A Survey of Current Embedding Techniques and Applications

1st Maxat Alibayev
Computer Science and Engineering
University of South Florida
Tampa, United States

2nd Fred Mubang
Computer Science and Engineering
University of South Florida
Tampa, United States

***Abstract*—Data embedding has become an increasingly popular technique to aid in the processing of unstructured data. By mapping data to a lower dimensional space, one can perform various machine learning tasks with much greater ease, such as image and video segmentation, recommendation system building, and semantic search, among many other tasks. However, there is a considerable amount of literature to parse through when trying to decide the most relevant embedding approach for one's problem. This can be daunting for new researchers to the field, so we attempt to address this issue through this survey paper. In this paper, we describe recent approaches in embedding techniques and applications. The domains covered are as follows: (1) Graph Embedding, (2) Cross Modal Embedding, (3) Embedding for Segmentation, and (4) Embedding within the context of Zero-Shot Learning.**

***Index Terms*—Embedding, Representation Learning, Segmentation, Graph Theory, Zero-Shot Learning, Cross-Modal Retrieval**

I. Introduction

One of the most common issues in Machine Learning is figuring out how to properly wrangle large amounts of unstructured data when creating models. Embedding data into a lower dimensional space seems to aid greatly in alleviating this issue. Embedding can be described as the translation of high dimensional data into a lower dimensional form. Some, but not all applications of embedding include natural language processing, image and video captioning, recommendation system building, semantic search, social network analysis, and drug discovery.

This paper is a survey paper of several recent breakthroughs within the field of embedding. Specifically, this paper focuses on the fields of graph embedding and embedding in computer vision. The contributions of this paper are as follows:

1) A general overview of several popular embedding domains are given. Basic concepts and definitions are described for those new to the field.
2) The state-of-the art algorithms within each embedding domain are described, which is useful for any researcher desiring a succinct summary of recent approaches.
3) The similarities and differences between the different embedding approaches are discussed, which allows the reader to know any tradeoffs or benefits that come with using one type of algorithm over another.

This paper's format is as follows. Section II will cover Graph Embedding Techniques and Approaches. This section will define the graph embedding problem, establish basic definitions, and discuss applications. There will be 4 different types of graph embedding domains discussed: (1) graph embedding for recommendation systems, (2) graph embedding for knowledge graphs, (3) dynamic graph embedding, and (4) the importance of leveraging proximity information when embedding nodes in a graph.

Section III will focus on the applications embedding in computer vision problems, such as cross-modal retrieval (5 papers), video/image segmentation (3 papers), and zero-shot learning (2 papers). We will discuss how these algorithms incorporate the embedding concepts into their frameworks, as well as their similarities and differences.

II. Graph Embedding

A graph is a mathematical structure used to model the relationships between objects. It can be represented as a set $G = (V, E)$. V is the set of nodes in G, and E is the set of edges in G. The number of nodes is $n = |V|$. A single node can be represented as v_i, such that $i \leq n$. An edge can be represented as a tuple of form (v_i, v_j) in which $i \leq n$ and $j \leq n$.

Graph Embedding is the representation of a graph in a lower dimensional space. It is useful because most adjacency matrix representations of graphs suffer from sparsity. In this section, some current graph embedding techniques and applications are discussed. We also discuss potential future research ideas within this domain.

A. A Formal Definition of Graph Embedding

The objective of graph embedding is follows. Given a graph, G with n vertices, and a predefined dimensionality, $d < n$, we must convert G into a d-dimensional space. d is usually much smaller than n. The embedding of G into a smaller dimensional space is done to achieve a dense representation of the original graph. Figure 1 from Hu et al. shows an example of embedding a graph into a 2-dimensional space [1].

B. Additional Concepts and Definitions

Before discussing the different graph embedding approaches, we present some basic definitions.

1) **Homogeneous vs. Heterogeneous Graphs**: A homogeneous graph is a graph in which there is only 1 type of

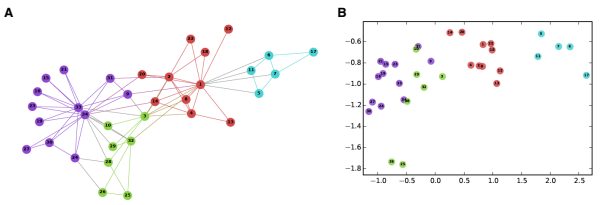

Figure 1. An example of a graph embedding from Hamilton et al. [1]. The network used is the Karate Network, a popular small graph dataset. In (A), we have our initial, unstructured graph. In (B), we have its embedding. In this case, the number of dimensions, d, is equal to 2, but more dimensions can be used.

node and 1 type of edge. For example, in [2], Hu et al. describe their experiments in an item-item graph in which the 1 type of node is an item, and the 1 type of edge is an (item, item) edge. In a heterogeneous graph, there can be more than type of node or edge. For example, in [3], Ruan et al. use an herb-symptom graph, in which there are two different node types: herbs and symptoms. There are also different edge types: (herb, symptom), (herb, herb), and (symptom, symptom).

2) **Static vs. Dynamic Graphs**: As previously mentioned, a graph is simply a structure used to model the relationships between objects. This basic type of graph can be considered "static" because there is no temporal information encoded into this modelling. However, a dynamic graph is a structure used to model the relationships between objects over time. So, it can be represented as a set, $G_{temporal}$ which is comprised of graph snapshots, $G_{t1}, G_{t2}, ...G_{tT}$, at T different time steps, $t_1 < ... < t_T$. The *SPE [2], HS2Vec [3], WWV-KG [4]*, and *NECS* [5] algorithms in this paper are all static graph algorithms. *tNodeEmbed* [6] and *DynGAN* [7] are dynamic graph algorithms.

3) **Adjacency Matrix**: An adjacency matrix is one type of way to represent a graph. If n is the number of nodes in a graph, an adjacency matrix A, would be a matrix of size n x n. Each entry, v_{ij} in an adjacency matrix indicates whether an edge is formed between nodes v_i and v_j. Typically 1 is used to indicate they are connected, and 0 is used to indicate they are not. Additionally, a weight may be applied to these values as well to indicate the relative "strength" of the edge.

4) **Proximity**: Proximity refers to the similarity of two nodes. Formally, $proximity(v_i, v_j) = s_{ij} = s(v_i, v_j)$, in which v_i and v_j are nodes, and s is a similarity function. Different pieces of literature describe proximity in different ways. We take our definition from [1], [2], and [5].

5) **First Order Proximity**: This is the weight of the edge between two nodes v_i and v_j. One can use a Facebook network graph as an example to illustrate this concept. Assume the nodes are users and assume that the edges represent whether or not a user v_i messaged user v_j. The first order proximity could represent how many times user v_i messaged user v_j [5]..

6) **Second Order Proximity**: Second Order Proximity refers to the similarity between the neighborhoods of two nodes v_i and v_j [5].

7) **Higher (Kth-Order) Proximity**: There are various definitions of k higher order proximity depending on the literature, but in this paper we will use this term to refer to the k-hop transition probability between two nodes v_i and v_j. In other words if one were to perform a random walk on the graph G, what is the probability of reaching node v_j from v_i in k or less hops? We take our definition of higher order proximity from [5].

