Learning K-way D-dimensional Discrete Embedding for Hierarchical Data Visualization and Retrieval

Xiaoyuan Liang[1,2] , **Martin Renqiang Min**[1] , **Hongyu Guo**[3] and **Guiling Wang**[2]
[1]NEC Labs America - Princeton
[2]New Jersey Institute of Technology
[3]National Research Council Canada
xl367@njit.edu, renqiang@nec-labs.com, hongyu.guo@nrc-cnrc.gc.ca, gwang@njit.edu

Abstract

Traditional embedding approaches associate a real-valued embedding vector with each symbol or data point, which is equivalent to applying a linear transformation to "one-hot" encoding of discrete symbols or data objects. Despite simplicity, these methods generate storage-inefficient representations and fail to effectively encode the internal semantic structure of data, especially when the number of symbols or data points and the dimensionality of the real-valued embedding vectors are large. In this paper, we propose a regularized autoencoder framework to learn compact Hierarchical K-way D-dimensional (HKD) discrete embedding of symbols or data points, aiming at capturing essential semantic structures of data. Experimental results on synthetic and real-world datasets show that our proposed HKD embedding can effectively reveal the semantic structure of data via hierarchical data visualization and greatly reduce the search space of nearest neighbor retrieval while preserving high accuracy.

1 Introduction

Data embedding methods have been successfully deployed in many real-world applications, including unsupervised and supervised data visualization [Maaten and Hinton, 2008; Min *et al.*, 2010; Min *et al.*, 2017], natural language understanding [Mikolov *et al.*, 2013; Pennington *et al.*, 2014; Shen *et al.*, 2018], computer vision [Frome *et al.*, 2013], information retrieval [Clinchant and Perronnin, 2013], bioinformatics analysis [Du *et al.*, 2018], and many others.

These embedding strategies, however, fail to sufficiently reveal essential semantic structures of the data in the embedded space. Typically, these methods associate a real-valued embedding vector with each symbol or data point, which is equivalent to applying a linear transformation to "one-hot" encoding of discrete symbols or data points. Despite their simplicity, these methods are incapable of encoding the internal semantic structure of data, failing to effectively preserve the interplay of the symbols/data points in the embedded space, such as the hierarchical relationship of the symbols or data samples. Hierarchical clusters of data will allow one to know how the symbols/data points are grouped and how lower layer groups form upper layer clusters. Such structural information is, therefore, critical for data understanding and fast information retrieval.

To cope with the aforementioned challenge, in this paper, we propose a regularized autoencoder framework for data embedding. Our approach is capable of capturing essential semantic structures of the data, thus leading to both hierarchical data visualization/exploration and efficient nearest neighbor retrieval. Our method builds on the success of the recent K-way D-dimensional discrete encoding [Chen *et al.*, 2018; Shu and Nakayama, 2018]. These discrete encoding algorithms encode, through deep neural networks, data points with discrete codes, thus being able to significantly reduce the storage space when compared to real-valued embedding. In this paper, we aim at enforcing the discrete codes to have structural information: different bits of a code are used to identify their relationships with other data points. In detail, we here leverage a regularized autoencoder to learn compact hierarchical K-way D-dimensional discrete embedding of symbols or data points. We employ an autoencoder framework with a discrete embedding layer regularized by a stochastic exemplar-centered neighborhood preserving loss, in which we combine different dimensions of a discrete code vector using exponentially decaying weights to achieve hierarchical K-way D-dimensional embedding (HKD). Consequently, the HKD embedding codes have a tree structure, where similar symbols tend to have the same codes in front bits while the back codes are different from each other to separate them. In addition, the autoencoder is regularized to preserve exemplar-centered neighborhoods, resulting in embeddings with similar codes tightly close to each other.

Experimental results on synthetic and real-world datasets show that, our proposed HKD embedding can, in addition to storage efficiency, reveal the semantic structure of data via hierarchical data visualization and greatly reduce search space of nearest neighbor retrieval while preserving high accuracy.

To the best of our knowledge, we are the first to propose a method to learn hierarchical discrete embedding, thus enabling hierarchical data visualization and fast nearest neighbor retrieval in addition to embedding storage efficiency. These salient features make our embedding strategy particularly attractive in practice, where both the computation power and storage resources may not be abundant.

