Article

Design and Application of Deep Hash Embedding Algorithm with Fusion Entity Attribute Information(FADH)

XIAOLI HUANG [1], HAIBO CHEN [1] and ZHENG ZHANG [1]

[1] Research Institution of Signal Detection and Information Processing Technology, Xihua University, Chengdu, 610039, China

Abstract: Most machine learning and deep learning algorithms can only use low-dimensional data as input, but the data that must be processed in practical applications is diverse and irregular. There are two main problems with big dynamic data. (1) The size of the embedding table grows linearly with the vocabulary size, resulting in massive memory consumption. (2) For different newly added vocabularies. To solve these two problems, this paper proposes a novel embedding algorithm that can learn attribute associations with entities based on deep and hash algorithms. Taking movie data as an example, the encoding method and the specific flow of the algorithm are presented in detail, and the effect of dynamic reuse of data models is realized. Compared with the four existing embedding algorithms which can fuse entity attribute information, the deep hash embedding algorithm proposed in this paper has obvious optimization of time and space complexity.

Keywords: hash embedding; deep learning; attribute information; entity coding

0. Introduction

Machine learning and deep learning can learn and compute only in terms of mathematical vectors. Still, they contaminate discrete and character non-continuous digital features in the field of practical applications. The embedding algorithm[1][2][3][4], as an essential feature extraction algorithm that sparsely transforms high-dimensional features into low-dimensional dense features or maps discrete variables into low-dimensional space vectors, plays a vital role in machine learning and deep learning, usually as the first step of various deep learning tasks, whose expression effect and embedding performance often determine the results of subsequent tasks, such as the recommendation accuracy of the recommendation model and the semantic understanding effect of the NLP model. Since Google introduced the word2vec algorithm[5] in 2013,embedding technology has developed rapidly in natural language processing, recommendation algorithms,and graph data science. The word2vec algorithm proposes the CBOW and skip-gram models for computing word vectors. The training cost is low, but the expression effect is good, and the grammatical similarity and semantic similarity are used for evaluation. In 2016, Facebook proposed the fastText algorithm[6], which can significantly reduce training time. Microsoft presented the item2vec algorithm[7] based on the Word2vec method, which treats the item as a word and the user behavior as a sequence set and applies this algorithm to the item2item similarity calculation in the recommendation scenario. Youtube[8] uses a neural network to combine basic information about the user and train a user vector. The softmax weight in the whole recommendation model is output as the video embedding. In the same year, Google also proposed the wide&deep[9] model. The wide part provides interpretable features, and the deep part is used to discover combined higher-order features not found in the training set. In 2020, Google will again propose the DHE model[3] to solve dynamic data growth problems effectively, excessive data volume taking up too much storage and uneven data distribution. In graph embedding, the DeepWalk algorithm[10] in 2014 used random walks to sample nodes in the graph to simulate the corpus and adopted the idea of word2vec to learn the co-occurrence relationship of nodes so that similar nodes in the original graph were obtained. It is also close to its low-dimensional expression space. In 2017, Stanford College proposed the GraphSAGE algorithm[11]to extend the node neighbor

 doi:10.20944/preprints202212.0040.v1

aggregation function for an inductive learning task that can generalize unknown nodes and quickly generalize to the evolving graph structure to obtain embedding vectors. The FastRP algorithm[12][13] proposed by Chen et al. (2019) enables iterative computation of node embeddings while maintaining similarity between nodes and neighboring nodes and improving speed by constructing similarity matrices without explicitness. Among the algorithms that use hash[14][15][21] ideas, Hash2Vec proposes to apply feature hash to create word embeddings for natural language processing[17][18][19]. The algorithm requires no training to capture the semantics of words. It can easily handle a massive vocabulary in the order of millions—binary-oriented learning of hash embeddings[16] that can compress memory space arbitrarily while maintaining model accuracy. While there are many research results in the field of word embedding[23], graph embedding[20][22][24], and item embedding, most of these embeddings are based on the contextual relationship between entities. They do not integrate multiple feature attributes of entities. We develop a deep hash embedding algorithm that incorporates entity attribute information in this work. The time and space requirements of different embedding dimensions are tested, the distribution of space tests the clustering effect, and the mutual influence and characteristics of entities in their vector space are analyzed.

1. Introduction to related algorithms

The descriptions of the symbols used in the text are shown in Table 1.

Table 1. Symbol Description

Symbol definition	Meaning description
s	Feature word
e	Embedding vector
V	Feature value (representation of feature word)
W	Embedding table: one-hot coding with all feature words
N	number of datasets
k∈ $\mathbb{N}$	number of hash functions
n∈ $\mathbb{N}$	the size of the vocabulary
d∈ $\mathbb{N}$	Embedded dimension
m∈ $\mathbb{N}$	Size of the hash table (usually m<n)
H: V→[m]	Hash function: map feature words to 1,2,3,...m

1.1. The introduction of One-hot encoding

Data mining and storage usually use data in the form of natural language text, which makes it much easier for humans to read and understand. However, most machine learning algorithm models cannot be used directly. The solution is to convert text data into numeric data. For example, "sun", "moon", "earth",(e.g., 0 for the Sun, 1 for the Moon, and 2 for the Earth). The category values in the form of a set containing a set field of n categories can be easily viewed as a continuous numeric type with constant integers in the interval [0,n-1] One-hot coding can effectively solve this problem. One-hot encoding is for a set of n categories, with n-bit state flag bits to encode each class; each flag bit corresponds to one class, only one flag bit is one at any time, and the rest is 0. Defining the encoding function E with one-hot encoding E(s) for the feature word s is defined as formula (1):

$$\begin{cases} E(s) = b \in \{0,1\}^n \\ b_s = 1, b_j = 0 (j \neq s) \end{cases} \tag{1}$$

For example, "sun," "moon," "earth" are respectively depicted as shown in Figure 1:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

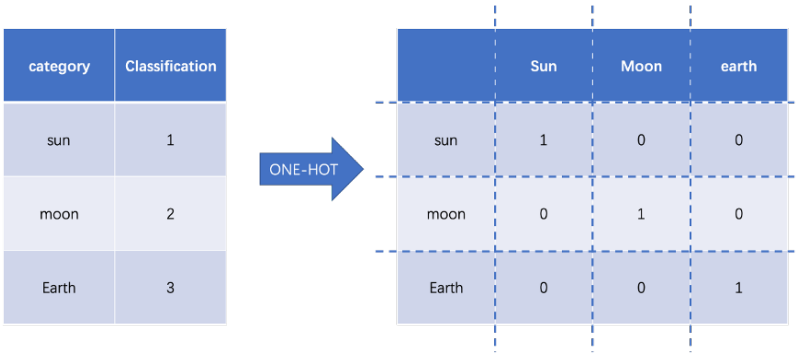

Figure 1. one-hot encoding example

The codes of these three categories of feature words are [1, 0, 0], [0, 1, 0], and [0, 0, 1]. If there are n categories in the set domain, the one-hot coding is a vector of length n. In each case, only 1 component of each vector is 1 and the others are 0. This determines the uniqueness of each category.

