MELL: Effective Embedding Method for Multiplex Networks

Ryuta Matsuno
Department of Computer Science, School of Computing
Tokyo Institute of Technology
Meguro, Tokyo, Japan
ryutamatsuno@net.c.titech.ac.jp

Tsuyoshi Murata
Department of Computer Science, School of Computing
Tokyo Institute of Technology
Meguro, Tokyo, Japan
murata@c.titech.ac.jp

ABSTRACT

Network embedding is a method for converting nodes in a network into low dimensional vectors, preserving its structure and the similarities among the nodes. Embedding is widely used in many applications, e.g., social network analysis and knowledge discovery. Because of its wide usage, many studies have been proposed, such as DeepWalk, LINE and node2vec. These works are designed for single-layer networks, however, real world networks often possess not just one, but multiple types of connections. Hence it is more appropriate to represent them as multiplex networks, which consist of multiple layers each of which represents one type of relationship. Embedding multiplex networks is difficult because all layer structures have to be taken into consideration.

In this paper, we propose MELL, a novel embedding method for multiplex networks, which incorporates an idea of layer vector that captures and characterizes each layer's connectivity. This method exploits the overall structure effectively, and embeds both directed and undirected multiplex networks, whether their layer structures are similar or complementary. We focus on link prediction tasks and test our method and other baseline methods using five data sets from different domains. The results show that our method outperforms all of the baseline methods for all of the data sets.

CCS CONCEPTS

• **Computing methodologies** → **Unsupervised learning**; **Learning latent representations**; • **Theory of computation** → *Social networks*;

KEYWORDS

Multiplex network, Network embedding, Link prediction, Network analysis

ACM Reference Format:
Ryuta Matsuno and Tsuyoshi Murata. 2018. MELL: Effective Embedding Method for Multiplex Networks. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3184558.3191565

1 INTRODUCTION

Network embedding aims at converting nodes in a network to lower dimensional vectors. The embedding vectors are learned in order to preserve some particular information such as structural connectivity, structural distances or similarities among nodes [8]. Due to its wide usage, many methods are proposed recently, such as DeepWalk [13], LINE [16], node2vec [9] and APP [18]. The common tasks to test embedding methods are visualization, multi-class labeling and link prediction. Link prediction is useful for many applications, e.g., friend recommendation in online social networks and knowledge discovery in semantic networks. Thus we focus on link prediction task in this paper.

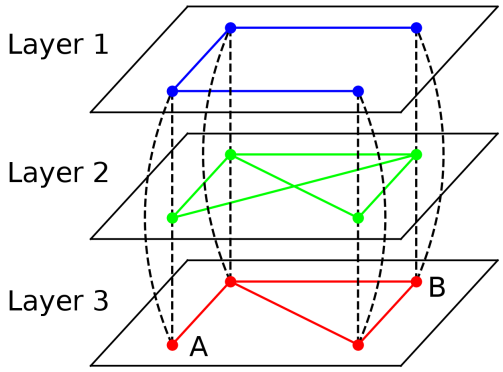


Figure 1: A toy image of undirected multiplex network. This multiplex network has three layers, and they share four nodes. Continuous lines illustrate the edges in each layer. Two nodes connected by a dashed line are the same corresponding nodes; they are counterparts of each other.

Existing embedding methods are mainly for single-layer networks, however, networks in real world are often represented as multiplex networks [7]. Multiplex network is a multi-layer network, where the layers share the same set of nodes [14]. A multiplex network represents different types of connectivity between one single set of nodes. For example, online social networks used by the same people are expressed as a multiplex network where each layer illustrates one online social network service. In the case of genetic networks, the multiple types of interactions among genes are described as a multiplex network.

In multiplex networks, connectivity in layers affects each other and that makes it difficult to analyze each layer independently. **Figure 1** shows a toy image of an undirected multiplex network. This multiplex network has three layers and the layers share four nodes. The structure of layer 2 and layer 3 are almost equal, so it is natural to predict a potential link between node A and node B in layer 3. This prediction is easy and reasonable, however, without

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3191565

layer 2, it is difficult to predict that link because of the limited structural information. Therefore, existing embedding methods for single-layer networks do not work for multiplex networks.