C. Link Prediction

In this section we will discuss both static and dynamic link prediction. These are two common tasks that are highly prevalent in the graph embedding papers discussed in this survey, as well as in the field in general.

1) **Static Link Prediction** This is a common task for static graph prediction models. The input to the model is a graph in which some edges are missing or removed. A successful link prediction model must be able to predict the values of the missing edges given the present edges. An application of this task in the real world is item recommendation. In this setting, the edges would represent whether a particular user would buy a particular item, in which case the edge would take on the form (user, item). Alternatively, the edge could represent whether someone who bought item A would also buy item B (item, item). Figure [8] shows an example of static link prediction within the domain of user-item recommendation. The *SPE [2], HS2Vec [3], WWV-KG [4]*, and *NECS* [5] algorithms in this paper can be used for static link prediction.

2) **Dynamic Link Prediction** In this setting, the model must predict whether an edge between two nodes will exist at some point in the future. Figure 3 from [9] shows an example of dynamic link prediction. For example, in the image, in timestep G_t, edge (2, 3) does not exist, but in timestep G_{t+1}, it does. An application of dynamic link prediction would be predicting which users will interact with each other on a social media platform, or predicting whether two proteins will interact in a protein-protein interaction network (PPI). *tNodeEmbed* [6] and *DynGAN* [7] can be used for dynamic link prediction.

D. Embedding for Recommendation Systems

In this section we will discuss graph embedding techniques for Recommendation systems. We will discuss 2 different algorithms: Semi-Parametric Embedding (SPE) [2] and and HS2Vec [3].

*1) **Recommendation Algorithm #1: Semi-Parametric Embedding**:* The SPE model was made for item to item recommendation (I2IR). I2IR is the field of study that is concerned with how to recommend items to a user given their purchasing history [2]. An example of I2I recommendation is Amazon's "Customers Also Purchased" feature on their website. One

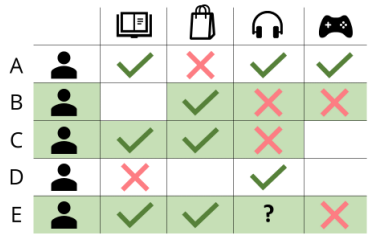

Figure 2. A real-world static link prediction example. Given the partially completed matrix of user-item ratings, we must predict whether or not a user should be recommended a particular item. Source: [8].

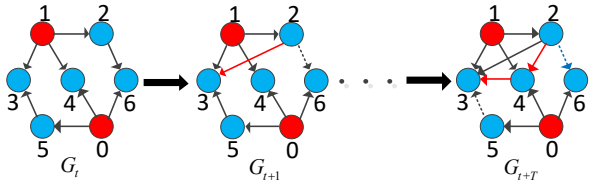

Figure 3. An example of dynamic link prediction. In timestep G_t, edge (2, 3) does not exist, but in timestep G_{t+1}, it does. Source: [9].

of the main issues of the I2I field is figuring out the best methodology for recommending items to a user. There are 2 approaches.

Firstly, there is the behavior-based approach which makes the assumption that similar users will prefer similar items. In other words, if Stan likes a movie, X, the recommendation system will look at the movie history of other users who viewed movie X and will recommend Stan a movie based on those users' history.

Secondly, there is the content-based approach, which assumes that similar items will be preferred by similar users. For example, if Stan likes a movie, X, the recommendation system will solely recommend a movie to Stan based on the features of that movie. So, if he saw a movie about cars, the algorithm will try to find other movies about cars that Stan would like. A content-based model does not take into consideration what other users liked.

The SPE algorithm seeks to combine both the behavior-based and content-based recommendation methdologies into 1 model [2]. Specifically, it takes as input both an item-item co-occurrence matrix, **R**, the item feature (content) matrix, **C**, and a statistics matrix **B**. To evaluate the effectivess of SPE, the authors used a similarity score to compare the results of SPE to real-life item-item co-occurrences. A successful recommendation algorithm should be able to give a high ranking to similar items and a low ranking to less similar items. The similarity score the authors used is the following:

$$s(v_i, v_j) = \frac{1}{1 + e^{-v_i^T v_j}} \tag{1}$$

Vectors v_i and v_j are the embedding vector for two items. The sigmoid function is used to keep the similarity score between 0 and 1.

*2) **Recommendation Algorithm #2 HS2Vec for Herb-Symptom Graphs**:* : The authors of [3] created an embedding algorithm for an herb-symptom co-occurrence graph. The application of these embeddings would be for regularities analysis and herb recommendation in the context of Traditional Chinese Medicine. An algorithm that utilized this embedded herb-symptom graph would be addressing the following problem: *Given a list of symptoms, what would be the best herbs to use for treatment?*

The steps of this algorithm are as follows:

1) First, an herb-symptom co-occurrence matrix is created from prescription data. There is an edge between an herb and a symptom if they co-occur in the same prescription. There is an edge between two herbs if they can address the same symptom, and there is an edge between two symptoms if they can be treated by the same herb. This concept of connecting two nodes by their association with another node is known as *meta-path based proximity*.
2) This resulting matrix, T is the *Traditional Chinese Medicine Network* (TCMN). It is sent through an autoencoder that has the following objective: to embed the matrix to the lower dimension so that 1st and 2nd order proximity is maintained throughout each node embedding. In other words, if two herbs treat similar symptoms, then their embeddings should be similar to one another.
3) The autoencoder has now successfully created an embedding space for the TCMN network. These embedding vectors can be used for a variety of downstream tasks such as clustering and link prediction.

Figure 4 shows the architecture of HS2Vec.

*3) **HSVec and SPE: How Are They Similar?**:* SPE [2] and HSVec [3] are similar in the following ways. They are both embedding approaches with use cases cases within the field of recommendation. SPE is used for embedding item-item matrices for general item recommendation. As a matter of fact, the authors used Amazon, Yelp, and Alibaba datasets in their experiments to show its capability of operating in such a domain. These 3 companies are well known for using recommendation systems as part of their infrastructure.

HS2Vec can be used for recommending the best herb treatment given a list of symptoms in Traditional Chinese Medicine. It can also be extended to recommend the best medications given a list of symptoms in Western Medicine. Additionally, both approaches are similar in that they require the calculation of co-occurrence matrices. For SPE, the co-occurrence matrix represents how often each item is bought or clicked by the same user. For HS2Vec, the co-occurrence matrix represents how often certain herbs and symptoms co-occur within the same prescription [3].

*4) **HSVec and SPE: How Are They Different?**:* The two algorithms differ in the following ways. The item-item matrix used in SPE is a *homogeneous network*. Recall that a homogeneous network is one in which there is only one type of node and edge [2]. In this case, the node type is only *item* and the edge type is *(item, item)*.