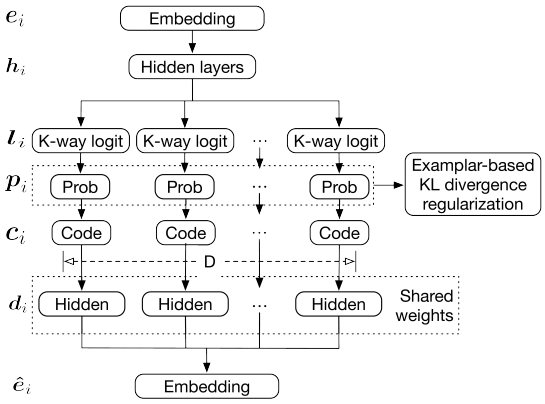


Figure 1: An illustration to the framework in the HKD method.

2 Hierarchical K-way D-dimensional Codes

Our Hierarchical K-way D-dimensional Discrete Embedding method (denoted as HKD encoding) leverages an autoencoder framework, where data features/embeddings are first encoded into the HKD embeddings which are then required to be able to decode (reconstruct) the original given data features/embeddings. Two novel components are devised to attain the goals of the HKD encoding, as follows. First, the HKD embedding codes have a tree structure, where similar symbols tend to have the same codes in front bits while the back codes are different from each other to separate them. Second, the autoencoder is regularized to preserve exemplar-centered neighborhoods, resulting in embeddings with similar codes tightly close to each other. We will discuss in detail the two novel components next.

2.1 HKD with an Autoencoder

The aim of the proposed HKD encoding method is to associate every symbol (data point) with a K-way D-dimensional discrete code. The whole process from embedding to discrete codes and verse vice are illustrated in Figures 1, and will be discussed in detail next.

Suppose, the discrete code for the i^{th} symbol (data point) is denoted by $\boldsymbol{c}_i = (\boldsymbol{c}_{i,1}, \boldsymbol{c}_{i,2}, ..., \boldsymbol{c}_{i,D})$, where $\boldsymbol{c}_{i,d}$ is a set of code bits with cardinality K. Consider $\boldsymbol{c}_{i,d}$ is a one-hot vector.

With this setting, given a symbol/data point i's embedding $\boldsymbol{e}_i$, the HKD first uses an encoder to learn its discrete codes $\boldsymbol{c}_i$. Next, a decoder in the HKD framework is then deployed to reconstruct the embedding $\hat{\boldsymbol{e}}_i$ to approach the real embedding $\boldsymbol{e}_i$ as much as possible. The encoding and decoding processes are formally formulated as follows.

HKD Encoding As illustrated in Figure 1, given the embedding $\boldsymbol{e}_i$, the hidden layers of the neural network first transfer the embedding into the K*D dimensional values $\boldsymbol{h}_i$:

$$\boldsymbol{h}_i = \mathbf{f}(\boldsymbol{M}\boldsymbol{e}_i), \tag{1}$$

where $\boldsymbol{M}$ denotes the weights of the hidden layer and $\mathbf{f}$ is a nonlinear activation function with multiple hidden layers.

Subsequently, the hidden layer' outputs are equally split into D partitions, where each partition has K values and each corresponds to exactly one dimension in the final discrete codes. Let $\boldsymbol{l}_{i,d}(d = 1, 2, ..., D)$ denote the d^{th} partition and $l_{j,i,d}$ denote the exact j^{th} value in the d^{th} partition, the code probabilities $\boldsymbol{p}_{i,d}$ are then calculated via a Softmax function on every partition, as follows.

$$\begin{aligned}\boldsymbol{p}_{i,d} &= \text{softmax}(\boldsymbol{l}_{i,d}) \\ &= \frac{\exp(\boldsymbol{l}_{i,d})}{\sum_{j=1}^{K}\exp(l_{j,i,d})}.\end{aligned} \tag{2}$$

The computed code probabilities $\boldsymbol{p}_{i,d}$ are used to form the discrete encodes by first passing through an argmax function and then representing by a one-hot vector:

$$\boldsymbol{c}_{i,d} = \text{one\_hot}\left(\arg\max_j \{p_{j,i,d}\}\right), j = 1, 2, ..., K. \tag{3}$$

To cope with the possible gap between the discrete codes and continuous variables, we use a temperature τ to approximate the discrete codes during training as in [Chen *et al.*, 2018],

$$\boldsymbol{c}_{i,d} \approx \text{softmax}\left(\frac{\boldsymbol{h}_{i,d}}{\tau}\right). \tag{4}$$

Similar techniques have been introduced in a Gumbel-Softmax trick [Jang *et al.*, 2016; Maddison *et al.*, 2016].