One-hot encoding is performed for each feature word with a set domain to obtain a vector representation of all feature words,which is essentially an embedding table composed of all feature word vectors.By querying the s-th row of the embedding table($W \in \mathbb{R}^{n* d}$), we obtain the representation vector e. One-hot encoding is essentially a neural network that can be viewed as a layer of unbiased terms. The corresponding embedding is expressed as $e = W^T b$[3],After processing data using one-hot coding for discrete data, downstream tasks such as similarity calculation and distance calculation are more useful, and applicability to related machine models is more appropriate,and availability is improved[1][2].

1.2. The Introduction of Hash Embedding Algorithms

Hash, also known as hashing, converts an arbitrary length input into an output of fixed length by a hashing algorithm, where the output value is the hash value. If a record in the hash table corresponds to the original input v of the algorithm, then v must be at the position of H(v). If the hash values of the two feature words are different, then the original inputs corresponding to the two hash values must be different. For large datasets, the vocabulary size can reach hundreds of thousands, and the model parameters can easily reach millions or even hundreds of millions. With a large number of vocabularies, there are problems with the size of the embedding. Due to the excellent properties of the hash function, all these problems can be solved by hash embedding. The use of hash embeddings has the following four advantages: (1) When using hash functions, it is not necessary to implement the creation of a dictionary that can handle dynamically growing vocabularies. (2) Hash embeddings have a mechanism that allows implicit vocabulary cleaning. (3) Hash embedding is implemented based on hash hashing, but this method adds a trainable mechanism based on hash hashing that effectively resolves hash collisions. (4) hash embedding usually reduces the parameters by orders of magnitude. Hash functions inevitably have their shortcomings: For a large amount of data, using a hash map for a large number of original inputs leads to hash value collisions, i.e., multiple initial values with the same hash value The value of the situation. When the hash value directly represents the original input, it is impossible to distinguish which initial input it is, affecting the model's impact. Hash embedding uses several different hash functions to represent the same initial value to reduce the possibility of collisions. The overall schematic representation of the algorithm can be seen in Figure 2:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

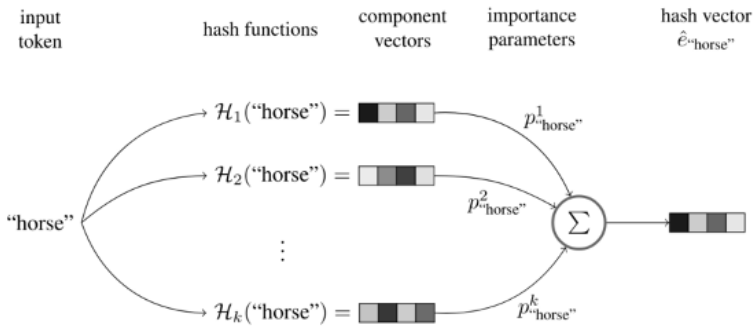

Figure 2. hash embedding

Steps of the algorithm: Step1: Use k different hash functions $\mathrm{H}_1, \ldots\ldots, \mathrm{H}_\mathrm{k}$ for the feature word s to be embedded and calculate k different hash values[4].

Step2: Combine the k components from Step1 into a weighted sum[3]:

$$\widehat{e_s} = \sum_{i=1}^{k} p_s^i H_i(s), \quad p_s = \left(p_s^1, \ldots\ldots, p_s^k\right)^T \in \mathbb{R}^k \tag{2}$$

Step3: Optional. The weighting parameters p_y of the feature words can be concatenated with $\hat{e}_s$ to pro e the final vector e.

The feature word s is converted into a complete representation of the hash code ($\oplus$ is the concatenation operator) according to formula (3)

$$\begin{cases} c_s = (\mathrm{H}_1(\mathrm{s}), \ldots\ldots, \mathrm{H}_\mathrm{k}(s))^T \\ p_s = \left(p_s^1, \ldots\ldots, p_s^k\right)^T \\ \widehat{e_s}^1 = {p_s}^T c_s \\ e_s^T = \widehat{e_s^T} \oplus p_s^T(\text{ optional }) \end{cases} \tag{3}$$

If step3 is not selected,$\hat{e}_s$ is the final vector representation, and if step 3 is selected, $e_S^T$is the final vector representation.

2. Embedding algorithm based on a deep hash fusion of entity information

As the amount of data increases and the content becomes richer, the entity description information becomes more comprehensive. Embedding algorithms are an area with extremely high requirements for expressive features that are both dynamic and highly complex. In practical applications, entities that belong to a particular category have category-attribute labels in the same domain. For example, a movie can be considered an entity, and its attribute labels can be divided into director, actor, genre, language, rating, etc.; a car can also be considered an entity, and its attribute labels can be divided into performance type, stage, engine capacity, power, color, etc. In this paper, we develop a deep hash embedding model that combines information about entity attributes to improve representation. The overall structure of the model is shown in Figure 3

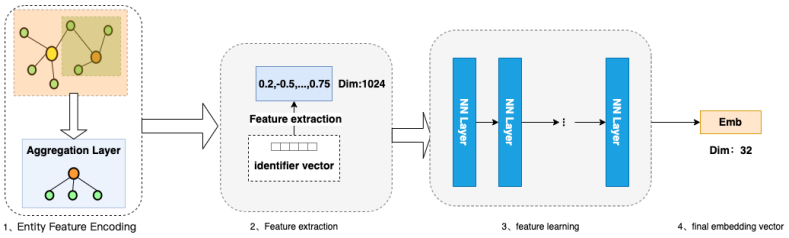

Figure 3. FADH algorithm model

The model structure is divided into three parts:

1. Entity Feature Encoding: First, in the first part, all features of the entity are fused and encoded to express the features of additional information in the vector representation

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

of the entity nodes, and after encoding, a splicing code $v_{\text{initial encoding}}$ of indefinite length is obtained;

2. Coded feature extraction: In the feature extraction in the second part, the extracted features are further processed, and k unique hash values are obtained as feature representations;
3. Using neural networks for feature learning: In the third part, we train a general deep neural network that inputs k different hash values encoded by each entity separately into the neural network, learns the inherent, implicit association between k features, and outputs the outputs in the desired dimension to obtain the embedding vector.