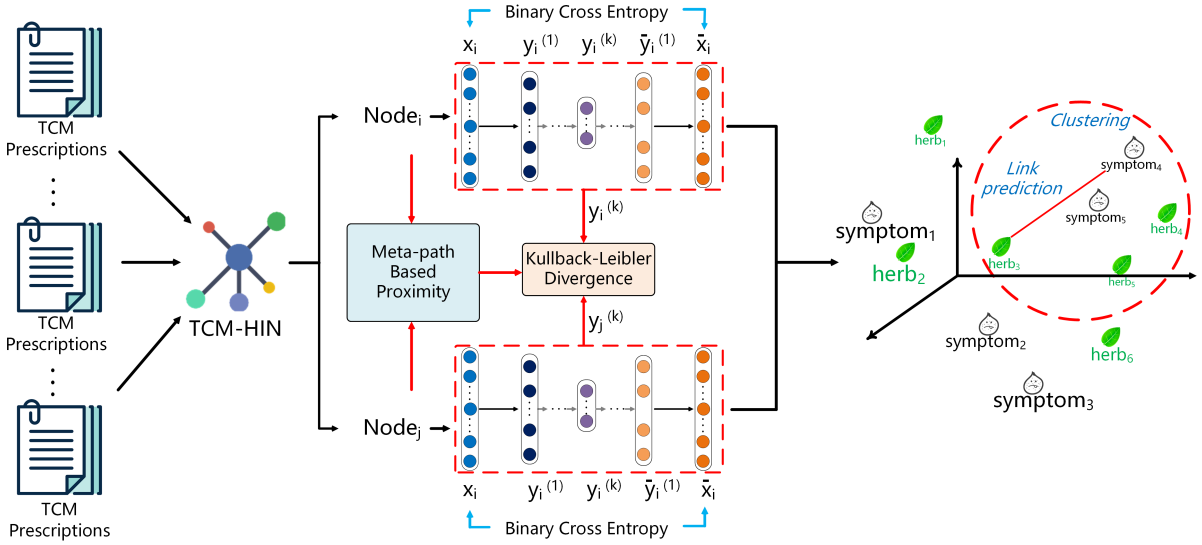

Figure 4. The HS2Vec architecture [3].

Contrast this with the Traditional Chinese Medicine Network used in HS2 Vec, which is a *heterogeneous* network [3]. Recall that a heterogeneous network is one in which there is more than 1 type of node and/or more than 1 type of edge. In the TCMN network, there are two types of nodes, *herb* and *symptom*. Furthermore, there are 3 types of edges: *(herb, herb)*, *(herb, symptom)*, and *(symptom, symptom)*.

The difference in corresponding graph types leads into the next major difference between the two algorithms. Since HS2vec takes as input a heterogeneous graph, the authors had to use meta-path-based proximity in order to accurately capture the relationships among herbs and symptoms. To calculate meta-path-based proximity, one must find the metapath edges.

A metapath edge is an edge created by creating a path between two otherwise disconnected nodes. A path is found between the two nodes by their association with 1 or more nodes in the graph. For example, let H_1 and H_2 be herbs, and let S be a symptom. If herbs H_1 and H_2 could be both be used to treat symptom S, it can then be said that there is a metapath that exists from H_1 to H_2 of the following form:

$$H_1 \rightarrow S \rightarrow H_2$$

Applying this methodology at scale to thousands of prescriptions allows one to create a richer Traditional Chinese Medicine Network, which in turn would allow HS2Vec to create better node embeddings [3].

E. Embedding for Knowledge Graphs

This section introduces the concept of knowledge graphs. They have many applications in query systems. However, recent research has applied them to recommendation systems as well. First, the definition of knowledge graphs will be given, followed by a discussion of a recent knowledge graph embedding algorithm, called the *Weighted-Word Vector Model for Knowledge Graphs* (WWV-KG) [4]. The last part of this section will discuss potential future research ideas combining knowledge graphs with recommendation system embedding algorithms such as HS2Vec and SPE.

1) ***Knowledge Graph Definition****:* A knowledge graph is a representation of knowledge as a collection of facts. As described by Veria et al. in [4], each fact is a triplet comprised of *subject*, *predicate*, and *object*. More formally, a triplet, *(subject, object, predicate)*, would be notated as (e_s, r_p, e_o). e_s and e_o are nodes known as *entities*, and the *relation*, r_p, would be the type of edge that connects these two nodes together. Note that a knowledge graph is a type of heterogeneous graph because there are two types of nodes: subject and predicate nodes, and there are multiple types of relation edges. An example of a fact triplet would be *(Thomas Jefferson, "Is A Founding Father in addition to", George Washington)*. *Thomas Jefferson* and *George Washington* are the entities, and they are related to one another because they are both Founding Fathers of the United States of America.

An example of a knowledge graph application would be Google's Knowledge Graph as described by Singhal in [10]. It is a knowledge base implemented as a knowledge graph, and it is used by Google to enhance a user's search results. When a user enters a query into Google Search, on the results page, there is an info box that appears with some information related to the query.

2) ***Improving Knowledge Graphs with Text Data****:* As one can see, the strength of a knowledge graph relies on its underlying knowledge base. Without a rich collection of fact triplets, a knowledge graph cannot be of any use. The authors of [4] presented an unsupervised approach to improving the collection of knowledge graph triplets using text data. The framework is as follows.

1) The input to the framework is the full corpus of text data being used, as well as all relations of interest from the

pre-existing knowledge graph.
2) This text is used to a train a Word2Vec model. Each word in the corpus now has an associated word vector.
3) Entity vectors are then created by performing a weighted mean among all relevant word vectors. For example, a word vector would be considered relevant if it appeared within the same sentence as an entity. Irrelevant word vectors are not included in the weighted mean.

The triplets can then be mapped to an embedding space. The loss function of this algorithm is a triplet loss, in which a lower loss is output if the distance between the anchor embedding and positive embedding is smaller than the distance between the negative embedding and the anchor.

*3) **Using Knowledge Graphs With HS2Vec**:* The ideas presented in the HS2Vec paper [3] can be applied to knowledge graph generation as well. Note that the limitation of this network is that the only information used to associate symptoms with herbs is their co-occurrences and meta-paths. If one were to convert the TCMN into a knowledge graph, the types of fact triplets derived could take the following form: ($herb_i$, "can be used to treat $symptom_j$", $herb_k$), or ($symptom_i$, "can both be treated by $herb_j$", $symptom_k$).

However, in the real world, much more information might be taken into account when treating a patient, such as their medical history, genetic information, etc. Some additional node types that could be added to the TCM Knowledge Graph could be: *patient*, *disease*, etc. An example of a triplet fact within this new type of knowledge graph could be: (patient, "is allergic to" herb).

F. Dynamic Graph Embedding

Thus far, this section has mainly discussed embedding techniques and applications for static graphs. However, many networks in the real world are dynamic and change over time. New edges can appear at different points in time, as shown in Figure 3. In this section, we will discuss dynamic network embedding techniques and applications.

*1) **tNodeEmbed**:* In [6], Singer et al. present their algorithm, *tNodeEmbed*, a node embedding algorithm for temporal graphs. Their goal is to find for each node $v \in V$ at time T a feature vector f_T that minimizes the loss of any given prediction tasks. The two tasks the authors focus on is node classification and link prediction.

tNodeEmbed uses a matrix Q_t, for each timestep t, that is initialized using the popular node-embedding algorithm, *node2vec*. The objective of node2vec is to embed a node so that its entire neighborhood vector can be predicted from it. tNodeEmbed utilizes node2vec's initial embeddings so that it can capture how a node's neighborhood vector evolves over time [6].

