Hierarchical k-Means for Unsupervised Learning

Harry Gifford[*]

Carnegie Mellon University

Abstract

In this paper we investigate how to accelerate k-means based unsupervised learning algorithms with hierarchical k-means.

We show that hierarchical k-means significantly speeds up k-means based learning approaches in both the training and query phases at minimal cost to test accuracy. This speedup allows for much larger numbers of centroids to be used, which in turn leads to much better learning.

We evaluate the quality of the learned features by performing image classification on natural images.

1 Introduction

Unsupervised feature learning is becoming increasingly popular and important, as it offers relatively painless, but powerful methods to automatically generate features that can be used in many standard machine learning and vision domains [5].

k-means has been successfully applied to unsupervised learning and is particularly noteworthy for its simple implementation, fast training and lookup speed. Unfortunately it suffers from the problem that a very large number of centroids are required to generate good features which can offset its speed advantage [2].

In this paper we propose a simple change to the usual k-means pipeline which vastly improves the speed of algorithms using k-means for unsupervised learning.

1.1 k-Means Pipeline

We now describe a basic pipeline for learning features from unlabaled data, first described by [3]. We will describe the algorithm on images, although this can be generalized to other forms of data, such as speech.

[*]hgifford@cmu.edu

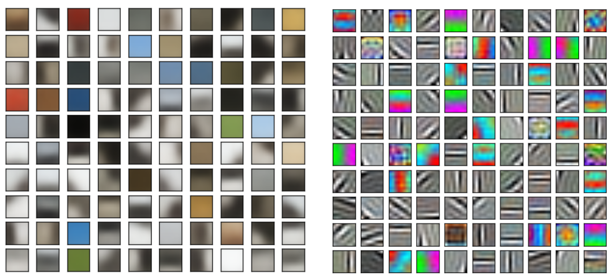

Figure 1: 100 centroids trained on natural images from the CIFAR-10 database. Top are centroids learned from unwhitened patches and above are centroids learned from whitened patches. Notice the different oriented Gabor like structures that are apparent. Also notice that edges between opposing colors begin to form, such as the red-blue edges as well as the pink-green edges.

Assume we are given input data and an algorithm parameterized by k - the number of centroids and s - the patch size. Then the algorithm proceeds as follows:

1. Extract a specified number of patches from the input data, for example by splitting up the patches.

2. Whiten the data. In order to do this we subtract off the mean of the image and subtract the standard deviation. We then apply a principle component transform in order to make neighboring pixels as independent as possible. This is very similar to what the center-surround cells in the retina do and tends to emphasize edge detail.

   This has the nice property that it forces our learning algorithm to find underlying structure quickly, instead of focusing on discovering the obvious dependence between neighboring pixels.

   The difference between whitened and unwhitened images can be seen in Figure 1. The centroids in from whitened images focus only on the 'interesting' parts of the image structure - namely the parts of the image that make good features.

In the algorithm we describe we used PCA and ZCA to transform the data. There is no difference in terms of their performance and their ability to make out structure, but it will affect visualization.

3. Perform k-means clustering on the whitened patches. This gives us a set of bases, which act as 'representatives' for the dataset we are analysing. Many variants of the standard k-means algorithm exist - mainly defining how to initialize k-means [1], how to measure distance between data points and what constitutes a centroid [4]. A common distance metric is the cosine distance used in spherical k-means [7] [10].

4. In order to get a good descriptor to describe a region of the original input image, we select a region of the image, divide it into patches that are the same dimension as the centroids C, and then for each region i, we create a feature vector, $\mathbf{v}$ where

$$p_j = \frac{1}{|C|}\left[\sum_{k=1}^{|C|} ||i - C_k||_2\right] - ||i - C_j||_2$$

and

$$v_j = \max(0, p_j)$$

In other words, each region is assigned a descriptor which indicates the relative activation of each centroid with that region.

