
end for
Return x

Figure 5: The SMOOTHED-SIMULATED-ANNEALING Subroutine

A More Details for Experiments

First, we explain the graph-based nearest neighbor search. Let N be a positive integer. Let $\mathcal{G}$ be a proximity graph constructed on dataset $\mathcal{V}$ in an offline phase, i.e. $\mathcal{V}$ is the set of nodes of $\mathcal{G}$, and each point in $\mathcal{V}$ is connected to its N nearest neighbors with respect to some distance metric $\ell : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. In our experiments, we use the Euclidean metric. Given the graph $\mathcal{G}$ and the query point $y \in \mathbb{R}^d$, the problem is reduced to minimizing function $f = \ell(., y)$ over a graph $\mathcal{G}$. The algorithm is shown in Figures 4 and 5. The SGNN continues for a fixed number of iterations. In our experiments, we run the simulated annealing procedure for $\log n$ rounds, where n is the size of the training set. See Figure 5 for a pseudo-code. Finally, the SGNN runs the simulated annealing procedure several times and returns the best outcome of these runs. The resulting algorithm with random restarts is shown in Figure 4. The above algorithm returns an approximate nearest neighbor point. To find K nearest neighbors for $K > 1$, we simply return the best K elements in the last line in Figures 4. We use $K = 50$ approximate nearest neighbors to predict a class for each given query. We construct a directed graph $\mathcal{G}$ by connecting each node to its $N = 30$ closest nodes in Euclidean distance. For smoothing, we tried random walks of length $T = 1$ and $T = 2$. This means that, to evaluate a node, we run a random walk of length T from that node and return the observed value at the stopping point as an estimate of the value of the node. This operation smoothens the function, and generally improves the performance. The SGNN method with $T = 1$ is denoted by SGNN(1), and SGNN with $T = 0$, i.e. pure simulated annealing on the graph, is denoted by SGNN(0). For the SGNN algorithm, the number of rounds is $J = \log(\text{training size})$ in each restart.

The graph based nearest neighbor search has been studied by Arya and Mount (1993), Brito et al. (1997), Eppstein et al. (1997), Miller et al. (1997), Plaku and Kavraki (2007), Chen et al. (2009), Connor and Kumar (2010), Dong et al. (2011), Hajebi et al. (2011), Wang et al. (2012). In the worst case, construction of the proximity graph has complexity $O(n^2)$, but this is an offline operation. Choice of N impacts the prediction accuracy and computation complexity; smaller N means

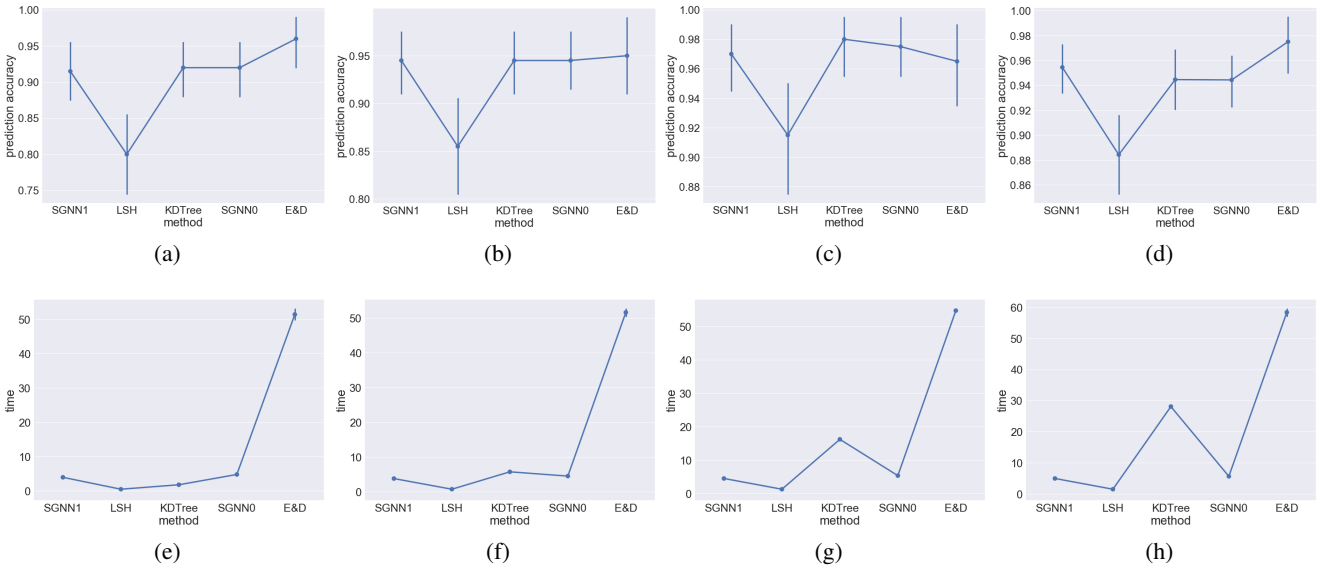


Figure 6: Prediction accuracy and running time of different methods on COIL-100 dataset as the size of training set increases. (a,e) Using 25% of training data. (b,f) Using 50% of training data. (c,g) Using 75% of training data. (d,h) Using 100% of training data.

lighter training phase computation, and heavier test phase computation (as we need more random restarts to achieve a certain prediction accuracy). Having a very large N will also make the test phase computation heavy.

