Autoencoded k-means: Using jointly trained autoencoders for robust clustering

Colin Samplawski
College of Information and Computer Sciences
University of Massachusetts Amherst
csamplawski@cs.umass.edu

Abstract

The classical methods for k-means clustering can perform poorly in high dimensional feature spaces or in the presence of noise. This motivates the addition of an autoencoder architecture to simultaneously reduce the dimensionality of the data and remove noise. In this work, I provide a model which jointly trains a denoising autoencoder and optimizes the k-means objective. I see that my model is capable of producing higher quality clusterings than the standard algorithm.

1 Introduction

The workhorse method for data clustering is k-means. Given a dataset of N points $\{x_1, x_2, \ldots, x_N\}$, the algorithm finds k cluster centers $C = \{c_1, c_2, \ldots, c_k\}$ which minimize the sum of the distances between each data point and the cluster center that it is closest to. This can be expressed as the following objective:

$$\mathcal{L} = \sum_{i=1}^{N} \min_{c_j \in C} ||x_i - c_j||^2 \tag{1}$$

Since k-means uses squared Euclidean distance, all the features of the data contribute equally to the distance calculation. This can be problematic in high dimensional features spaces since some features may be uninformative. Furthermore, it has been shown that k-means performs poorly in the presence of noise [1]. This motivates the addition of an autoencoder.

An autoencoder is a neural network which learns to reproduce its input. In the most basic setting, it consists of a single hidden layer followed by a linear output layer that is equal in size to the input. If we set the hidden layer size to be less than the input size, we can train the autoencoder to produce representations of the data in a smaller dimensional space. This implicitly performs feature selection on the data. The computations of the autoencoder are as follows:

$$\begin{aligned} h &= f(xW_1 + b_1) \\ x' &= hW_2 + b_2 \end{aligned} \tag{2}$$

where f is a nonlinearity, W_1, W_2 are the weights of the network, and b_1, b_2 are the biases. We call x' the reconstruction of the input x and h the hidden state. We train to minimize the reconstruction error, given by:

$$\sum_{i=1}^{N} ||x_i - x_i'||^2 \tag{3}$$

Autoencoders have been shown to be an effective way to reduce dimensionality in an unsupervised way [2]. Furthermore, they can be extended to also denoise data. A denoising autoencoder has nearly the same architecture as above, only a noisy version of the input is given at training time.

The reconstruction error is then calculated using the clean version of the input. Denoising autoencoders have been shown to be an effective method of denoising data [2]. In this work I provide an architecture which jointly learns a denoising autoencoder and optimizes the k-means objective.

The rest of this paper is structured as follows: Section 2 discusses relevant previous work. Section 3 provides a short overview of cluster evaluation metrics. In Section 4 I describe the model architecture. Section 5 discusses experimental design and results. Section 6 discusses possible future work. Finally, Section 7 concludes.

2 Related Work

The traditional method of handling uninformative features is weighted k-means [3]. By assigning a weight to each feature, uninformative features won't play as large of a role in the cluster distance calculation, which can lead to improved performance. Unfortunately, this is not usually feasible since it requires domain knowledge at the feature level, which often doesn't exist. Furthermore, it does little in the face of noisy data.

The use of autoencoders to improve clustering performance has a reasonable body of prior work. Most relevant to this work is the work done by Xie et al. [4]. They jointly trained a deep autoencoder and optimized a clustering objective to improve clustering of textual and image data. However, unlike my work, they used gradient descent to optimize KL-divergence for clustering.

Tian et al. used autoencoders to improve the performance of graph clustering [5]. Coates et al. showed that autoencoders can be an effective method of extracting features from image data [6]. They also demonstrated that k-means could successively cluster images, however, the two were not used together. Finally, Dilokthanakul et al. used variational autoencoders to improve clustering using Gaussian mixture models [7].

3 Cluster Evaluation

Evaluating clusterings is generally a more difficult task than the case of classification. Fortunately, the datasets I am using have ground truth labels, so we can use metrics which take advantage of that information. I used three commonly used cluster performance metrics to evaluate the clusterings produced by my model. These three were chosen because they share many proprieties. Namely, they all have range [0,1], and they output 0 in the case of random assignments and 1 in the case of perfect clustering. Also, all three are insensitive to the number of clusters used, unlike other metrics, such as cluster purity. All three metrics were calculated using implementations found in scikit-learn's metric package.[1] I now provide a brief summary of each metric. Let D be a dataset of N points, C be clusters of points from D, and L be the ground truth labels.