Negative activation is disallowed. This in fact acts as a sparsity constraint and pools some of the features. This means we end up with a sparse feature representation, which is amenable to learning.

5. Finally, we can take these feature vectors we have just learned and apply the whole pipeline again, in order to learn higher level features. In general we will want to enforce some kind of sparsity constraint, which is relatively easy to derive from the feature vectors. This allows us to throw out particular collinearity in the learning process.

2 Hierarchical k-means algorithm

We incorporate a simple, natural hierarchical k-means algorithm described in [9] into our pipeline in order to accelerate generation of centroids. We briefly describe the algorithm below:

In hierarchical k-means we pick some k to be the branching factor. This defines the number of clusters

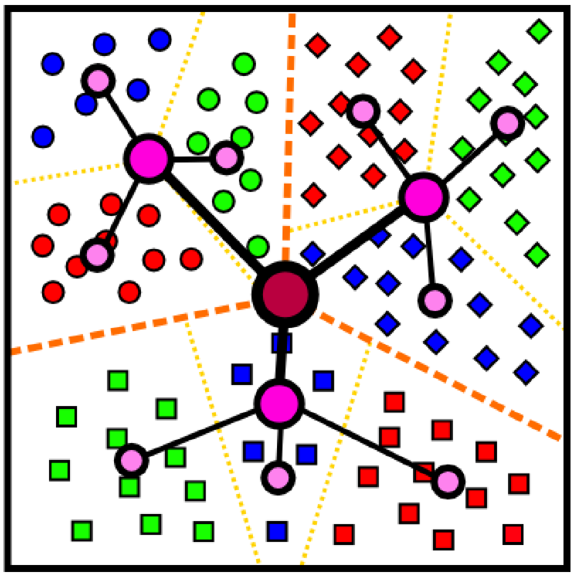

Figure 2: An example of $k = 3$ means hierarchical clustering. First sort the points into clusters and then recursively cluster each clustered set of points.

at each level of the clustering hierarchy. We then cluster the set of points into k clusters using a standard k-means algorithm. Finally, we recursively cluster each sub-cluster until we hit some small fixed number of points.

A visualization of heirarchical k-means is shown in Figure 2

2.1 Hierarchical k-means for unsupervised learning

We use this hierarchical k-means to accelerate the clustering, feature vector construction and lookup. However, there are a few questions that we might have about the validity of this hierarchical approach:

- This algorithm is not guaranteed to generate the same quality of clusters as a traditional k-means algorithm. How can we know that we haven't sacrificed good centroids for speed? Later we show through experimentation on classification that this does not seem to be the case.

- What happens to clusters that contain few data points? As can be seen in Figure 3, if there are not enough data points in a given cluster then we end up generating degenerate centroids that will never be activated. While degenerate centroids do not directly hurt classification performance, they do waste CPU cycles as well as sample patches. Therefore we would like to maintain a relatively uniform number of data points for each centroid.

  Fortunately, with hierarchical clustering we can

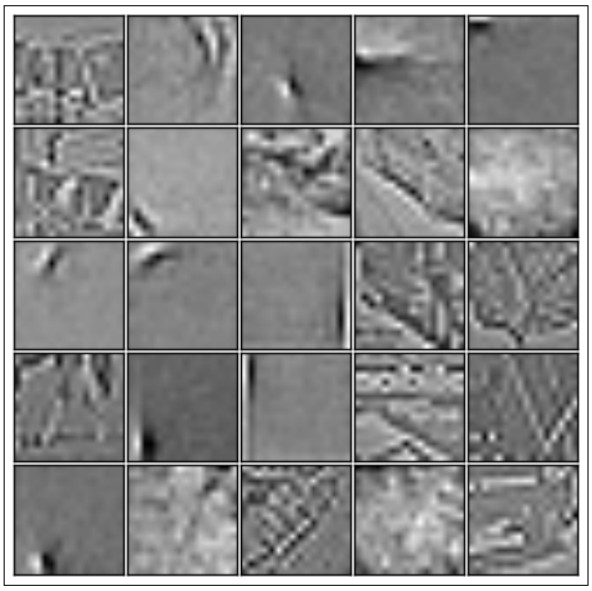

Figure 3: Centroids with insufficient data points tend to make poor features. The singleton points are evident by the specific images and not edge-like Gabor filters.