HKD Decoding After the encoding phase, a decoder is applied on the discrete codes generated by the encoder to reconstruct the original embeddings, as follows.

$$\hat{\boldsymbol{e}}_i = \sum_d w_d\, \mathbf{g}(Proj\,(\boldsymbol{A}_{i,d}\boldsymbol{c}_{i,d})), \tag{5}$$

where $\mathbf{g}(\cdot)$ is a sub-neural network with one or more hidden layers shared by all code dimensions as in Figure 1, $\boldsymbol{A_{i,d}}$ is the transformation weights for the i^{th} symbol in the d^{th} dimension, and w_d is the decayed weight for dimension d, which will be presented in the next subsection. $Proj(\boldsymbol{x})$ is a projection function, which is shown as follows,

$$Proj(\boldsymbol{x}) = \begin{cases} \frac{\boldsymbol{x}}{||\boldsymbol{x}||+\epsilon} & \text{if } ||\boldsymbol{x}|| \geq 1 \\ \boldsymbol{x} & \text{otherwise} \end{cases}. \tag{6}$$

The loss of the autoencoder here is to minimize the reconstruction error, which is defined by the mean squared error,

$$E = \frac{1}{n}\sum_i^n ||\hat{\boldsymbol{e_i}} - \boldsymbol{e_i}||^2, \tag{7}$$

where n denotes the number of data points in the dataset.

To further capture hierarchical semantic structures of the given data in the embedding space, we leverage a regularizer to force the model to consider data neighborhood information during the encoding process, which is discussed next.

2.2 Regularized Autoencoder

Our adopted regularization method aims at enabling the generated discrete KD codes to capture the semantic information in the data. To this end, we leverage the parametric t-distributed stochastic exemplar-centered embedding (pt-SEE) strategy [Min *et al.*, 2018], by extending pt-SEE to weight

different dimensions of the KD codes, to model the neighborhood information of the data points. Pt-SEE is an extension of t-SNE [Maaten and Hinton, 2008], which is an effective method to preserve the neighboring information when learning low-dimensional embeddings. pt-SEE significantly reduces the computational complexity of t-SNE. In specific, unlike t-SNE, pt-SEE does not compute pairwise neighboring probabilities. Instead, it chooses an enough number z exemplars to represent the distribution of raw data ($z \ll n$). The z exemplars can be formed in two ways, one is chosen by running some iterations of k-means on the raw data features/embeddings and the other is randomly chosen from the dataset. Promisingly, it has at most linear computational complexity with respect to the size of the whole dataset.

Formally, let $\boldsymbol{e}_j$ denote the raw embedding of the j^{th} exemplar chosen by k-means or random sampling, where $j \in [1, z]$. Same as before, $\boldsymbol{e}_i$ denotes the raw embedding/feature vector of the i^{th} data point. The neighboring probability in the raw data feature/embedding space is estimated by a Gaussian distribution.

$$\begin{aligned} p_{j|i} &= \frac{\exp(-d(\boldsymbol{e}_i, \boldsymbol{e}_j)/2\sigma_i^2)}{\sum_{k=1}^{z} \exp(-d(\boldsymbol{e}_i, \boldsymbol{e}_k)/2\sigma_i^2)}, \\ p_{j|i} &= \frac{p_{j|i}}{n}. \end{aligned} \tag{8}$$

Here, $d(\cdot)$ is a problem-specific distance function, for e.g., squared Euclidean distance or Poincaré distance, $i \in [1, n]$, and $j \in [1, z]$. Variance of the Gaussian distribution σ_i is set such that the perplexity of the conditional distribution $p_{j|i}$ equals to a user-specified perplexity u that can be interpreted as the expected number of nearest exemplars of data point i.