2.1. Entity Feature Encoding

A new encoding method is proposed: encoding entities based on domain attributes. First, after all attribute information in a class of entities, we count all attribute categories belonging to $|E|$ different fields and define E as the set of all fields, $E = F_1, \ldots\ldots, F_n$ where F_i is a domain, each domain contains m different features

$$F_i = \{f_1, \ldots\ldots, f_m\}, \quad F_i \in E \tag{4}$$

At the same time, each feature f_i belongs to only one domain F_i. The overall flow of the coding algorithm is shown in Figure 4:

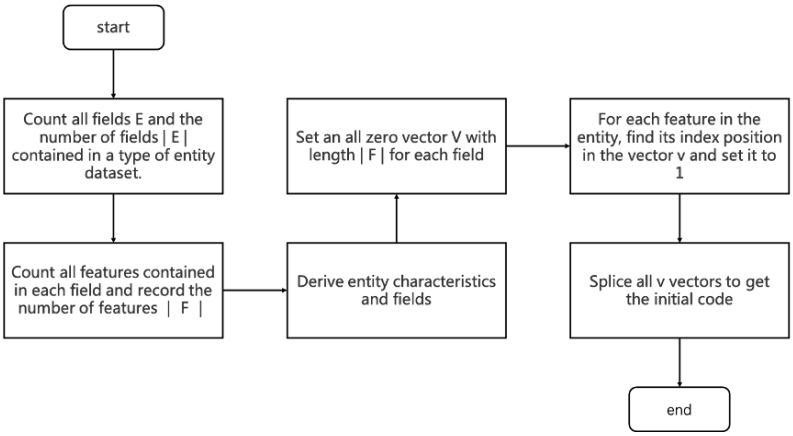

Figure 4. Coding Flowchart

Step 1: Count all domains E and the number of domains $|E|$ contained in a class of entity records.

Step 2:Count all the features F contained in each range, sorting them so that the order of F is fixed each time;and count the number of all features $|F|$.

Step 3:If the features in the domain are discrete variables, set a vector V of length $|F|$ containing all zeros for this domain. Continuous variables are not processed in this process for now and will be processed in the following 2.3 Feature Learning.

Step 4.:Further improvement based on one-hot coding: no longer use one coding for each tag, but one coding for a range. Set the encoding for the entity in a particular domain: determine the domain F_i to which each feature f_i of the entity belongs, find the position index of f_i in F_i, and set the value of the index in the vector V to change from 0 to 1.

$$V = \{0, 1\}^{|F|} \tag{5}$$

$$\begin{cases} \text{if: } F[\text{ index }] = f_i, V[\text{ index }] = 1 \\ \text{if: } F[\text{ index }] \neq f_i, V[\text{ index }] = 0 \end{cases} \tag{6}$$

Step5:Perform step 4 coding for each domain in the entity, and code to get the vector V_i of each domain.

$$e' = V_1 \| V_2 \| \cdots \| V_n \tag{7}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

Taking the movie entity as an example of coding, Figure. 5 is an attribute domain diagram of the attribute information contained in the movie, and Figure 6 shows the attribute information in each domain of The Shawshank Redemption.

