ECE 8527: Final

Christopher Campbell
Department of Electrical and Computer Engineering, Temple University
christopher.campbell@temple.edu

Introduction: The objective of this final project was to design multiple algorithms to classify a given dataset. The data had 2 variants, one with 2 dimensions and one with 5 dimensions. For each variant there was a train/dev/eval dataset, with 10000/2000/2000 and 100000/10000/10000 samples for the 2D and 5D versions, respectively. A neural network and non-neural network approach are considered.

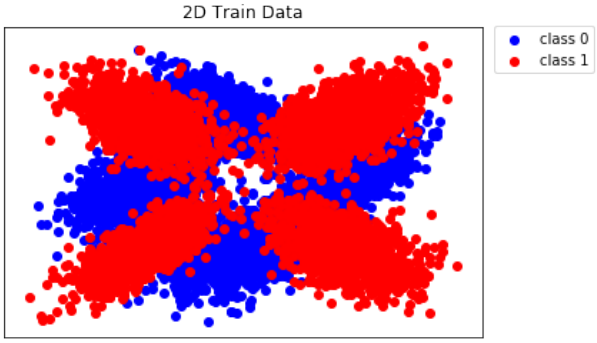

Figure 1: Plot of the 2D training data

k-Nearest Neighbors (kNN): The idea behind knn [1] is that by comparing all the points in the train/dev/eval sets against the points in the training dataset, it should be possible to use the k closest points in the training data to each test point to determine the class assignment of each test point. This algorithm seemed like a viable option because the data was a set of points in Euclidean space and there wasn't any special temporal or long-distance spatial relationships between points so the labeled points in the immediate vicinity should be enough for reasonably good performance.

This algorithm was implemented in Python. An aggregate set of the train data and the dataset under test (i.e. train/dev/eval) was created and then the pairwise distance between all points in the aggregate data was computed, and then recast into a square matrix. The portion of that matrix representing the distances between the dataset under test and the training data was then extracted, then for each test point all the distances to the training test points were sorted, and the top k (the results reported were obtained using k=16) were selected. Once the k nearest points were found, the class assignments for each of those points was found from the list of class labels, and then a majority vote determined the class assignment of each test point.

Semi-Supervised Generative Adversarial Network (GAN): The GAN [2] is an unsupervised neural network architecture which attempts to generate samples which could have plausibly been drawn from a particular dataset. The way this is accomplished is by creating two networks: a discriminator network and a generator network. The discriminator network takes in a sample (or batch of samples) and classifies the samples as whether or not they came from the real data or if they were generated from the generator network, which takes a random noise vector as an input and outputs a piece of synthetic data.

In order to use this for a supervised problem, one method proposed in [3] is to change the output of the generator to, instead of doing a binary classification, distinguish between all the classes of the real data, and also have one extra class which represents the synthetic data. This way, a batch of data given to the

generator consists of real labeled data, as well as synthetic data (with no class assignment). Hence, this is a semi-supervised GAN.

As is the case with a regular GAN, the semi-supervised GAN is trained by optimizing 2 different loss functions: one for each network. [2] suggests a schedule of alternately training the discriminator and generator networks (for this architecture the discriminator network was trained for 5 steps consecutively, then the generator network was trained for one step, per batch). The discriminator in this case behaves like a classifier and can be trained by minimizing the cross entropy between the network predictions and the true class labels. The generator is trained by taking the generated fake data and feeding it into the discriminator network and then taking the cross entropy of the discriminator output and the assignment corresponding to all-real data [4]. For example, if '0' represents that the discriminator predicted any of the real classes and '1' represents that the discriminator predicted that it was synthetic data, the goal of the generator network should be to output synthetic data that results in all '0' outputs from the discriminator, so it can be trained by minimizing the cross entropy between the discriminator output with the synthetic input, and the appropriately-sized zero vector.

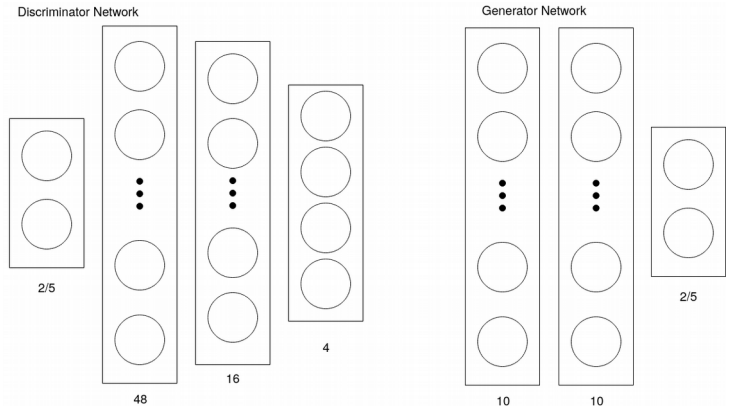

Figure 2: Semi-supervised GAN architecture

Results: Table 1 summarizes the results of the different algorithms on the 2D and 5D datasets. Since the data looks like it was sampled from a relatively low-dimension probability distribution (i.e. there should not be a much larger presence of outliers in the eval data that didn't appear in the training data) the results were fairly consistent for each algorithm between the train/dev/eval sets. In all cases, the k nearest neighbor approach performed better than the semi supervised GAN.

Conclusions: One likely explanation for the limited effectiveness of the GAN in this scenario is that looking at Figure 1, since these are points drawn from a distribution and the generator essentially begins by outputting random data, it is very likely that those random points could already plausibly be members of the training data since the feature space is so small, which would significantly advantage the generator over the discriminator. This is also corroborated by the fact that the average generator loss was much lower during training than the discriminator loss.

	2D Data			5D Data		
Algorithm	**Train**	**Dev Test**	**Eval**	**Train**	**Dev Test**	**Eval**
kNN	07.69%	07.85%	08.50%	38.49%	38.93%	39.16%
Semi-supervised GAN	09.57%	08.80%	09.25%	40.11%	40.20%	40.07%

Table 1. Algorithm performance comparison for the 2D and 5D datasets

References

[1] R. O. Duda, P. E. Hart, and D. G. Stork, “Pattern classification.” John Wiley & Sons, 2000.

[2] I. J. Goodfellow *et al.*, “Generative Adversarial Networks,” Jun. 2014.

[3] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved Techniques for Training GANs,” Jun. 2016.

[4] “Deep Convolutional Generative Adversarial Network.” [Online]. Available: https://www.tensorflow.org/tutorials/generative/dcgan.