In our HKD encoding approach, because discrete codes cannot be directly used to calculate the neighboring probabilities, we use the code probabilities instead. In detail, to compute the neighboring probabilities in the code space, we use a t-distribution:

$$\begin{aligned} q_{j|i} &= \frac{(1 + d_{ij})^{-1}}{\sum_{i=1}^{n} \sum_{j=1}^{z} (1 + d_{ij})^{-1}}, \\ d_{ij} &= ||\boldsymbol{p}_i - \boldsymbol{p}_{e_j}||^2. \end{aligned} \tag{9}$$

where $\boldsymbol{p}_{e_j}$ denotes the code probabilities of the j^{th} exemplar.

In this way, the neighboring probabilities in the discrete code space are obtained. However, doing so, the KL divergence strategy simply treats every KD code equally. To attain a hierarchical coding, we consider a weighted version of distance calculation, which makes the front codes more important than the back codes, resulting in the following distance calculation formula,

$$d_{ij} = ||\boldsymbol{w} \circ (\boldsymbol{p}_i - \boldsymbol{p}_{e_j})||^2, \tag{10}$$

where $\circ$ denotes element-wise multiplication, and $\boldsymbol{w}$ is a weight vector with the same size as $\boldsymbol{p}_i$, in which all the weights for the d^{th} dimension of $\boldsymbol{p}_i$ have the same values w_d calculated by a decay function,

$$w_d = w_0 \exp(-\lambda d), \tag{11}$$

where w_0 is the initial starting weight. And the exemplar-based KL divergence is computed as follows,

$$KL = \sum_{i=1}^{n} \sum_{j=1}^{z} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}. \tag{12}$$

The final cost function of our HKD approach consists of two parts: the reconstruction error plus the exemplar-based KL divergence.

$$J = \alpha E + \beta KL, \tag{13}$$

where β is the penalty coefficient for the KL divergence regularization term.

3 Experiments

3.1 Settings and Baselines

We evaluate our proposed method in terms of its capability to hierarchically organize codes for speeding up nearest neighbor search and visualizing the semantic structure of the given data. To evaluate the method, we adopt the following two metrics. First, we examine the percentage of nearest neighbor entities that share the same code in the first N (out of D) codes and the reduced percentage of entities that are not neighbors and have different codes. Second, we visualize how the embedding codes correspond to the clusters of the given data.

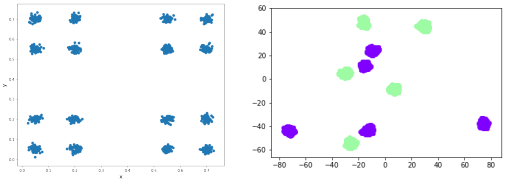
We compare our approach against the state-of-the-art KD code learning method [Shu and Nakayama, 2018] on three datasets: a synthetic dataset, Poincaré embedding [Nickel and Kiela, 2017] on WORDNET [Miller, 1995] and embedding on the CIFAR100 dataset [Krizhevsky, 2009]. We first use a synthetic dataset, aiming at better understanding the behavior of our HKD encoding schema. For the synthetic data, we generate the data using two-dimensional independent Gaussian distributions. There are in total 16 clusters, which is shown in Figure 2(a). Second, we evaluate our encoding schema using the Poincaré embedding, with the aim of investigating how our method can keep the hierarchy in the code space. Our last experiment uses the widely used CIFAR100 dataset.

In the evaluation, the examplars are made by two parts, one is the centers generated by k-means, and the other is centers combined with 10 nearest neighbors of every point. It means different points' examplars are different. The number of examplars are 10$\sim$20% of the number of training data points. We train our network using RMSprop [Tieleman and Hinton, 2012] with learning rate of 0.0001 and mini-batch of size 128. The whole model is built using PyTorch [Paszke *et al.*, 2017] and is trained using a GTX 1080 Ti GPU. The hyperparameters are chosen based on the validation data by comparing the magnitude of different loss terms. They are different in different models, which will be presented in the their experimental results. Due to space limit, here we only present the results on the test datasets, and the results on the training datasets are available in the supplementary material. Additional experimental results on the widely used CIFAR10 dataset [Krizhevsky, 2009], which is closely related to the CIFAR100 dataset, are included in the supplementary material[1].

[1]The supplementary materials are available at https://sites.google.com/view/hkd-supplementary/.