Table 1: CIFAR-10 Full dataset. Performance of our hierarchical k-means approach compared to other published results.

Architecture	**Accuracy / %**	**Speed / s**
1 Layer K-means [2]	78.3%	475
3 K-means [2]	82.0%	3061
Conv. DBN [8]	80.49%	$>$ 200000
$\star$ 1 Layer HK-means	77.8%	49
$\star$ **3 Layer HK-means**	**82.89%**	**221**

easy merge clusters with few points into neighboring clusters, by simply cutting off the tree at the appropriate depth. Thus, singleton clusters are not an issue, unless very large patches are used (which suffers from curse of dimensionality) or not enough data exists.

3 Evaluation

In order to evaluate the performance of our k-means approach we built a single-layer k-means classification network and applied it to the CIFAR-10 natural images database.

We took the pipeline described in 1.1 and used a single layer k-means representation, along with a three layer k-means representation. In the single layer representation we achieved 78.3% performance on full CIFAR-10 at best and with a multi-layer clustering we achieved 82.89% performance, which is competitive with the standard k-means result, but at a fraction of the speed.

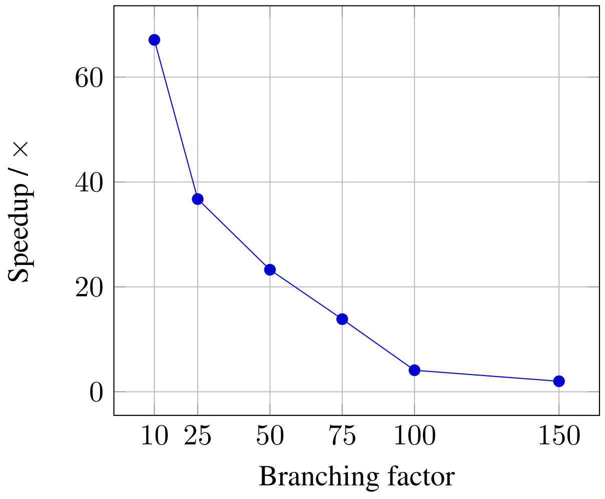

Figure 4: Speedup of depth 3 hierarchical k-means on full CIFAR-10 (average of 3 runs). As the branching factor increases the speedup compared to the standard k-means implementation drops significantly.

Table 1 shows how our results compare with relevant published results, mostly those of k-means networks. All of the tests were run on GHC 5208 cluster laptops, using NumPy/SciPy. The convolutional deep belief network results were not recorded by me, but were run on fast desktop GPUs.

This table highlights the success of k-means networks when compared to standard DBNs. Our own results, highlighted by the $\star$ are competitive with the other k-means results, which shows that it is possible to get major speedups with hierarchical k-means.

We performed n-way cross validation in order to tune and test parameters, and the final results used everything except the test set to compute accuracy. This is in line with the results from [2] and [8].

3.1 Branching Factor

The main controllable parameter by a user of our system is the branching factor. It turns out that the branching factor acts as an excellent slider for converting between speed and accuracy.

As can be seen from Figure 5 an increase in branching factor leads to an increase in accuracy. This is unsurprising, since if we take the branching factor to infinity then our algorithm degenerates to standard k-means. What is surprising is that the branching factor is a relatively good predictor for the accuracy on benchmarks we tested.