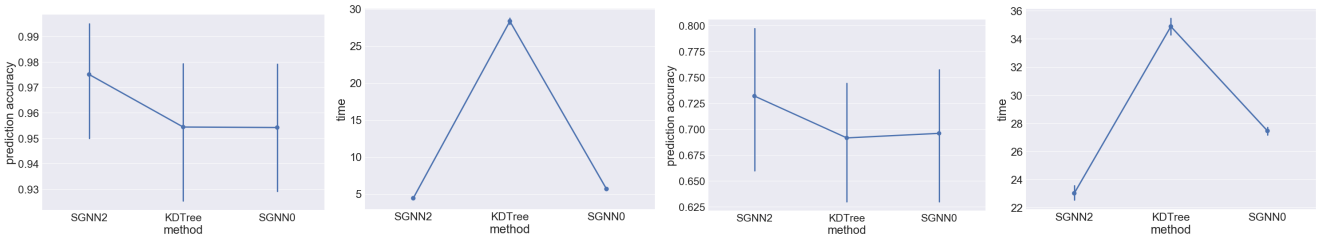
We used the MNIST and COIL-100 datasets, that are standard datasets for image classification. The MNIST dataset is a black and white image dataset, consisting of 60000 training images and 10000 test images in 10 classes. Each image is 28×28 pixels. The COIL-100 dataset is a colored image dataset, consisting of 100 objects, and 72 images of each object at every 5x angle. Each image is 128×128 pixels, We use 80% of images for training and 20% of images for testing.

For LSH and KDTree algorithms, we use the implemented methods in the scikit- learn library with the following parameters. For LSH, we use LSHForest with min hash match=4, #candidates=50, #estimators=50, #neighbors=50, radius=1.0, radius cutoff ratio=0.9. For KDTree, we use leaf size=1 and K=50, meaning that indices of 50 closest neighbors are returned. The KDTree method always significantly outperforms LSH. For SGNN, we pick the number of restarts so that all methods have similar prediction accuracy.

Figure 6 (a-d) shows the accuracy of different methods on different portions of COIL-100 dataset. As the size of training set increases, the prediction accuracy of all methods improve. Figure 6 (e-h) shows that the test phase runtime of the SGNN method has a more modest growth for larger datasets. In contrast, KDTree becomes much slower for larger training datasets. When using all training data, the proposed method has roughly the same accuracy, while having less than 50% of the test phase runtime of KDTree. Using the exact nearest neighbor search, we get the following prediction accuracy results (the error bands are 95% bootstrapped confidence intervals): with full data, accuracy is 0.955 ± 0.01; with 3/4 of data, accuracy is 0.951 ± 0.01; with 1/2 of data, accuracy is 0.943 ± 0.01; and with 1/4 of data, accuracy is 0.932 ± 0.01.

Next, we study how the performance of SGNN changes with the length of random walks. We choose $T = 2$ and compare different methods on the same datasets. The results are shown in Figure 7. The SGNN(2) method outperforms the competitors. Interestingly, SGNN(2) also outperforms the exact nearest neighbor algorithm on the MNIST dataset. This result might appear counter-intuitive, but we explain the result as follows. Given that we use a simple metric (Euclidean distance), the exact K-nearest neighbors are not necessarily appropriate candidates for making a prediction; Although the exact nearest neighbor algorithm finds the global minima, the neighbors of the global minima on the graph might have large values. On the other hand, the SGNN(2) method finds points that have small values and also have neighbors with small values. This stability acts as an implicit regularization in the SGNN(2) algorithm, leading to an improved performance.

These results show the advantages of using graph-based nearest neighbor algorithms; as the size of training set increases, the proposed method is much faster than KDTree.



(a) (b) (c) (d)

Figure 7: Prediction accuracy and running time of the SGNN method with random walks of length two (a) Accuracy on MNIST dataset using 100% of training data. (b) Running time on MNIST dataset using 100% of training data. (c) Accuracy on COIL-100 dataset using 100% of training data. (d) Running time on COIL-100 dataset using 100% of training data.