First N code (s)		1	2	3	4	5	6	7	8
KNN=5	Shu *et al.*	0.57	0.39	0.31	0.24	0.23	0.22	0.20	0.2
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.968
KNN=10	Shu *et al.*	0.52	0.31	0.23	0.14	0.13	0.12	0.11	0.10
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.962
KNN=15	Shu *et al.*	0.49	0.28	0.19	0.10	0.09	0.07	0.06	0.06
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.958
KNN=20	Shu *et al.*	0.43	0.19	0.15	0.10	0.06	0.05	0.05	0.05
	HKD	1.0	1.0	0.996	0.996	0.996	0.996	0.996	0.953
Search complexity reduction (%)		64.3	77.1	85.3	89.1	90.7	91.4	92.3	93.6

Table 1: Nearest neighbor preserving percentage by KD codes on the test set of the synthetic data



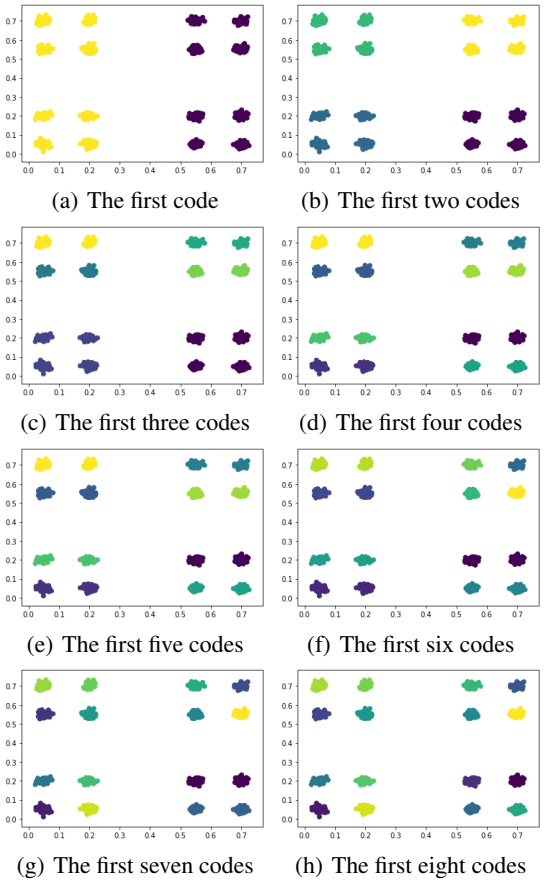
(a) Synthetic dataset (b) Two superclasses in CIFAR100

Figure 2: Two datasets in this paper

3.2 Results on the Synthetic Dataset

The synthetic dataset is split into two parts, training and test. The code performance on the test set is shown in Tables 1, while the performance on the training dataset is available in the supplementary material. In this dataset, the number of hidden units is set 20 considering there are only 2 dimensions in the raw data. The value of D is set 16 and K is 16. The value of α is 0.1. The two KL divergences' weights are both 1 and the perplexities in the two KL divergences are 5 and 11 respectively. We evaluate the hierarchical property by the accuracy of the same first N codes in every entity's nearest neighbors. The accuracy is defined by the percentage of same codes in the first N codes in the nearest neighbors. We also evaluate how smaller the search space can be reduced in finding nearest neighbors, which is defined by the average percentage of the number of entities that have the same first N codes to the total number of entities. This metric indicates that we can reduce the research space from the whole entities.

We also explore the data visualization using generated HKD codes. If two codes are exactly the same in the first N dimensions, they have the same color. The results are shown in Figure 3. The figures in Figure 3 clearly show that the HKD codes can form hierarchical clusters which are consistent with the known clusters of the synthetic data. For example, with the first layer code (sub-figure (a)), the embeddings are clustered into 2 clusters, which are consistent with the known super clusters as shown in Figure 2(a). When moving down the hierarchical structure of codes formed, more and more sub-clusters are formed by the HKD codes. As an example, on the second layer of the codes (sub-figure (b)), the two clusters from sub-figure (a) are perfectly divided into four clusters. These four clusters are further divided into 8 clusters when moving down one more layer of the HKD code hierarchy, as shown in sub-figure (c). When moving down to the either layer of the HKD codes (sub-figure (h)), 16 sub-clusters are formed.



(a) The first code (b) The first two codes

(c) The first three codes (d) The first four codes

(e) The first five codes (f) The first six codes

(g) The first seven codes (h) The first eight codes

Figure 3: Results in the synthetic dataset

We also did ablation study on the synthetic data by the following variants: no reconstruction error, no KL divergence, and no decay weights. The results in Tables 2 clearly indicate that the HKD model can accurately retrieval nearest neighbors via searching codes step by step along the hierarchical structure formed, significantly outperforming the variants. For example, our method with the first two layers of codes can cover 100% of all the data points, which are meaningfully better than about 50% achieved by the best variant.