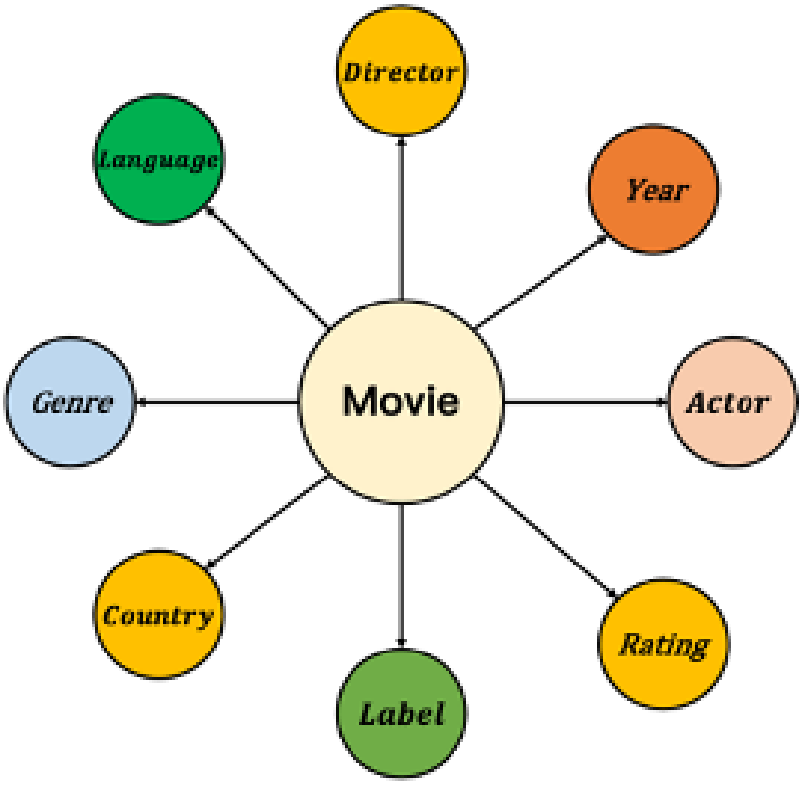

Figure 5. Domains contained in the movie entity

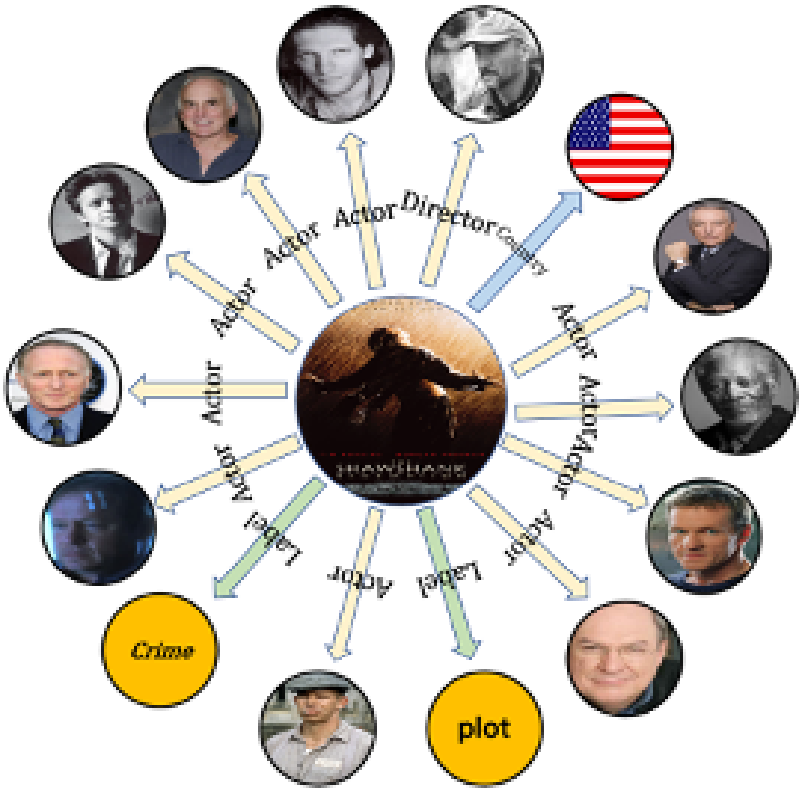

Figure 6. The movie example of The Shawshank Redemption

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

Throughout the movie dataset, all attribute information includes eight categories: Director, Year, Rating, Actor, Label, Country, Genre, and Language. The discrete data types are actor, director, label, country, and language. Five domains.

$E =$ Actor, Director, Label, Country, Language

First, the number of all features in each field is counted, and for each field, a vector of zeros is set whose length is equal to the number of features in the field. Sort all features, so they are ordered and fixed in sequence, with each bit representing a feature. If that feature occurs in the movie, set the 0 representing the job to 1. The initial code s is obtained by concatenating the representation vectors of all domains. A detailed coding example is shown in Figure 7.

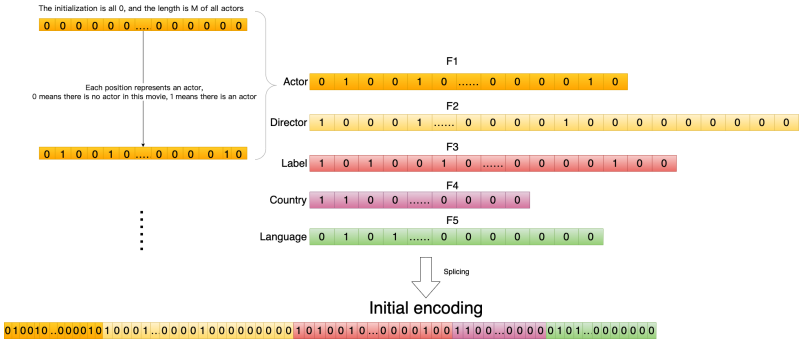

Figure 7. initial encoding

2.2. Extract features using hash functions

After the attribute information is merged, the initial encoding of the entity is the sum of the lengths of all fields, and the initial encoding is all represented by a string composed of 0s and 1s. Although the hashing method can effectively reduce the size of the code.even in the case of an ideal uniform distribution, the code of each entity in the dataset is uniformly distributed by [n/m], mapping a large number of codes simultaneously to a smaller one. the formula is as follows:

$$H(x) = ((ax + b) \bmod p) \bmod m \tag{8}$$

a and b are random integers, $b \neq 0$ and p is a large prime, much larger than m.Usually, $m \ll n$ is the problem of the embedding conflict. The same hash value may represent two or more initial values. This inevitably affects the performance of the model. Different entities have the same embedded representation, and the model cannot distinguish different feature values.

The solution uses multiple hash functions to extract features that collectively represent an entity to mitigate the conflict problem. The key idea is to concatenate the hash values produced by several different hash functions to reduce the possibility of collisions significantly. The vector encoded by the initial part is encoded with several hash functions, as shown in Figure 8, using encoding function $E_k \rightarrow H_k$ The eigenvalues are mapped to k dimensions using K general hash functions. $E'(s) = \left[H^1(s), \ldots\ldots, H^k(s)\right]$ Where $H : V \rightarrow [m]$, m has nothing to do with the size of the embedding table, just set m to a larger number, then the hash function can hash evenly on 1, 2, 3,..., m on. The hashed eigenvalues become k-dimensional features, but the integer E′(s) is unsuitable input for the neural network. Therefore, the encoding must be done by a suitable transformation.

$$E(s) = \text{Transform}(E'(s)) \tag{9}$$

The transformation function chooses a uniform distribution for simplicity and ease of operation.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

$$\text{Transform}(x) = 2\left(\frac{x-1}{m-1}\right) - 1 \tag{10}$$

No memory is required during this encoding process since all computations can be performed dynamically. This is also a nice feature of using multiple hashes, as we get cleaner high-dimensional encodings without increasing the amount of memory required.

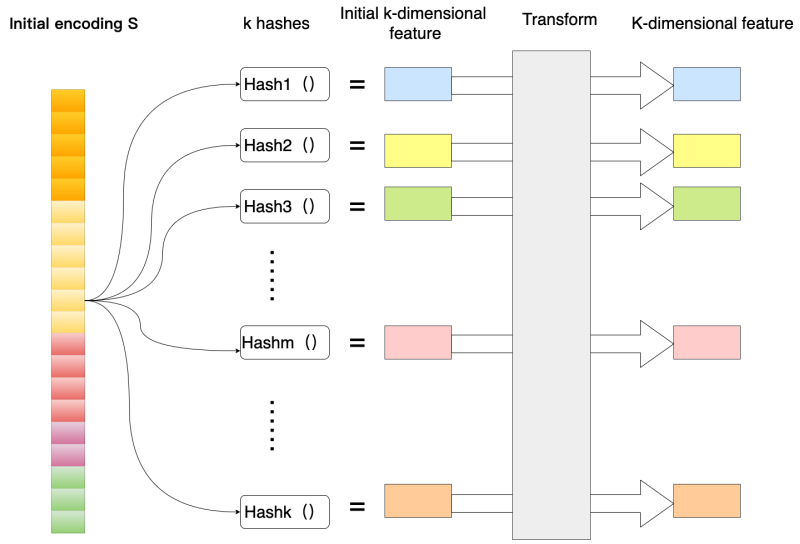

Figure 8. multiple hash encoding

2.3. feature learning

The third part uses a deep neural network to learn internal hidden features, fuses different attribute information of entities, and identifies specific internal associations. The k-dimensional integer feature after hashing is not suitable for query embedding, and the data must be mapped and transformed. The mapping process is very similar to the highly nonlinear feature transformation, where the input features are fixed, and it is not easy to learn their mutual influence. A deep neural network is a general function approximator, so a powerful deep neural network model is used to fit such complex transformations.