3.1 V-Measure

The V-measure (validity measure) is an entropy based clustering metric which measures a clusterings trade-off between homogeneity (h) and completeness (c). A cluster is homogeneous if it contains points that all are of the same class. A clustering is complete if points from the same class are assigned to the same cluster. They are defined as follows:

$$h = \begin{cases} 1 & \text{if } H(L,C) = 0 \\ 1 - \frac{H(L|C)}{H(L)} & \text{else} \end{cases} \qquad c = \begin{cases} 1 & \text{if } H(C,L) = 0 \\ 1 - \frac{H(C|L)}{H(C)} & \text{else} \end{cases} \tag{4}$$

where H is information entropy. Note that these components often are opposing, increasing one usually decreases the other. V-measure is then the harmonic mean of homogeneity and completeness:

$$\frac{2 \cdot c \cdot h}{c + h} \tag{5}$$

In this way, it is somewhat analogous to the F1 measure commonly used in information extraction. A full analysis of V-Measure can be found in [8].

[1] `http://scikit-learn.org/stable/modules/model_evaluation.html`

Table 1: Comparison of k-means implementations. We compare scikit-learn's MiniBatchKMeans method[2] and my gradient k-means implementation. We see that my implementation results in higher quality clusterings for both datasets (Section 5.1).

	MNIST		EMNIST	
Score	scikit-learn	Gradient k-means	scikit-learn	Gradient k-means
V-Measure	0.514	0.533	0.466	0.467
NMI	0.387	0.441	0.368	0.372
Rand Score	0.138	0.338	0.089	0.185

3.2 Normalized Mutual Information

Normalized Mutual information (NMI) is an information theoretic metric which is defined as follows:

$$\frac{I(C;L)}{H(C)+H(L)} \tag{6}$$

where $I(C;L)$ is the mutual information between C and L, and H is again entropy. An in depth discussion of this and other information theoretic metrics can be found in [9].

3.3 Rand Score

The Rand score, named after William M. Rand, measures the similarity of two partitions of a dataset. The Rand score is defined as:

$$\frac{a+b}{\binom{N}{2}} \tag{7}$$

where a is the number of pairs of points that are in the same subset in C and in the same subset in L, and b is the number of pairs of points that are in different subsets in C and in different subsets in L. Note that $\binom{N}{2}$ is the total number of pairs of points in D. A full analysis of this metric can be found in [10].

4 Model

4.1 Gradient k-means

Most implementations of k-means work without the use of gradient methods, usually by using Lloyd's algorithm [11]. However, a gradient descent version of k-means is fairly simple to derive. Following the derivation provided by Bottou and Bengio [12], we frame k-means as the following minimization problem:

$$\min \mathcal{L} = \min \sum_{i=1}^{N} \frac{1}{2} ||x_i - c_{x_i}||^2 \tag{8}$$

where c_{x_i} is the closest cluster center to data point x_i. This then gives the following gradient:

$$\frac{\partial \mathcal{L}}{\partial c_j} = \sum_{i=1}^{N} \begin{cases} x_i - c_j & \text{if } c_j = c_{x_i} \\ 0 & \text{else} \end{cases} \tag{9}$$

With the gradient in hand, we can use a gradient descent algorithm to perform k-means clustering. In fact, Sculley has shown that a mini-batched version of this approach converges faster than any other algorithm when working in a large data regime [13]. We will call this model gradient k-means. As a baseline for my model, I implemented mini-batched gradient k-means in Tensorflow, which did not exist elsewhere to the best of my knowledge. Table 1 shows that it generates clusterings of higher quality than the algorithm provided by scikit-learn.[2]

[2] `http://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html`

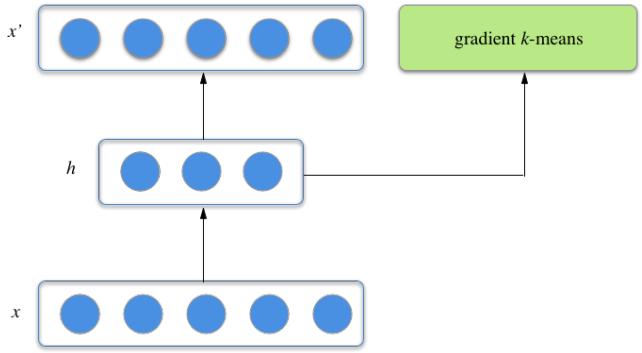

Figure 1: The autoencoded k-means architecture.