Also interesting is the relationship between speedup, with respect to a baseline k-means and the branching factor. Here it is less surprising that there is an obvious relationship between the branching factor and speedup: as the branching factor increases, the

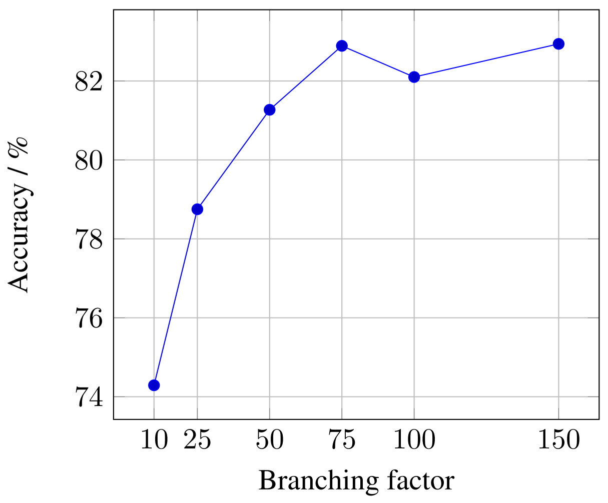

Figure 5: Performance of depth 3 hierarchical k-means on full CIFAR-10 (average of 3 runs). As the branching factor of the k-means tree increases, accuracy also tends to increase. However, this is at the expense of time. As the branching factor gets greater, the closer the algorithm tends towards vanilla k-means.

benefit of the hierarchy begins to diminish as we begin to have to check more leaves and more point-point distance checks. At the limit this will simply give us standard k-means again.

3.2 Classification

In order to perform object classification from the resulting features generated by our model we can simply pass our result through a simple one-versus-all classifier. The classifier was not of any particular interest to us in this case, and both an SVM and softmax regression classifier gave similar results, generally with the SVM slightly outperforming the softmax regression, but at the expense of slow classification. Since we were only interested in relative speedup we opted to use an SVM classifier for all tests.

4 Future Work

It would be interesting to see if we could combine Dropout [6] or some other distributed, genetic style deep network with hierarchical k-means. Dropout and its derivatives achieve some of the best classification rates to date, but such systems suffer from reasonably slow training and lookup rates, as compared to clustering based approaches.

Indeed dropout could potentially be added to the system when choosing which centroids to drop for generating the next level in the k-means deep network.

It would also be interesting to compare our hierarchical k-means approach with hierarchies of other clustering techniques. k-means makes some very strong assumptions about the distribution of the clusters, of which we have not thoroughly verified. k-means requires that all clusters be of roughly the same size and they should be roughly spherical. This is not necessarily a valid assumption, especially at higher levels of a deep network where features can get complicated.

5 Conclusion

In this paper we have described a way to speed up k-means based unsupervised feature learning. We achieve good speed-up, at minimal loss of accuracy - of which is more than made up for by the improved speed. We also show empirically that our method allows for a gradual variation in speed versus accuracy.

References

[1] Arthur, D., and Vassilvitskii, S. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2007), SODA '07, Society for Industrial and Applied Mathematics, pp. 1027–1035.

[2] Coates, A., and Ng, A. Y. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade (2nd ed.)*. 2012, pp. 561–580.

[3] Coates, A., Ng, A. Y., and Lee, H. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS* (2011), pp. 215–223.

[4] Hartigan, J. A., and Wong, M. A. A K-means clustering algorithm. *Applied Statistics 28* (1979), 100–108.

[5] Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput. 18*, 7 (July 2006), 1527–1554.

[6] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR abs/1207.0580* (2012).

[7] Hornik, K., Feinerer, I., Kober, M., and Buchta, C. Spherical k-means clustering. *Journal of Statistical Software 50*, 10 (9 2012), 1–22.

[8] Krizhevsky, A. Convolutional deep belief networks on cifar-10, 2010.

[9] Nister, D., and Stewenius, H. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2* (Washington, DC, USA, 2006), CVPR '06, IEEE Computer Society, pp. 2161–2168.

[10] Zhong, S. Efficient online spherical k-means clustering. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on* (2005), vol. 5, pp. 3180–3185 vol. 5.