The attribute information of the entity includes discrete variables and continuous variables. In step 2, only the area of the entity belonging to the discrete variables is encoded. Some of the constant variables only need to perform simple normalization to use the neural network. The network learns features and outputs features. Simple linear normalization transforms continuous data into [0,1]. The formula is as follows:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{11}$$

This method realizes the proportional scaling of the original data, X is the original data,X_{max} is the maximum value of the original data set, and X_{min} is the minimum value of the original data set.

After the year and score in the movie entity attributes are normalized by the formula, they are input to the feedback neural network together with the k-dimensional features extracted in step 2 to learn the intrinsic features, as shown in Figure 9.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

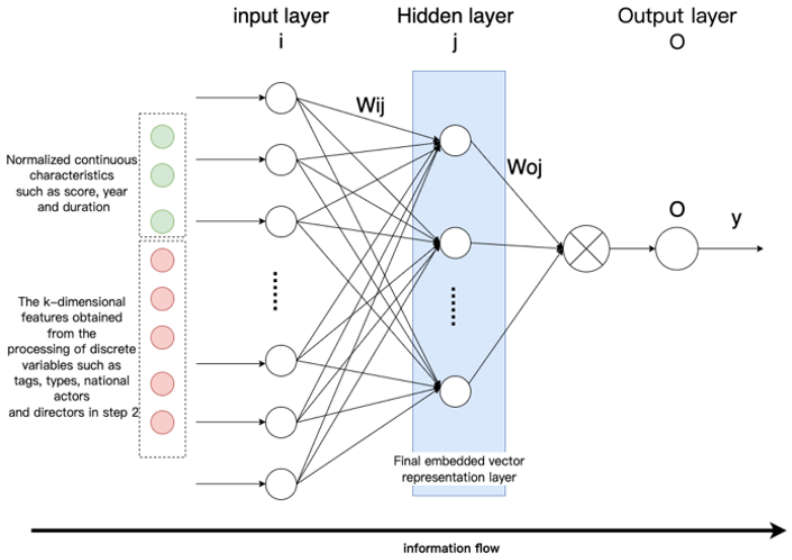

Figure 9. Schematic diagram of feedback neural network feature learning

Set an all-1 vector of length |e| as a control, and use the Jaccard similarity to calculate the similarity values as the label value of the feedback neural network as supervision. This process uses a single hidden layer feedforward neural network, the input data vector is $(x1, x2, \ldots, x_n)^T$, and the output y is data. Taking the network structure where the hidden layer is set to 1 layer as an example, the principle of multi-layer hidden layers is the same. The input and output of the hidden layer nodes are as follows:

$$n_{ji} = \sum_{i=1}^{n} \omega_{ji} * x_{n-i} + a_i \tag{12}$$

$$x_j = f_j(\text{net}_{ji}) \tag{13}$$

The input and output of the output layer node are respectively the formula:

$$n^{net}{}_{jo} = \sum_{i=1}^{n} \omega_{oj} * x_{n-i} + a_o \tag{14}$$

$$x_j = f_o(\text{net}_{oj}) \tag{15}$$

ω_{ji},ω_{oj} j The weights of the connection between the input layer and the hidden layer and the weights of the connection between the hidden layer and the output layer are in the range [-1,1].f_j, $f_0$are the activation functions of the hidden layer and the output layer, respectively.

During the backpropagation process of the BP algorithm, when there is a difference between the actual output ŷt and the expected output y_t, the error calculation is as follows:

$$E = \frac{1}{2}(\hat{y_t} - y_t)^2 \tag{16}$$

According to the gradient descent method, the update formula of the weights can be obtained as:

$$\Delta\omega_{ed} = -\lambda \frac{\partial E}{\partial \omega_{ed}} \tag{17}$$

Where λ is the learning rate andω_{ed} is the connection weight between two nodes. After the model is trained, the eigenvalues of the entity are input to the neural network for calculation, and the d-dimensional output of the calculation result of the last hidden layer (the blue part in Figure 8) is used as the final embedding vector.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

3. Experimental Simulation and Analysis

3.1. Experimental setup and dataset

The experiment uses a computer with Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz and 16g memory.The operating system is macOS, the compiler software is python3.8, and the deep learning framework is pytorch1.10.0. To verify the proposed FADH algorithm, experiments are carried out on the same data set using different angles. Table 2 gives the detailed statistics of the data set.

hetrec2011-movielens-2k-v2 The dataset's source is the 2nd International Symposium on Information Heterogeneity and Fusion for Recommender Systems, an extended MovieLens-10m, including 10,197 movies, 20 movie types, 4,060 directors, and 95,321 actors on average per movie. There are 22.778 actors, the number of countries in the movie is 10197, and each movie has an average of 1 attribution country, involving a total of 47899 attribution country locations.

Table 2. Dataset statistics

Dataset	total	genre	Actor	User	ratings
hetrec2011-movielens-2k-v2	10197	20	95321	2113	855598

3.2. Baseline Model

The baseline method is as follows:

Node2vec:Node2vec defines the concept of a flexible vertex network domain and implements a biased random walk technique that explores different neighborhoods to achieve richer representations.

GraphSage: GraphSage is an inductive learning framework that combines topology and entity vertex attribute information to efficiently generate embeddings of unknown vertices in graphs. The core idea is to generate the embedding vector of the target vertex by performing an aggregation representation function on a pair of neighbor vertices.