First N code (s)		1	2	3	4	5	6	7	8
KNN=5	No reconstruction	0.51	0.19	0.16	0.12	0.08	0.07	0.06	0.06
	No KL divergence	0.17	0.12	0.07	0.06	0.05	0.05	0.05	0.05
	No decay weights	0.45	0.28	0.17	0.11	0.11	0.08	0.07	0.07
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.968
KNN=10	No reconstruction	0.55	0.23	0.19	0.13	0.09	0.08	0.08	0.08
	No KL divergence	0.20	0.14	0.08	0.06	0.06	0.06	0.05	0.05
	No decay weights	0.50	0.31	0.19	0.14	0.14	0.10	0.09	0.08
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.962
KNN=15	No reconstruction	0.55	0.23	0.19	0.13	0.10	0.08	0.08	0.08
	No KL divergence	0.20	0.14	0.09	0.06	0.06	0.06	0.05	0.05
	No decay weights	0.51	0.31	0.19	0.14	0.11	0.10	0.10	0.09
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.958
KNN=20	No reconstruction	0.53	0.25	0.20	0.12	0.10	0.08	0.08	0.08
	No KL divergence	0.21	0.14	0.10	0.07	0.06	0.06	0.05	0.05
	No decay weights	0.52	0.31	0.20	0.15	0.15	0.11	0.10	0.08
	HKD	1.0	1.0	0.996	0.996	0.996	0.996	0.996	0.953

Table 2: Ablation study results of HKD codes preserving nearest neighbors on the test set of the synthetic data

First N code (s)		1	2	3	4	5	6	7	8
KNN=5	Shu *et al.*	0.53	0.20	0.16	0.09	0.04	0.03	0.02	0.02
	HKD	0.99	0.99	0.98	0.97	0.97	0.96	0.95	0.93
KNN=10	Shu *et al.*	0.52	0.20	0.15	0.09	0.04	0.02	0.2	0.01
	HKD	0.99	0.98	0.97	0.96	0.95	0.94	0.92	0.90
KNN=15	Shu *et al.*	0.52	0.20	0.15	0.09	0.04	0.02	0.2	0.01
	HKD	0.99	0.96	0.95	0.94	0.94	0.92	0.92	0.89
KNN=20	Shu *et al.*	0.52	0.20	0.15	0.09	0.04	0.02	0.1	0.01
	HKD	0.98	0.96	0.93	0.92	0.90	0.89	0.87	0.86
Search complexity reduction (%)		76.8	88.0	88.8	91.6	93.2	94.0	94.8	95.4

Table 3: Nearest neighbor preserving percentage by KD codes on the test set of Poincaré embedding dataset

We can see that the reconstruction, examplar-based KL divergence and decay weights are important in our model.

3.3 Results on the Poincaré Embedding

Hierarchical embedding can be achieved by the Poincaré embedding method [Nickel and Kiela, 2017]. We choose the embedding generated by the Poincaré embedding method to train our hierarchical codes to explore whether our codes can maintain the hierarchical property.

In this task, we choose the mammal subtree in the WORDNET dataset. In the dataset, there are 1182 entities and 7724 semantic relations among them. We first train the Poincaré embedding with 10 dimensions per entity for 30 epochs. We then use the trained Poincaré embedding to encode the hierarchical KD codes. The code size is 16×16. We use 200 centroids from K-means and 100 samples that are randomly selected as the examplars. The value of α is 1. One KL divergence is used, and its weight β is 1 and perplexity is 10.

The results on the Poincaré embedding are shown in Table 3. We choose the first 8 codes and change the number of nearest neighbors between 5 and 20. We can see that our method can have about 100% of the 5 nearest neighbors having the same first codes and the percentage maintains over 90% after searching the first 8 codes. Meanwhile, the search space shrinks to only 5.6% of the whole dataset after looking for the first 6 codes.