Figure 2: Sample images. MNIST (upper) consists of handwritten digits 0-9 and EMNIST (lower) consists of handwritten letters a-z and A-Z. Both consist of 28x28 centered gray-scale images.

4.2 Autoencoded k-means

Since gradient k-means produces gradients, we can backpropagate the error signal from the k-means objective through a neural architecture. I have a designed a model which consists of an autoencoder whose hidden state is given as input to gradient k-means (Figure 1). We will refer to this model as autoencoded k-means. During training, the error signals from both the autoencoder reconstruction and gradient k-means is backpropagated jointly through the hidden state. In this way, we are not only learning a smaller representation for the input, we are biasing that representation to be well suited for clustering. The objective for the entire architecture is:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{F}||x_i - x_i'||^2 + \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}||h_i - c_{h_i}||^2 \tag{10}$$

where F is the number of features.

5 Experiments

5.1 Datasets

To test my model I used two image datasets. I chose to use image data because they have a large number of features, even for simple images, and noise can be easily added artificially. In particular, I ran my experiments on the MNIST [14] and EMNIST [15] datasets (Figure 2). MNIST is a classical dataset in image processing which consists of 60,000 handwritten digits displayed in 28x28 gray-scale images. EMNIST (extended MNIST) is a recent dataset which consists of images in the same format as MNIST, but includes letters A-Z as well as digits. For this work, I only considered the 124,800 images from the dataset which were of a letter. Letters in the dataset can be upper or lowercase, however, there is only 26 classes. That is, images of "a" and "A" are considered the same class.

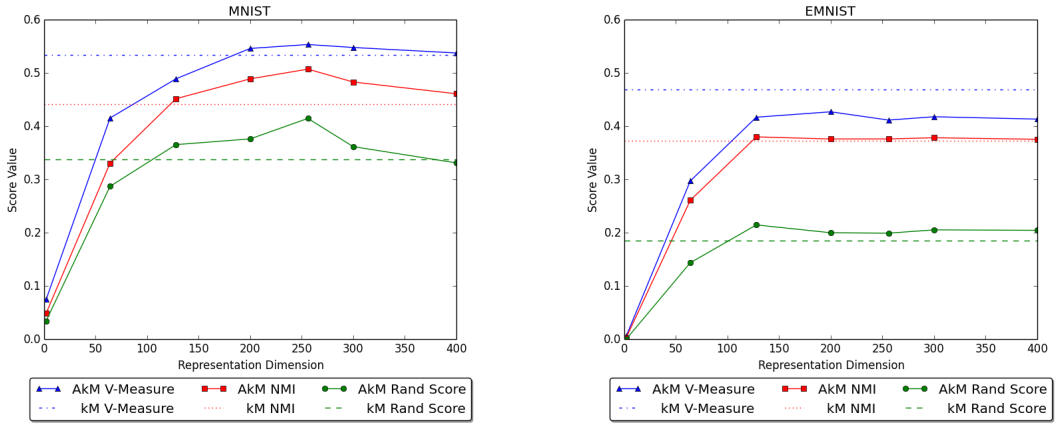

Figure 3: Effect of representation dimension on performance. Solid lines show score values from the clusterings generated by autoencoded k-means (AkM). Horizontal dashed lines show score values of clusterings produced by gradient k-means (kM) without any autoencoder.

5.2 Implementation Details

All my experiments and models were developed using Tensorflow[3] (my code is publicly available[4]). For both datasets a batch size of 100 was used. Optimization was performed via the Adam optimizer [16] using the default parameters of the Tensorflow implementation[5]. The number of clusters used was determined to be 10 times the number of classes, so 100 for MNIST and 260 for EMNIST. The autoencoder used a tanh nonlinearity. The matrix of cluster centers was initialized as the pseudo-identity matrix, and the weights and biases of the autoencoder was initialized using the Xavier initialization method [17]. Training took place on a GPU.