*2) **DynGAN**:* In [7], Maheshwari et al. present a DynGAN, which is a dynamic graph embedding model that leverages Generative Adversarial Networks, or GANs. GANs were first introduced by Ian Goodfellow et al. in [11]. They are neural network architectures each comprised of two networks, a Generator (G), and a Discriminator (D). The goal of G is to create samples that are similar to the training set. The goal of D is to be able to successfully distinguish between *real* examples from the training set, and *fake* examples from G. If G and D are trained properly, G will be able to create samples so similar to the training set that D cannot tell the difference.

There are 2 variations of the DynGAN model, the DynGAN and DynGAN LSTM model. The DynGAN model is an ensemble model comprised of a sequence of GAN models of length $t+l$. l is a look-back factor that represents how many timesteps the GAN model should "look-back" in time when training on the dynamic graph sequence. The DynGAN-LSTM model has a similar setup to the DynGAN model, except it also includes LSTM layers. The input to either DynGAN model is a sequence of graphs of length $t+l$. The output of the model is the adjacency matrix of the graph at time $t+l+1$. In other words, the model takes in a sequence of graphs and tries to reconstruct what the adjacency matrix would look like at time $t+l+1$.

The objective function is as follows:

$$\begin{aligned}&\min_{\theta_G}\max_{\theta_D} V(G,D)\\&= \mathbb{E}_{v\sim p_{true(.|v_c)}}[\log D(v,v_c;\theta_D,w_D)]\\&\quad + \mathbb{E}_{v\sim p_{G(.|v_c;\theta_G,w_G)}}[\log(1-(D(v,v_c;\theta_D,w_D))]\end{aligned} \tag{2}$$

v_c is a context node. v is a neighbor of v_c. θ_D and w_D represent the parameters and weights of D, respectively. $p_true(.|v_c)$ represents the node neighborhood of context node v_c and p_G represents the predicted node neighborhood of the generator, G. The objective of the generator is to predict what a node's neighborhood will look like at time t, while D's objective is to be able to distinguish whether G's output is correct or not.

Figure 5 shows the architecture of DynGAN-LSTM.

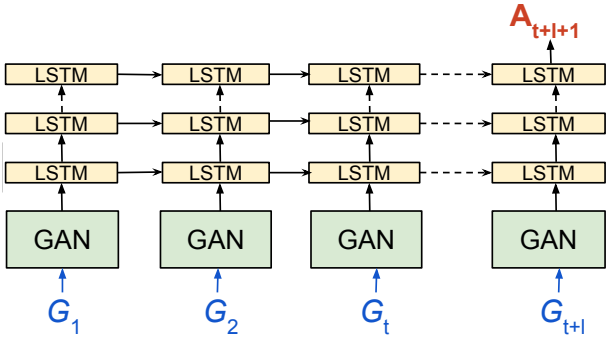

Figure 5. The LSTM DynGAN architecture. The setup is similar to the vanilla DynGAN algorithm, but now there are LSTM layers included in the architecture.

*3) **DynGAN and tNodeEmbed: Similarities**:* DynGAN and tNodeEmbed are similar in the following ways. They both utilize previous time step embeddings to inform their embeddings at the next timestep. They both can be utilized for future link prediction, and the objective both of their loss functions is to re-create the neighborhood of any given node.

*4) **DynGAN and tNodeEmbed: Differences**:* The algorithms differ in some obvious ways. Firstly, DynGAN utilizes an adversarial approach to learn the graph at the next time step, whereas tNodeEmbed does not.

Secondly, both algorithms utilize the previous time step embeddings to make future embeddings, they take different approaches in doing so. In both DynGAN and DynGAN-LSTM, the weights learned by a GAN sub-module at timestep $t-1$ are used to initialize the GAN sub-module at timestep t. However, in tNodeEmbed, the authors also utilize a technique called *matrix rotating*. The intuition behind this technique is that it will keep each node embedding similar throughout each time step even though the node may change behavior over time.

The tradeoff that comes with this technique is that one can only use tNodeEmbed with networks that have the same number of nodes every time step. In order to rotationally align two matrices, the matrices must be of the same size. This may not be ideal because in the real world, many dynamic graphs increase their number of nodes over time. Take any social network, for instance. The number of users on Facebook or Twitter has only grown ever since the beginning of those sites.

DynGAN, on the other hand, can be used with graphs whose nodes increase over time. As a matter of fact, for one of their experiments, the authors used DynGAN on a dynamic graph whose number of nodes increased over time.

The tNodeEmbed creators tested the algorithm on on the Cora and DBLP datasets both with and without the use of the rotation technique. Table II shows the results of this experiment. While this experiment shows that using this technique does improve performance, one must consider if the improvement is worth losing the ability for accounting for growing graphs. For example, in the table, when using the Micro F1 metric, the performance improvement is about 2% and for DBLP the improvement is about 4%. For the Macro F1 metric, the improvement was 4% and 11% for Cora and DBLP respectively. While the Macro F1 definitely shows a considerable improvement, the Micro F1 score does not show much improvement. So, depending on the problem and type of metric that one is most concerned with, it may be better to remove the matrix rotation component of tNodeEmbed if one is working with growing graphs. Otherwise, one could only predict the evolution of the nodes that existed in the first time step. An interesting future experiment would be to compare the performance of tNodeEmbed and DynGAN on graphs that retain the same number of nodes over time. Then, one could remove the rotation matrix component of tNodeEmbed and compare that with DynGAN on graphs that grow in size over time.

Table I
TNODEEMBED NODE CLASSIFICATION PERFORMANCE BOTH WITH AND WITHOUT THE ROTATION ALIGNMENT STEP. WHILE IT DOES MAKE A DIFFERENCE, ONE MUST CONSIDER IF THIS DIFFERENCE WARRANTS LOSING THE ABILITY TO PREDICT PERFORMANCE ON GROWING GRAPHS.

Dataset	Micro F1		Macro F1		AUC		CC
	with	without	with	without	with	without	
Cora	**0.668**	0.644	**0.513**	0.475	**0.925**	0.919	0.275
DBLP	**0.822**	0.785	**0.504**	0.390	**0.997**	0.959	0.002

G. Retaining Proximity Information in Graph Embedding and the NECS Algorithm

An important aspect of graph embedding is retaining the proximity information of each node. Here are 2 ways to incorporate proximity information when embedding:

1) Define node similarity in terms of how many neighbors two nodes have in common, and the weights between any 2 given nodes. Then, in the objective function, optimize the embeddings for maximum similarity between similar nodes or maximum distance between dissimilar nodes.
2) Maximize the probability that the model can re-create the full neighborhood vector of any given node (2nd order proximity) as well as the corresponding weights (1st order proximity). This entails that nodes with similar neighborhoods will have similar embeddings.