FastRP:Fast Random Projection is referred to as FastRP. It is a node embedding algorithm in the random projection algorithm family. The Johnson-Lindenstrauss lemma theoretically supports these algorithms, according to which n vectors of arbitrary dimension can be projected to O(log(n)) dimension. Such techniques allow aggressive dimensionality reduction while preserving most of the distance information. The FastRP algorithm operates on graphs and, in this case, is concerned with maintaining the similarity between nodes and their neighbors. This means that two nodes with similar neighborhoods should be assigned similar embedding vectors. Conversely, two dissimilar nodes should not be assigned similar embedding vectors.

3.3. Parameter setting

- FADH Model parameter settings;
  For all datasets, the size of the embedded entity dimension is [32, 64, 128, 256, 1024] for the baseline algorithm model and the parameter model of the FADH algorithm in this paper, the number of training rounds is 1, and the initial learning rate of the neural network is 0.01. The learning rate is reduced by a minimum of 0.0001 after each training iteration. The network structure designed in this paper is shown in Table 3 below:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

Table 3. Fully connected neural network structure

	input	Hidden layer 1	Hidden layer 2	Hidden layer 3	Hidden layer 4	Hidden layer 5	Hidden layer 6	Output layer
1	1024	1024	1024	1024	32	32	32	1
2	1024	1024	1024	1024	64	64	64	1
3	1024	1024	1024	1024	128	128	128	1
4	1024	1024	1024	1024	256	256	256	1
5	1024	1024	1024	1024	1024	1024	1024	1

Since the equal-width deep network has higher parameter utilization, the model effect is better. In this experiment, the neural network model is set as the input layer of 1024 dimensions, the middle hidden layer is six layers, the first three layers are all 1024 nodes, and the sixth hidden layer has the same number of nodes as the required dimension.

- Baseline model parameter settings
  The Node2vec model parameters are set, the depth of the random walk is 80, the number of random walks generated by each node is 10, the localization parameter (the tendency of the random walk to stay close to the starting node or fan out) is set to 1, a higher value Means to keep local state. The training neural network context window size is 10. The number of negative samples generated for each positive sample is 5. GraphSage and FastRP model parameters are set according to the parameters shown in the literature.

3.4. data visualization

To further understand the performance results of the algorithm and observe the obtained vector space, the T-SNE dimensionality reduction algorithm is used. Entities with similar attribute features should be closer together in the space vector, as shown in the figure.

In the experimental data of this paper, each entity has attributes of 8 domains, encoded by the FADH algorithm and then reduced to 2-dimensional details by T-SNE for visual analysis. An example of dimension data is shown in Table 4.

Table 4. This is a table caption. Tables should be placed in the main text near to the first time they are cited.

Movie title	2-D vector	
Scream	0.21266807615756989	0.5888321399688721
Tommy Boy	0.9140979647636414	0.7566967606544495
Alien	0.9574490785598755	0.41326817870140076
	.....	
She's So Lovely	0.7865368723869324	0.10100453346967697
U.S. Marshals	0.0719037652015686	0.07177785784006119
	.....	
A Night at the Roxbury	0.5604234933853149	0.8568414449691772

Show some nodes in two-dimensional space as shown in Figure 10:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

Figure 10. Dimensionality reduction visualization

The application of embedding has been pervasive, but there is no perfect solution for its evaluation. In practical applications, the way to evaluate its quality is to use the embedded vector to evaluate the specific task. The most common way is to calculate the TOP N similarity, check whether the distance in the space is consistent with human intuition, and see if the model has learned it. Meaningful information; you can also learn from the method of clustering visualization in word embedding, sample, and compare to see how the effect is. The scheme in this paper is to encode the movie entity. To verify whether the embedding vector has learned the internal features, we calculate the relative distance and use the clustering algorithm to visualize the distribution effect of the vector.

In clustering the data, the K-elbow method is used first to determine the number of clusters. As the clustering coefficient K increases, the sample division will be more refined, the degree of aggregation of each cluster will gradually increase, and the sum of squared errors will gradually decrease. The clustering coefficient calculation results are shown in Figure 11.

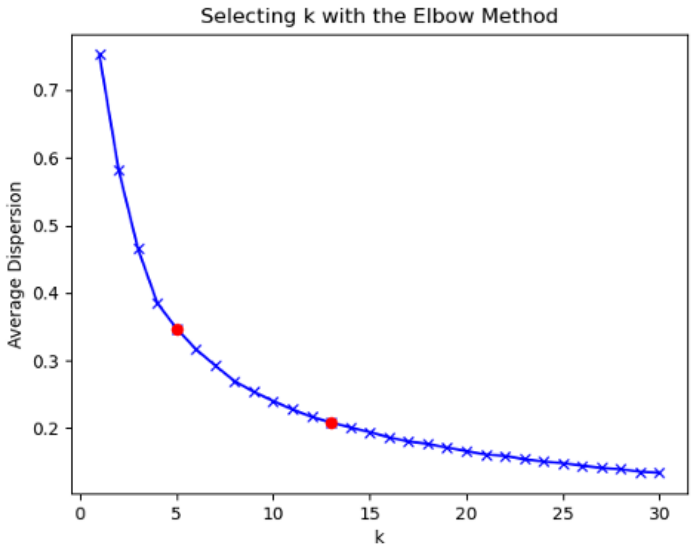

Figure 11. clustering coefficient

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

1. When the clustering coefficient is from 1 to 5, the sum of squares of errors decreases rapidly, indicating that the number of clusters is quickly optimized to find the optimal number of clusters within this interval.
2. When the clustering coefficient is 5 to 13, the rate of decline of the clustering coefficient slows down, and the most likely interval of the number of clusters is most likely to be close to the actual situation.
3. When the clustering coefficient is 13 to 30, as the clustering coefficient increases, the decrease of the sum of squared error is the slowest. It has been stated that the rise in the number of clusters has exceeded the actual number of clusters represented by the data.

In the interval of the clustering coefficient from 5 to 13, the specific clustering situation is displayed, as shown in the following figure.