5.3 Effect of Representation Dimension

We first experiment by changing the size of the representation produced by the autoencoder. For a range of values (2 at the lowest, 400 at the highest), I trained autoencoded k-means to learn a representation of that size and simultaneously cluster the data in that smaller dimensional feature space. We hope that the autoenocder will select the most informative features, resulting in better clustering. The results of these experiments can be found in Figure 3

We see a significant decrease in performance on both datasets for small dimensional values (2 and 64) when compared to k-means. This isn't surprising since it seems unlikely that a 784 feature space could be reasonably represented in 2. However, once a reasonable dimension size is reached, we see an increase in performance on both datasets. MNIST especially shows sizable gains over k-means for a dimension of size 256. Interestingly, as the dimension size increases past 256, we see a drop in performance on the MNIST data. This suggests that the autoencoder may be preserving unimportant features in its representation, leading to worse clustering performance.

From these experiments we see that EMNIST is significantly harder dataset to cluster. This isn't surprising given the increased complexity of the data. Autoencoded k-means is able to gain a small performance gain over k-means, however it scores consistently worse on the V-Measure metric, across all choices of dimension size. This is due to the fact that the completeness component of the V-Measure on the EMNIST data was consistently low. This means that points from the same class aren't being assigned to the same cluster. This may be due to the fact that EMNIST treats upper and lowercase letters as the same class. It seems likely that autoencoded k-means is learning to place "a" in a different cluster than "A", resulting in lower completeness.

[3] https://www.tensorflow.org
[4] https://github.com/colinski/AutoencodedKMeans
[5] https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer

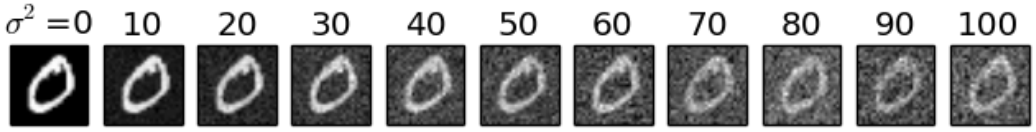

Figure 4: Example of noisy images. As the variance (σ^2) of the noise distribution increases, the images are considerably more corrupted.

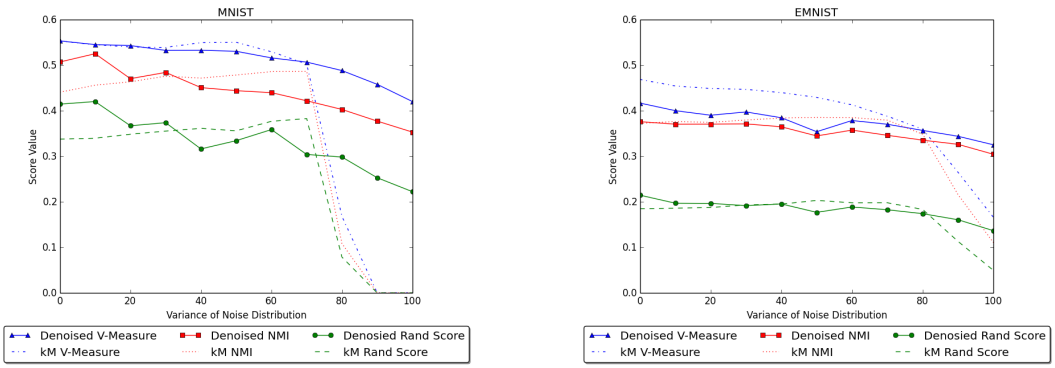

Figure 5: Effect of noisy data on performance. Solid lines show score values from the clusterings generated by autoencoded k-means (with denoising) on noisy data. Dashed lines show score values of clusterings produced by gradient k-means (kM) without any autoencoder on the same noisy data.