To have better insights into the encoding codes, we also conduct a case study using the 'dog' category in the WORDNET dataset. Results are presented in Table 4. In this table, we use the entity 'dog.n.01' as the base and calculate other entities' distances to it. Comparing the distance and the codes in this table, we can see that, in our method, the nearer the entity is to the 'dog.n.01' entity, the more similar the codes are to those of the 'dog.n.01' entity. More specifically, when the distance is closer, the more codes at the first places are the same as those of the 'dog.n.01' entity. When the distance becomes further, the different codes may become more front. For example, the first code that is different between 'hunting.dog.n.01' and 'dog.n.01' is at the 7^{th} dimension while the first code that is different between 'whitetail prairie dog.n.01' and 'dog.n.01' is at the 2^{nd} dimension because 'whitetail prairie dog.n.01' is further to 'dog.n.01'. Meanwhile, we choose 'flying_fox.n.01' to show entities at different categories have totally different codes.

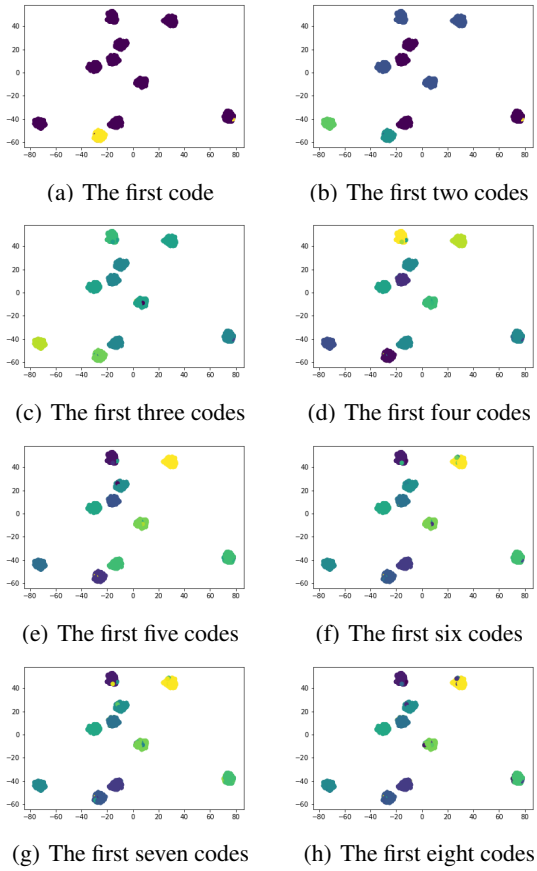
3.4 Results on the CIFAR100 Dataset

In the CIFAR100 dataset, there are 20 superclasses, and each superclass has 5 classes. This dataset has 50000 training images and 10000 test images in total. We use the wide resnet [Zagoruyko and Komodakis, 2016] to pre-train the dataset

Entity	Code	Distance to 'dog.n.01'
dog.n.01	[11 13 7 0 6 9 0 15 6 11 13 4 13 3 7 1]	0.00
hunting_dog.n.01	[11 13 7 0 6 9 11 15 6 4 13 4 13 3 7 1]	0.05
coondog.n.01	[11 13 7 0 6 9 13 5 2 11 13 2 13 3 7 1]	0.29
hearing_dog.n.01	[11 13 7 0 6 4 12 13 5 11 14 5 13 3 7 11]	0.43
crab-eating_dog.n.01	[11 13 7 0 6 15 5 11 6 10 13 0 13 3 8 9]	1.15
whitetail_prairie_dog.n.01	[11 12 7 0 13 11 4 1 5 10 13 5 9 12 2 14]	2.35
flying_fox.n.01	[7 13 7 8 13 1 1 1 6 10 14 4 6 3 0 10]	1.92

Table 4: Code case studies on Poincaré embedding dataset

based on class information to get every image's embedding. In this dataset, the number of hidden units is set 100. The value of D is set 16 and K is 16. The value of α is 0.01. In this one, we only use the KL divergence on the centers and nearest neighbors, whose weight is 1 and perplexity is 15. We choose two superclasses with 5 classes in each superclass, which are shown in Fig. 2(b) after t-SNE. Two kinds of colors denote two superclasses. We explore HKD codes in two superclasses via visualization as shown in Fig. 4. In these figures, dots in one color denote these points share the same codes in the first N dimensions. Through the first code, we can split one class from the two superclasses, which is shown in Fig. 4(a). When moving down the hierarchical structure of codes formed, more and more classes can be extracted, which are shown from Fig. 4(b) to Fig. 4(h). More specifically, our model can find smaller classes, which are grouped by only a few nearest neighbors, which is shown in Fig. 4(h). The retrieval results are available in the supplementary material.