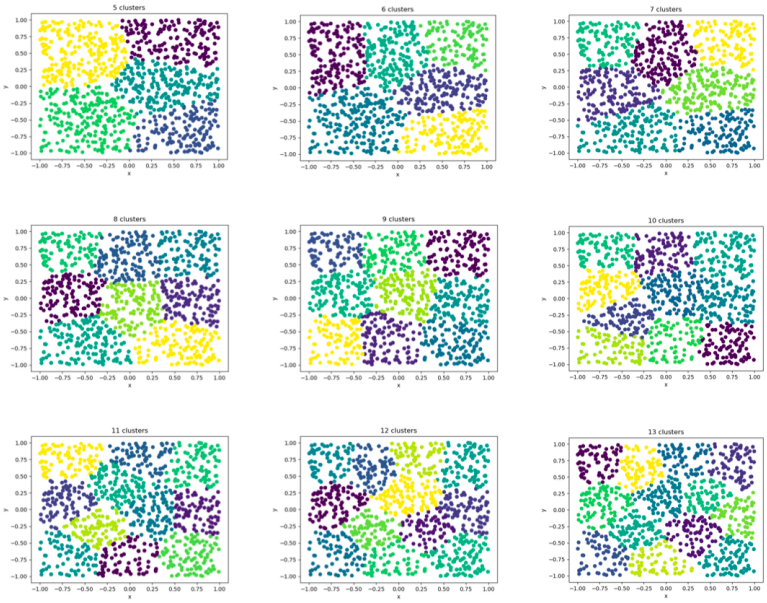

Figure 12. 9 different cluster visualizations

Evaluating the FADH embedding model from the visualization results, after integrating the attribute features of the entities, it learns the potential associations between the feature attributes and can effectively distinguish entities with similar features.

3.5. Algorithm performance

- A single feature is calculated by relative clustering distance;

The primary method of a single feature domain is to compare the distance between the internal embedding vectors belonging to the same feature and the distance between the external vectors that do not belong to the same feature. The smaller the calculated distance, the more similar the two entities are, and vice versa. The comparison results are shown in Figure 13:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

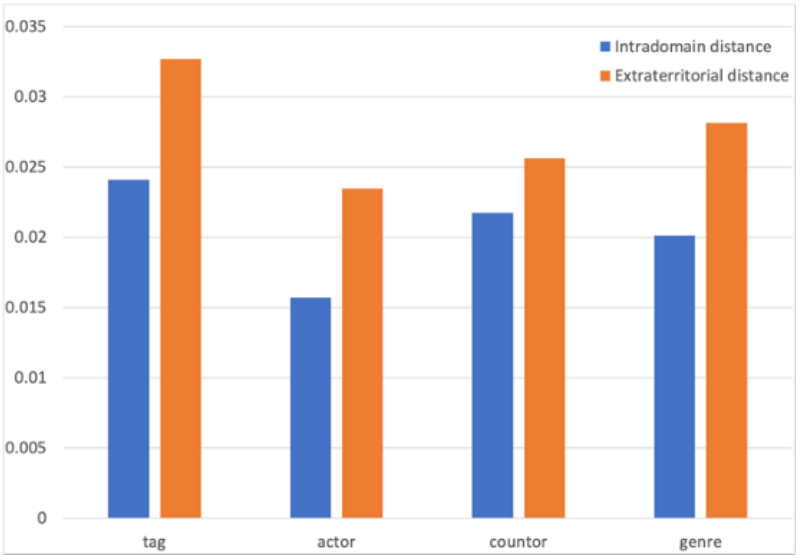

Figure 13. Intra-domain distance comparison

The blue histogram represents the average value of the distance calculation within the entity attribute domain, and the orange is the average value between the attribute domains. After calculating the average distance of each label and comparing, it can be intuitively analyzed from the graph that the average distance between types, actors, countries, and entities within the label is significantly smaller than the external distance. The model can effectively learn the attributes of entity attributes; entities with similar labels are closer in the space to express attribute information.

The experiment uses Node2vec, GraphSage, Fast Random Projection, and hash2vec four-node entity embedding algorithms to compare and test the algorithm in this paper.

Table 5 shows the five closest movie entities in space computed by each algorithm.

Table 5. top 5 nearest neighbor list

	Node2vec	GraphSage	FastRP	FADH
Titanic	Revolutionary Road	Top Gun	Ladder 49	Ghost
	Top Gun	Casablanca	Mighty Joe Young	Revolutionary Road
	Ghost	Revolutionary Road	Revolutionary Road	Ladder 49
	Firestorm	Ghost	Casablanca	Firestorm
	Ladder 49	Firestorm	Top Gun	Top Gun
The Shawshank Redemption	The Green Mile	Fight Club	Fight Club	A Clockwork Orange
	One Flew Over the Cuckoo's Nest	The Silence of the Lambs	One Flew Over the Cuckoo's Nest	The Silence of the Lambs
	Fight Club	The Sixth Sense	The Green Mile	The Green Mile
	A Clockwork Orange	Mr. Smith Goes to Washington	The Sixth Sense	Fight Club
	Forrest Gump	The Green Mile	The Bridge on the River Kwai	One Flew Over the Cuckoo's Nest

The vector of embedded nodes obtained by the FADH algorithm has a similar expression effect to the existing graph node embedding algorithm under the constraints of entity attribute information rules. For example, the embedding vector obtained by the embedding of the FADH algorithm in the movie "Titanic" has the same number of 95% as the 20 nearest

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2022 doi:10.20944/preprints202212.0040.v1

neighbors calculated by the Node2vec algorithm and 80% of the GraphSage algorithm and the FastRP algorithm. Similarity to the above. To sum up, the FADH algorithm in this paper has a better language expression effect.

3.6. Algorithm performance

In the experiment, the time and memory used to train the FADH and Node2vec models using the same data set are shown in Table 6 below:

Table 6. Time and memory consumed by the model

Dimension		32	64	128	256	1024
FADH	time/s	10431	10600	10833	13092	40319
	memory /MB	6.3	11.2	21.1	40.6	158.4
Node2vec	time/S	14773	17483	20849	32531	74773
	memory /MB	8.2	13.4	24.5	51.1	169

An essential factor for the FADH algorithm to dynamically add data is that when an entity generates an embedded vector, it is only related to its attributes and does not involve other entities. Only its attributes can generate the corresponding vector representation. The training time only involves the embedding dimension. When the number of texts is fixed, the training time and memory are linearly related to the embedding dimension. The larger the embedding dimension, the longer the time and the larger the memory usage. Compared with other models that require data embedding and overall data training, the FADH model has significant advantages.