5.4 Effect of Noisy Data

To evaluate the capacity of my model to denoise data, I artificially added noise to the images. I applied a technique known as Gaussian whitening. For each pixel, a value was drawn from a zero mean Gaussian distribution with variance σ^2. This value was added to the pixel's value and then clipped so it falls in a valid range for the image (0-255). Increasing the variance results in nosier images. Figure 4 shows examples of such images. For values of σ^2 from 0 to 100, I generated a noisy dataset and trained autoencoded k-means to denoise this data. The best performing dimension size (256 for MNIST and 128 for EMNIST) from the previous experiments were used. Therefore autoencoded k-means is in fact simultaneously reducing the dimensionality of the data and denoising it. Figure 5 shows that results of these experiments.

We first discuss the performance of k-means on noisy data. From my experiments we see that it performs poorly once the variance of the noise is ≥ 70. Both datasets are fairly robust to noise until that point, after which the performance sharply drops off. In fact, we actually see a small increase in performance for small variance noise, suggesting that the noise may be introducing a minor regularization effect. Performance on the MNIST dataset especially suffers from the addition of noise, with a drop to 0 on all metrics in the worst case, meaning that k-means is essentially producing random clusters in that scenario.

We see that the addition of a denoising autoencoder significantly improves the clustering performance on noisy data. My experiments show that increasing the variance of the noise distribution still hurts performance, however, the decline is much less intense. We see that for noisy data with variance ≥ 70, the performance is significantly improved. In fact, on the EMNIST dataset the performance is nearly decoupled from the amount of noise, suggesting that the autoencoder is able to completely learn the noise distribution.

6 Future Work

The results presented here lay a reasonable groundwork for future work. The most straightforward thing to try next would be to test my model's performance on other datasets. In fact, I originally had planned to test on the CIFAR-10 color image dataset [18], but I discovered that the state-of-art

method used 4000 clusters, which was outside of my computational budget. Additionally, it would be interesting to test my model on non-image datasets, such as textual data.

Another area to pursue would be to experiment with the autoencoder architecture. The results presented here use a very basic autoencoder, but significantly more advanced models exist. The most straightforward change would be to implement a deep autoencoder which has many more hidden layers. Additionally, there are has been some work in using deconvolution operations which could be especially useful for the image datasets used here. In any case, it seems likely that a more sophisticated architecture could produce better lower dimensional representations, leading to better cluster performance.

7 Conclusion

In closing, k-means can perform poorly on datasets with many features or in the presence of noise. I designed and implemented autoencoded k-means, a neural architecture where jointly learns an autoencoder and optimizes the k-means objective. We see that on two image datasets, this new model performs better than k-means by clustering the data in a lower dimensional space. Furthermore, autoencoded k-means can successfully learn the noise distribution of noisy data leading to a denoising effect. This results in significantly improved performance in the presence of noise.

References

[1] S. Ben-David and N. Haghtalab, "Clustering in the presence of background noise," *Proceedings of the 31st International Conference on Machine Learning*, vol. 32, 2014.

[2] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[3] M. Ackerman, S. Ben-David, S. Brânzei, and D. Loker, "Weighted clustering," *Computing Research Repository*, 2011.

[4] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," *Proceedings of the 33rd International Conference on Machine Learning*, vol. 48, 2016.

[5] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering," *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pp. 1293–1299, 2014.

[6] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, pp. 215–223, 2011.

[7] N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan, "Deep unsupervised clustering with gaussian mixture variational autoencoders," *6th International Conference on Learning Representations*, 2017. Under Review.

[8] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, vol. 48, pp. 410–420, 2007.

[9] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *Journal of Machine Learning Research*, vol. 11, pp. 2837–2854, 2010.

[10] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[11] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[12] L. Bottou and Y. Bengio, "Convergence properties of the k-means algorithms," *Advances in Neural Information Processing Systems*, pp. 585–592, 1995.

[13] D. Sculley, "Web-scale k-means clustering," *Proceedings of the 19th International Conference on World Wide Web*, pp. 1177–1178, 2010.

[14] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998. `http://yann.lecun.com/exdb/mnist/`.

[15] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: an extension of mnist to handwritten letters," 2017. `https://www.nist.gov/itl/iad/image-group/emnist-dataset`.

[16] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[17] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[18] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. `https://www.cs.toronto.edu/~kriz/cifar.html`.