(a) The first code (b) The first two codes

(c) The first three codes (d) The first four codes

(e) The first five codes (f) The first six codes

(g) The first seven codes (h) The first eight codes

Figure 4: Results in the CIFAR100 dataset

4 Related Work

Conventional methods considering the hierarchical structure in data retrieval are usually based on Huffman coding [Huffman, 1952]. But Huffman codes do not contain the semantic information. The works most related to ours are the KD code learning as introduced in [Shu and Nakayama, 2018; Chen *et al.*, 2018]. [Shu and Nakayama, 2018] provides an autoencoder framework to learn the KD codes while [Chen *et al.*, 2018] proposes an end-to-end KD code learning framework for different tasks. In contrast, we introduce a novel autoencoder regularizer to force the autoencoder for the hierarchical structures while generating discrete embedding codes.

Our autoencoder regularization schema builds on the pt-SEE strategy [Min *et al.*, 2018], where forming embeddings are encouraged to cluster around the prototypes or exemplars. Nevertheless, this method treats all embedding dimensions equally when computing their distance with the exemplars, thus discard the semantic structures of those data points. Our proposed approach here treats each embedding dimensions with different weights which correspond to their closeness with different exemplars. Those distances directly reflect their relationship with other data points.

5 Conclusion

In this paper, we propose a regularized autoencoder framework to generate hierarchical K-way D-dimensional codes from symbol/data point embeddings. The generated codes can significantly speed up the retrieval process by effectively reducing the search space. Such reduction is attained by making neighbor embeddings hold the same codes in the front dimensions. Experimental results on synthetic and real-world datasets show that our method can successfully build a hierarchical structure in the discrete KD codes, with over 90% nearest neighbors sharing the same codes in the first several dimensions. Our empirical studies also indicate that our approach reveals the semantic structure via hierarchical visualization.

References

[Chen *et al.*, 2018] Ting Chen, Martin Renqiang Min, and Yizhou Sun. Learning k-way d-dimensional discrete codes for compact embedding representations. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, June 2018.

[Clinchant and Perronnin, 2013] Stéphane Clinchant and Florent Perronnin. Aggregating continuous word embeddings for information retrieval. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality, CVSM@ACL 2013, Sofia, Bulgaria, August 9, 2013*, pages 100–109, 2013.

[Du *et al.*, 2018] Jingcheng Du, Peilin Jia, Yulin Dai, Cui Tao, Zhongming Zhao, and Degui Zhi. Gene2vec: Distributed representation of genes based on co-expression. 2018.

[Frome *et al.*, 2013] Andrea Frome, Gregory S. Corrado, Jonathon Shlens, Samy Bengio, Jeffrey Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. Devise: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems, Lake Tahoe, Nevada, United States.*, pages 2121–2129, 2013.

[Huffman, 1952] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.

[Jang *et al.*, 2016] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2016.

[Krizhevsky, 2009] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[Maaten and Hinton, 2008] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, November 2008.

[Maddison *et al.*, 2016] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712, 2016.

[Mikolov *et al.*, 2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[Miller, 1995] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[Min *et al.*, 2010] Martin Renqiang Min, Laurens van der Maaten, Zineng Yuan, Anthony Bonner, and Zhaolei Zhang. Deep supervised t-distributed embedding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010.

[Min *et al.*, 2017] Martin Renqiang Min, Hongyu Guo, and Dongjin Song. Exemplar-centered supervised shallow parametric data embedding. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 2479–2485. AAAI Press, 2017.

[Min *et al.*, 2018] Martin Renqiang Min, Hongyu Guo, and Dinghan Shen. Parametric t-distributed stochastic exemplar-centered embedding. In *Proceedings of the 2018 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 2018.

[Nickel and Kiela, 2017] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems (NIPS)*, pages 6338–6347, 2017.

[Paszke *et al.*, 2017] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

[Shen *et al.*, 2018] Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL, Melbourne, Australia*, pages 440–450, 2018.

[Shu and Nakayama, 2018] Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[Tieleman and Hinton, 2012] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[Zagoruyko and Komodakis, 2016] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.