4. Conclusion

Based on deep neural networks and hashing algorithms, this paper proposes a novel embedding algorithm that can learn the attribute association between entities. An entity coding method that integrates attribute information is adopted to extract entity information from three aspects: coding fusion, feature extraction, and feature learning of entity attributes. And to solve the problem that different entities may have the same embedding representation, this paper proposes using multiple hash functions to extract features to represent an entity . This effectively alleviates the problem that the model cannot distinguish between different feature values, thereby improving the model's efficiency at the samently time; this paper takes the film data as an example to introduce the coding method and specific algorithm process in detail to realize the effect of dynamically adding data model reuse. The experimental setup is to compare the FADH algorithm designed in this paper with the Node2vec algorithm, GraphSage algorithm, and Fast Random Projection algorithm, respectively, and analyze the time performance, memory consumption, and expression ability of different algorithms in embedding vectors of different dimensions. The experimental results are shown as follows: First, according to the five relevant entity properties closest in the vector space, referring to the first 20 nearest neighbors of the embedding vector under different algorithms, the similarity between the FADH algorithm and the GraphSage algorithm and the FastRP algorithm in this paper reaches more than 80%, and the similarity with the Node2vec algorithm reaches more than 95%. Second, compared with other algorithms, the FADH algorithm designed in this paper consumes the least time and occupies the smallest memory. In summary, the FADH algorithm developed in this paper has a good language expression effect, and the performance of time and spatial complexity is significantly improved.

The FADH algorithm designed in this paper has considerable potential in processing entity attribute information fusion. However, there are certain shortcomings. First, there is no perfect solution to the problem of embedding model evaluation. How to obtain more truthful evaluation criteria remains to be discussed. Second, the model ignores the degree of influence of different features on different entities. In practice, the effects of other features on various entities are not necessarily the same.

In the future, the promotion of embedded algorithms incorporating additional information in the field of personalized recommendation in recommender system research is still a challenge. At the same time, the influence weight between different features and entities is the following research focus. We still have a long way to go in applying algorithms to more novel and meaningful other applications.

5. ACKNOWLEDGEMENTS

This work is supported by: Key Project of Application Foundation of Sichuan Science and Technology Department, China(Grant No.2019YJ0455); ChunHui plan project of Ministry of Education, China(Grant No.z2011089); Graduate Innovation Fund of Xihua University, China(Grant No. YCJJ2021072).

References

1. Qiao Y, Yang X, Wu E. The research of BP neural network based on one-hot encoding and principle component analysis in determining the therapeutic effect of diabetes mellitus *IOP Conference Series: Earth and Environmental Science. IOP Publishing* **2019**, *267(4)*:042178.
2. Li, Jia, et al. Deep convolutional neural network based ECG classification system using information fusion and one-hot encoding techniques. *Mathematical problems in engineering* **2018**
3. Kang W C, Cheng D Z, Yao T, et al. Learning to embed categorical features without embedding tables for recommendation. *arXiv preprint 2010.10784.* **2020**
4. Weinberger K, Dasgupta A, Langford J, et al. Feature hashing for large scale multitask learning *Proceedings of the 26th annual international conference on machine learning.* **2009**:1113-1120.
5. Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality *Advances in neural information processing systems.* **2013**,26.
6. Joulin A, Grave E, Bojanowski P, et al. Bag of tricks for efficient text classification *arXiv preprint arXiv:1607.01759.* **2016**
7. Barkan O, Koenigstein N. Item2vec: neural item embedding for collaborative filtering *IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP). IEEE* **2016**: 1-6.
8. Covington P, Adams J, Sargin E.Deep neural networks for youtube recommendations *Proceedings of the 10th ACM conference on recommender systems.* **2016**: 191-198.
9. Cheng H T, Koc L, Harmsen J, et al.Wide & deep learning for recommender systems[C]//Proceedings of the 1st workshop on deep learning for recommender systems. *Proceedings of the 10th ACM conference on recommender systems.* **2016**: 7-10.
10. Perozzi B, Al-Rfou R, Skiena S. Deepwalk: Online learning of social representations *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* **2014**:701-710.
11. Hamilton W, Ying Z, Leskovec J.Inductive representation learning on large graphs *Advances in neural information processing systems* **2017**,30.
12. Chen H, Sultan S F, Tian Y, et al. Fast and accurate network embeddings via very sparse random projection*Proceedings of the 28th ACM international conference on information and knowledge management.* **2019**,66(4): 399-408.
13. Achlioptas D. Database-friendly random projections: Johnson-Lindenstrauss with binary coins *Journal of computer and System Sciences.* **2003**,66(4): 671-687.
14. Argerich L, Zaffaroni J T, Cano M J. Hash2vec, feature hashing for word embeddings *arXiv preprint arXiv:1608.08940.* **2016**
15. Tito Svenstrup D, Hansen J, Winther O. Hash embeddings for efficient word representations *Advances in neural information processing systems.* **2017**,30
16. Yan B, Wang P, Liu J, et al.Binary code based hash embedding for web-scale applications *Proceedings of the 30th ACM International Conference on Information & Knowledge Management.* **2021**: 3563-3567.
17. Dan Tito Svenstrup, Jonas Hansen, and Ole Winther.Hash embeddings for efficient word representations. *Advances in neural information processing systems.* **2017**:30,4928–4936
18. Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. *In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.:165–175.
19. Fumito Yamaguchi and Hiroaki Nishi .Hardware-based hash functions for network applications. *In 19th IEEE International Conference on Networks (ICON). IEEE,.* **2013**,1–6.
20. Li F, Yan B, Long Q, et al.Explicit Semantic Cross Feature Learning via Pre-trained Graph Neural Networks for CTR Prediction *SIGIR.* **2016**: 2021
21. Luo X, Wang H, Wu D, et al. A survey on deep hashing methods *ACM Transactions on Knowledge Discovery from Data (TKDD).* **2020**
22. Deng Y. Recommender systems based on graph embedding techniques: A comprehensive review *arXiv preprint arXiv:2109.09587.* **2021**
23. Al-Ansari K. Survey on Word Embedding Techniques in Natural Language Processing **2020**

 doi:10.20944/preprints202212.0040.v1

24. Song T, Luo J, Huang L. Rot-pro: Modeling transitivity by projection in knowledge graph embedding *Advances in Neural Information Processing Systems.* **2021**,34: 24695-24706.
25. Grover A, Leskovec J. node2vec: Scalable feature learning for networks *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining.* **2016**: 855-864.