Here are some examples of how the algorithms in this paper utilized these ideas:

1) SPE [2] performs (2) when embedding an item feature matrix into the lower dimensional form $g(c)$. Secondly, when the final item embeddings are being used to recommend items, SPE uses (1) because the similarity between two item embedding vectors is calculated.
2) HS2Vec [3] utilizes a loss function that exploits both (1) and (2). It uses an autoencoder to create a lower dimensional embedding for an edge, (v_i, v_j) and its objective is to ensure that the similarity between the hidden layer representations of the edge and the re-created input vectors of the edge are the same as the original edge vector, (v_i, v_j).
3) tNodeEmbed [6] utilizes node2vec which in turn utilizes (2). The objective in node2vec is to embed a node such that one can re-create its neighborhood vector, *N(v)*.
4) DynGAN [7] utilizes (2). The Generator attempts to re-create a neighborhood vector, $N(v_c)$, given a context node 1-hot vector, v_c.

The last graph embedding algorithm to be discussed in this section of the paper is called *Network Embedding with Community Structural Information* (NECS) by Li et al. in [5]. This algorithm heavily utilizes node proximity information for embedding, albeit, in a different way than the aforementioned algorithms do.

The authors of the paper argue that most graph-embedding approaches do not utilize high-order proximity information for graph embeddings, so to remedy that issue, they propose NECS [5]. The input to NECS is a high-order proximity matrix, P, which is created from the graph adjacency matrix **A**:

$$\boldsymbol{P = W_1 A + W_2 A^2 + ... + W_l A^l} \tag{3}$$

In the adjacency matrix, A, the value at coordinate (i, j) is 1 if the nodes v_i and v_j are connected, and 0 otherwise.

Table II
A TABLE OF THE DIFFERENT GRAPH EMBEDDING ALGORITHMS COVERED IN THIS SURVEY, AS WELL AS THEIR OBJECTIVES AND APPLICATIONS.

Algorithm	Objective	Application
Semi Parametric Embedding [2]	Leverage both content-based and co-occurrence based item features to embed all items from a co-occurrence matrix.	Item-item recommendation for consumers (e.g., Amazon, Alibaba)
HS2Vec [3]	Leverage herb-drug, herb-herb, and drug-drug co-occurrences and metapaths to embed herbs and drugs in a shared latent space.	Drug-herb discovery for Traditional Chinese Medicine
Weighted-Word Vector Algorithm for Knowledge Graphs [4]	Calculate positive and negative fact entity triplets of form (subject, predicate, object) and use a triplet loss to embed the individual entities in a shared latent space.	Knowledge base creation, search engine creation (example: Google's Knowledge Graph)
DynGAN-LSTM [7]	Learn the embedding of a node's neighborhood vector at time t via an adversarial approach.	Dynamic link prediction and graph reconstruction
tNodeEmbed [6]	Learn the embedding of a node at time t.	Dynamic link prediction and node classification
Network Embedding with Community Structure [5]	Learn the static embedding of a node using its community and structural information in the network.	Static link prediction and node classification.

The **W** matrices are the weight matrices for each adjacency matrix. The l denotes the proximity. For example, **W** denotes the weight matrix for the 2nd proximity adjacency matrix. A^2 is the proximity matrix for the 2nd order proximity of **A**. It is found by squaring **A**.

The authors incorporate high-order proximity into their network embedding by performing a matrix factorization on matrix **P**. In other words, they are trying to find to matrices **U** and **V** such that the expression $||\mathbf{P} - \mathbf{V}\mathbf{U}^T||_2$ is minimized. Afterward, more matrix operations are applied to create embeddings for the overall "community" structure of the original graph. The authors do this because they claim that by doing so, they can create better node embeddings.

H. Graph Embedding: Discussion

In this section, graph embedding use cases and applications were discussed. First, we explained basic definitions and applications. Then, we discussed several domains of graph embedding and some of the latest algorithms associated with each one. For recommendation systems we discussed *SPE* [2] and *HS2Vec* [3]. We then introduced knowledge graphs and *WWV-KG* [4], an unsupervised method for improving knowledge graphs with text data. For dynamic graph embedding, *tNodeEmbed* [6] and *DynGAN* [7] were discussed. Lastly, we discussed the importance of retaining proximity information in graph algorithms and introduced an algorithm called *NECS* [5]. Table II shows the different graph embedding algorithms covered in this section.

This concludes the graph embedding section of the survey. The following section will explain the approaches within the fields of computer vision.

III. EMBEDDING IN COMPUTER VISION

The concepts of embedding are widely used in computer vision applications. In this section, we present the application of embedding concepts in cross-modal information retrieval, video and image segmentation, and zero-shot learning.

A. Cross-Modal Retrieval

Word embedding models, such as Word2Vec [13], were widely adopted in information retrieval tasks. The next logical step was to use embedding for retrieving data in different modalities, such as text, images, and videos. These frameworks mostly deal with captioning visual data or retrieving visual data using text [14]–[16]. Some works have more specific goals, such as food recipe retrieval from food images [12] or captioning images in different languages [17]. All these frameworks embed the visual feature vectors and text vectors into a shared space. The goal is to learn the embedding functions, such that relevant visual and text embeddings are mapped close to each other while keeping irrelevant embeddings as far as possible. A sample structure of a cross-modal retrieval framework is shown in Figure 6.

The first step of cross-modal retrieval frameworks is encoding visual and text data into lower-dimensional feature vectors. There are two types of visual data: images and videos. Both can be encoded into lower-dimensional feature vectors with convolutional neural networks. Video features can be obtained by adding a temporal dimension to the convolution filters or by combining visual features of individual frames. Text data are usually encoded via recurrent neural networks, such as LSTM, GRU, and their variations. The input data to the network are word vectors that can be computed with widely-used word-embedding models, such as Word2Vec or GloVe. One framework in the literature implements a novel word-embedding model, where it encodes the words on a character-based level [17]. The authors represent each character as a 24-dimensional vector and a single word is represented as a matrix. This matrix is passed through 2 fully connected layers and returned as a word-level embedding, which is ready to be consumed by text encoding RNN models. This kind of word-level embedding does not require extra space for thousands of word vectors as it depends on the number of letters in the alphabet and the size of fully connected layers. In cross-modal retrieval, visual and text feature vectors are further embedded

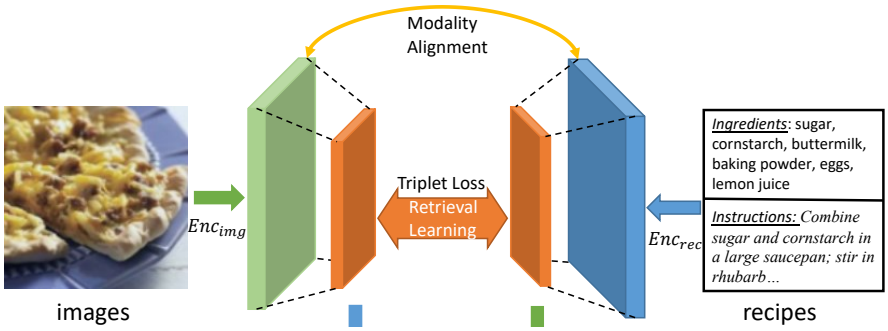

Figure 6. The framework of an image captioning model from [12].

into the shared embedding space. Most approaches use multi-layer perceptrons for the embedding step.

The state-of-the-art information retrieval frameworks incorporate a pairwise ranking loss function, also known as triplet loss, in their learning process. Cross-modal retrieval frameworks are not exceptions. The canonical equation for the pairwise ranking loss in cross-modal retrieval looks as follows:

$$\mathcal{L} = \sum_{i,j,k} \max(\gamma + d(f_{v_i}, g_{t_j}) - d(f_{v_i}, g_{t_k}), 0) \qquad (4)$$

The f_{v_i} represents the anchor visual embedding, g_{t_j} represents positive text embedding, and g_{t_k} represents negative text embedding. The function $d(x, y)$ computes the distance between two vectors. Some papers use a similarity function instead of a distance function and put the opposite sign in front. The second term would look identical with text embedding being the anchor and visual embeddings being the positive and negative instances.

It is important to mention that there are some modifications applied to this loss function in the literature to meet specific goals. For instance, Hao *et al.* adopt a hard sample mining strategy when computing triplet loss [12]. The idea is to give more preference to the most distant positive instances and to the closest negative instances. This way, they try to tackle the divergence of the loss caused by a high variation of food images for the same recipe. Wehrmann *et al.* also address the issue of quantitative dominance of weak negative samples (i.e., already far from the anchor) over hard negative samples [17]. They added a triplet loss term that is based on the hard negatives (Max of Hinges) along with the regular pairwise ranking loss term (Sum of Hinges):

$$\mathcal{L} = \lambda^{\epsilon} * \mathcal{L}_{sum} + (1 - \lambda^{\epsilon}) * \mathcal{L}_{max} \qquad (5)$$

They multiply the Max of Hinges loss with a factor that is growing over time, making the hardest negative example more significant after each iteration.

Miech *et al.* introduced another modification to ensure that the embedding model would be able to focus on the more important aspects of visual features [15]. Their goal was to train a model that would retrieve captions for instructional videos, which tend to have many different actions but with a constant background (e.g., cooking actions in the same kitchen). Therefore, they applied an intra-video negative sampling. Authors select the negative clip-text pairs, such that at least half of them came from the same video, making the irrelevant visual features to be ignored by embedding model. All mentioned methods were introduced to improve the quality of pair selection in a pairwise ranking loss. On the other side, the pairing of samples can be also non-trivial, especially for unsupervised learning. Laina *et al.* presented one example for pairing unlabeled sentences [14]. As their goal was unsupervised image captioning, they first defined a universal set of concepts that is obtained from the intersection of visual and semantic concepts. Then, the set of negative sentences is selected such that the sentences do not have any common visual concepts with the anchor, while the positive examples must have at least 2 common visual concepts. Following that, they selected positive pairs with probabilities that are proportional to the number of common concepts.

Another important aspect of training cross-modal embedding is to preserve the structure of the embedding space. There are multiple ways of doing that, but the most widely used methods are additional triplet loss terms and adversarial learning that keep the structure of the embedding space within each modality. Wray *et al.* introduced one example of an application of additional triplet loss [16]. Authors use 4 triplet loss functions for learning embedding functions: 2 cross-modal and 2 within-modal. Within-modal loss functions computed the triplet loss of video-video and text-text pairs to ensure that the neighborhood structure for each modality was preserved. Wehrmann *et al.* wanted to learn to caption images in different languages, so they added a triplet loss that was responsible for decreasing the distances between sentences with identical semantics but in different languages while pushing away sentences with different semantics [17]. Hao *et al.* used generative adversarial networks that generate food images from recipe embedding and classify the ingredients from the image embedding to make sure that the resulting embeddings are representative enough [12]. In contrast to the aforementioned methods that learn the embedding structure in parallel with cross-modal embedding, Laina *et al.* first build the joint embedding space only with text modality [14]. They use an encoder-decoder model to map the sentences into the

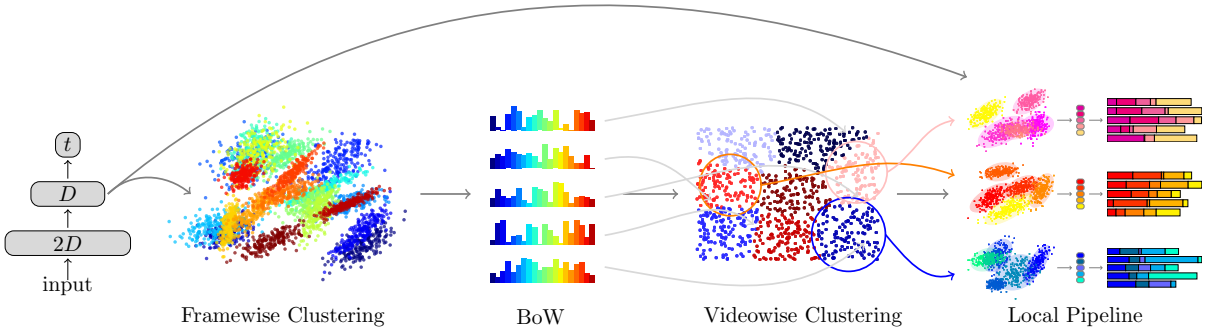

Figure 7. The framework of the unsupervised action class algorithm as presented in [18].

embedding space and learn it via triplet loss. After building this "semantic" space, they learn a translator that translates the visual features into this space, returning the visual embedding that can be decoded into an image caption.

B. Segmentation

Embedding also showed its potential in the area of segmentation, which is not surprising with the fact that embedded features are quite suitable for clustering. The literature suggested 2 types of segmentation using the concepts of embedding. The first type is frame-level embedding, where each individual frame of the video gets embedded into a low-dimensional vector, making it possible to cluster frames and segment videos into clips that represent single actions [18]. The second type is pixel-level segmentation, where pixels in the image or in the frame of a video are embedded individually and clustered to segment foreground objects [19] or text instances [20] from the background.

Kukleva *et al.* introduced the way to segment untrimmed video in an unsupervised fashion, making it much easier and cheaper to collect annotated video datasets that are in high demand for action recognition models [18]. The authors first train a model that predicts the relative timestamp of each frame and use its last layer as the embedded features of the frames. They justify this strategy by the fact that high-level activities mostly preserve the order of sub-actions with respect to each other. The features of all frames in the entire dataset are then clustered in the embedding space to represent each video as a bag-of-words. Then videos are clustered into some number of video sets, where each set is further clustered into the cluster that represents a single action label. The clusters are assigned with the mean timestamp of all embedded features in the cluster, allowing the model to order the actions. By computing the probability of each frame to belong to each cluster via Gaussian distribution, the model then temporally segments videos, such that the likelihood of the sequence of actions that consist of the sequence of labeled consecutive frames is at its maximum. The framework is represented as in Figure 7.

In a similar way, Li *et al.* pre-train a model that embeds every pixel in the image [19]. The authors wanted to segment foreground objects in the videos, but only using the model that was trained on static images. They use the similarity scores between the pixel embeddings to able to extract candidate seeds that are not on edges and are diverse. They also use the embedding graph, where the pixels are connected to their neighbors and the edges are proportional to the Euclidean distance between the vertices. The embedding graph is used to split the image into regions around the seeds, which is assigned with an optical flow vector averaged over the region. With flow vectors, they compute the motion saliency of each candidate foreground seed with respect to the background seeds. Finally, they compute the foreground scores with motion saliency and objectness scores for each seed and link the seeds with seeds in other frames using similarity scores. They use the similarity scores again to assign pixels to the foreground cluster. This algorithm has multiple steps for extracting seeds, computing their foreground scores, and linking to the other seeds. We can see that all steps are primarily derived from the similarity scores between pixels, which are computed via embedded vectors of those pixels. The authors claim that this method is a more robust way to segment video frames in comparison with segmentation using the objectness scores of each pixel, which usually struggles when there are two or more objects with high objectness scores.

The last segmentation method in the literature aims to segment text instances from the images [20]. The authors compute pixel-level embeddings for the image and use it in addition to the full map and center map segmentation networks. They introduce a Shape-Aware loss function to learn the embedding function. It has 2 components: variance loss and distance loss. Variance loss tries to minimize the distance between pixel embedding of one instance and the mean embedding vector of that instance. That distance is multiplied by a weight that is proportional to the ratio of the longest side length of the text to the longest side of the image. That means, the longer the text is, the function will gather pixels of that text more significantly. Distance loss tries to maximize the distance between mean embedding vectors of 2 different instances. In this case, that distance between 2 instances becomes a negative term. That term is also multiplied by a weight factor. That weight factor becomes smaller if those instances are close to each other,

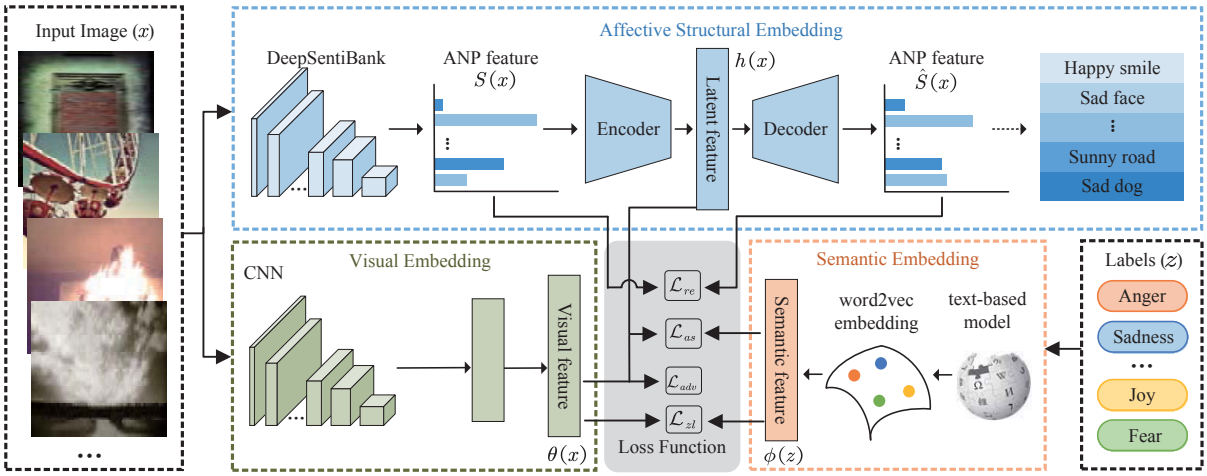

Figure 8. Framework for zero-shot emotion recognition from [21].

making the negative distance term less significant, which in turn makes the overall loss larger, thus encouraging instances that are close to each other to have more distant embeddings. The mentioned weight factors are responsible to balance the pushing and pulling force accordingly. Authors also claim that it makes it easier to distinguish 2 instances that are just one or two pixels away from each other. Given the full map and center map, the algorithm decides whether the pixel that is outside of the center map and inside the full map should be added to the center map. The pixel will be assigned to the center map if its embedded vector is close enough to the average embedding of the center map.

C. Zero-Shot Learning

Current state-of-the-art image and video classification models are implemented by extracting visual features via CNN and comparing it with a one-hot vector representation of the class label. The problem of such methodology is that there exists a significant gap between visual features and class labels, which is the biggest obstacle for zero-shot learning. This gap causes the models to overfit to the training data, which doesn't let them perform well when they see instances from novel classes. The first step for eliminating this gap was the integration of class attributes into the training process. This is particularly relevant for the classification of objects that can be described by the finite set of features (e.g., animals, birds). The standard procedure was to pass the visual data through a multi-label classifier that would identify the existence of the attributes in the instance. Then, the final class prediction would be the category that has a predicted set of attributes. The main limitation of this method is that the visual instance must have the features from that finite set of attributes, which does not generalize well and requires additional human annotation. On the other hand, the frameworks from the literature show that those shortcomings can be mitigated with continuous embedding space.

Zhan *et al.* suggest that zero-shot learning methods can help in the emotion recognition from images, which is becoming a more difficult task due to the developments in psychology that make the basic emotion categories more fine-grained [21]. The authors use a pre-trained DeepSentiBank model to extract the adjective-noun pair (ANP), which will be used as affective features that will bridge the gap between visual features and semantic labels. They extract visual features via a deep CNN and semantic features via a Word2Vec model. Both features are embedded into the same space as ANP features with a fully connected layer. At this point, this methodology looks quite similar to the cross-modal retrieval problem, except the fact that the learning methods are different [12], [14]–[17]. Unlike retrieval solutions, Zhan *et al.* do not incorporate pairwise ranking loss. Instead, they align the semantic features to both visual and ANP features via squared error function. In addition, the embedded space is defined by the affective ANP features. To preserve the affective embedding structure, authors use adversarial learning to make sure that embedded visual features preserve the structure established by ANP features. With the finite set of classes, they pre-compute the embedded vectors for all labels and find the nearest neighbor for classification. The overall architecture of this framework is depicted in Figure 8.

Zhu *et al.* propose a Generalized Zero-Shot Learning method that is agnostic to both unseen images and unseen semantic data during training [22]. They list 2 challenges in GZSL: visual2semantic gap and semantic2visual gap. Visual features, such as the final layers of deep networks are high-dimensional and don't have good semantic representation. On the other hand, semantic features are not visually meaningful and contain noisy components. The authors propose three concepts to improve GZSL. The first concept is visually semantic embedding. They take the high-dimensional visual features and break them down into some arbitrary number of parts using a multi-attention model. The obtained part vectors are then modeled as the Gaussian Mixture Models, where each mixture component of one part vector represents the probability of that part to be of some particular type. This

embedding can be visualized as a 2-dimensional matrix, where rows represent the parts, columns represent the types, and each entry represents the probability of part at this row to be the type of this column. The second concept is 3-Node Graphical Model, where they establish the 3-way connection $(X \leftrightarrow Y, X \leftrightarrow S, Y \leftrightarrow S)$ where X represents the input image, Y represents its label, and S represents the semantics of that label Y, given that semantic information and labels have a one-to-one mapping. This is quite different from the traditional relationship in zero-shot learning $(X \leftrightarrow S \leftrightarrow Y)$. They explain this by the fact that semantic information is not fully visual, visual information is not fully semantic, and class labels are not fully captured neither by visual features nor by semantic features. They also introduce a Visual Oracle Supervision, which is a ground truth matrix of part-types for each image. This is the alternative to the regular semantic oracle, where the semantic attributes (not to confuse with labels) were used for supervision. Such kind of supervision has much less noise than the regular semantic information and has discriminative capabilities during training. They wanted to see how the reduction of semantic noise affects performance.

D. Discussion

The described literature demonstrate that there are many ways to incorporate embedding concepts into computer vision frameworks. The ability to encode text via word embedding models and recurrent neural networks made it possible for researchers to come up with a new application for visual features extracted from deep CNN models, which is known as cross-modal retrieval. By embedding visual and text features into a shared vector space, they were able to leverage the alignment capacity of pairwise ranking loss to build structured embedding space and learn embedding functions that provide low-dimensional representations for high-dimensional visual data. This in turn makes it possible to accurately retrieve text from visual data for video/image captioning [14]–[17] or food recipe retrieval [12]. This is not to mention that a well structured embedding space could be a significant step towards unsupervised captioning [14], as well as captioning in different languages by making it language invariant [17].

The continuous nature of embedded features also makes them suitable for clustering, which is actively used in segmentation problems. We discovered how learning good embedding functions for video frames makes it convenient to segment untrimmed videos and label each segment with the most likely action class in an unsupervised fashion [18]. Some frameworks actively use pixel-level embeddings to compute the similarities and distances between each pair of pixels in the images to further use them as the decision criteria for including them into the segments [19], [20]. For instance, Li *et al.* describe a complex Video Object Segmentation framework whose core component is the similarity scores between embeddings [19].

In zero-shot learning scenario, embedded features can effectively bridge the gap between visual features and semantic labels [21], [22]. Zhan *et al.* define the structure of the embedding space with affective features that play significant role in emotion recognition [21]. The overall structure of their framework is somewhat similar to the cross-modal retrieval frameworks, though the learning process is quite different. Zhu *et al.* showed that embedding visual data into more compact embedding space that is also visually semantic is quite beneficial [22]. We would want to highlight one observation that the authors made. They found that using regular semantic attributes (e.g., words) can be quite noisy and bring unnecessary information. Their experiments, where they substituted semantic oracle with visually semantic oracle in their model and other GZSL models, showed that this method of reducing semantic noise can significantly improve the performance. This is a very interesting observation, which proves that some semantic attributes are indeed noisy, while the latent visually semantic features do not provide any extra information.

IV. Conclusion

In this survey paper, we summarized and discussed recent works in graph embedding, as well as the frameworks that use embedding in different areas of computer vision. Those frameworks demonstrate the variety and flexibility of embedding application.

For graph embedding, we explored several domains. We first discussed graph embedding approaches for recommendation systems. For general item recommendation, we examined the Semi-Parametric Embedding algorithm [2], and for drug recommendation we examined HS2Vec [3]. We then introduced the knowledge graph domain and the WWV-KG algorithm [4]. We discussed potential future work in which the WWV-KG and the HS2Vec algorithms could be combined to create a drug-discovery knowledge graph. Next, we shifted our focus to the dynamic graph algorithms tNodeEmbed [6] and DynGAN [7]. We observed their similarities, differences, and how they could be further explored in future research. Lastly, we observed how each graph embedding algorithm relies upon proximity preservation, which led us into a discussion of our final graph embedding algorithm, NECS [5].

For embedding in computer vision, we discussed the current frameworks that employ embeddings for cross-modal retrieval, video segmentation, object segmentation, and zero-shot learning. We observed how they build embedding spaces, learn low-dimensional visual data representations, and cluster the frames and pixels by leveraging the similarities and distances between embedded vectors. It is also worth to mention that many of those frameworks make significant steps towards semi-supervised and unsupervised learning [14]–[16], [18], [19], [21], [22].

Embedding is a powerful tool for dealing with unstructured data in many domains. There are a myriad of ways to utilize it, and this paper has provided several of them. Although this survey is not exhaustive, it is our hope that it will serve as a guide for researchers attempting to navigate the embedding literature.

REFERENCES

[1] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[2] P. Hu, R. Du, Y. Hu, and N. Li, "Du., r.: Hybrid item-item recommendation via semi-parametric embedding," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 10–16, 2019.

[3] C. Ruan, J. Ma, Y. Wang, Y. Zhang, and Y. Yang, "Discovering regularities from traditional chinese medicine prescriptions via bipartite embedding model," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pp. 3346–3352, International Joint Conferences on Artificial Intelligence, 2019.

[4] N. Veira, B. Keng, K. Padmanabhan, and A. Veneris, "Unsupervised embedding enhancements of knowledge graphs using textual associations," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 5218–5225, AAAI Press, 2019.

[5] Y. Li, Y. Wang, T. Zhang, J. Zhang, and Y. Chang, "Learning network embedding with community structural information," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 2937–2943, AAAI Press, 2019.

[6] U. Singer, I. Guy, and K. Radinsky, "Node embedding over temporal graphs," *arXiv preprint arXiv:1903.08889*, 2019.

[7] A. Maheshwari, A. Goyal, M. K. Hanawal, and G. Ramakrishnan, "Dyngan: Generative adversarial networks for dynamic network embedding," 2019.

[8] A. Chirkina and B. Rankov, "A recommender system for private banking," 08 2018.

[9] C. Jinyin, X. Xu, W. Yangyang, and H. Zheng, "Gc-lstm: Graph convolution embedded lstm for dynamic link prediction," 12 2018.

[10] A. Singhal, "Introducing the knowledge graph: Things not strings.," 2012.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Advances in Neural Information Processing Systems*, vol. 3, 06 2014.

[12] H. Wang, D. Sahoo, C. Liu, E.-p. Lim, and S. C. Hoi, "Learning cross-modal embeddings with adversarial networks for cooking recipes and

[21] C. Zhan, D. She, S. Zhao, M.-M. Cheng, and J. Yang, "Zero-shot emotion recognition via affective structural embedding," in *Proceedings* food images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11572–11581, 2019.

[13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[14] I. Laina, C. Rupprecht, and N. Navab, "Towards unsupervised image captioning with shared multimodal embeddings," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 7414–7424, 2019.

[15] A. Miech, D. Zhukov, J.-B. Alayrac, M. Tapaswi, I. Laptev, and J. Sivic, "Howto100m: Learning a text-video embedding by watching hundred million narrated video clips," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2630–2640, 2019.

[16] M. Wray, D. Larlus, G. Csurka, and D. Damen, "Fine-grained action retrieval through multiple parts-of-speech embeddings," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 450–459, 2019.

[17] J. Wehrmann, D. M. Souza, M. A. Lopes, and R. C. Barros, "Language-agnostic visual-semantic embeddings," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5804–5813, 2019.

[18] A. Kukleva, H. Kuehne, F. Sener, and J. Gall, "Unsupervised learning of action classes with continuous temporal embedding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12066–12074, 2019.

[19] S. Li, B. Seybold, A. Vorobyov, A. Fathi, Q. Huang, and C.-C. Jay Kuo, "Instance embedding transfer to unsupervised video object segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[20] Z. Tian, M. Shu, P. Lyu, R. Li, C. Zhou, X. Shen, and J. Jia, "Learning shape-aware embedding for scene text detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4234–4243, 2019.
of the IEEE International Conference on Computer Vision, pp. 1151–1160, 2019.

[22] P. Zhu, H. Wang, and V. Saligrama, "Generalized zero-shot recognition based on visually semantic embedding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2995–3003, 2019.