MTNE [17] is an existing embedding method for multiplex networks. This method jointly learns embedding vectors for each node in each layer via enforcing an extra information-sharing embedding. It predicts links in multiplex networks, but the problem is that their model seems to work only for multiplex networks where layers are similar to each other. Another approach for link prediction in multiplex networks is generative model. MULTITENSOR [4] is the one that decomposes adjacency tensor of a multiplex network using the Poisson distribution. Based on the decomposition, this method regenerates the adjacency tensor and uses the tensor for link prediction. However, it is not always the Poisson distribution that generates multiplex networks.

In this paper, we propose MELL (an abbreviation of Multiplex network Embedding via Learning Layer vectors). It embeds each layer into a lower dimensional embedding space, and enforces these embeddings to be close to each other in order to share each layer's connectivity among embeddings. We also incorporate a novel idea of layer vector that captures and characterizes each layer's connectivity. MELL learns embedding vectors and layer vectors at the same time using all of the layer structures, and based on these embeddings and layer vectors, this method calculates edge probabilities for link prediction. Thanks to the enforcing and the layer vectors, MELL works well for link prediction whether layer structures are similar or complementary. We test our method and four other baseline methods for link prediction tasks using five data sets from different domains : social, co-authorship, transportation, neuronal and genetic domains. The experimental results show that our method outperforms all of the baseline methods in all of the data sets. At maximum, the AUC score of our method is 25% higher than that of the baseline methods.

2 RELATED WORK

In this section, we introduce related works: network embedding and generative model.

2.1 Network Embedding

Network embedding is a method for embedding a network to a lower dimensional space by converting nodes in the network to vectors in the embedding space. One of the well-known approaches is the one based on random walks [3], e.g., DeepWalk [13], LINE [16], node2vec [9] and APP [18]. These methods use sampled paths obtained by random walks for updating and optimizing embedding vectors. We use APP as a representative single-layer embedding method for the experiments because APP is the latest work among these works, and APP is the best method for link prediction. Also APP can embed both undirected and directed networks.

For multiplex networks, MTNE, Multi-Task Network Embedding method [17] is one of the representative methods. It extends a simple embedding method to multiplex networks, and jointly learns embedding vectors for each layer in a multiplex network via enforcing an extra information-sharing embedding. The authors of MTNE assume that the same node will expose similar or complementary characteristics in layers. Enforcing information-sharing embedding is the way to share the characteristics in all layers. The embedding process is as follows: i) MTNE embeds each layer to the lower dimensional space, ii) it calculates the information-sharing embedding, iii) it optimizes embedding vectors using the information-sharing embedding, iv) it repeats the step ii) and iii) until embedding vectors converge. The authors propose two ways for the information-sharing embedding: MTNE-C, which shares a common embedding, and MTNE-R, which shares a consensus embedding.

They test MTNE methods through network visualization, link prediction and multi-label classification. MTNE-R and MTNE-C work better than other methods including DeepWalk and LINE. They also show that the best method for link prediction is "MTNE-R Consensus" (MTNE-R Cons. for short), a variant of MTNE-R. MTNE-R uses the consensus embedding merely for optimizing each layer's embedding, and uses each layer's embedding for predicting links in each layer. On the other hand, MTNE-R Cons. uses the consensus embedding to predict links in all layers. The consensus embedding is similar to the average embedding among all layers' embeddings, and it predicts links well in the experiments. However, MTNE-R Cons. ignores the differences among layers because it uses exactly the same embedding vectors for link prediction in all layers. For multiplex networks whose layer structures are different, MTNE-R Cons. does not work well. MTNE-C and MTNE-R also do not work because the information-sharing embedding cannot improve the overall embeddings. The problem of MTNE methods is that they assume similar layer structures. Indeed, social networks tends to have similar layer structures. So MTNE methods seem to work well for social networks but not for other networks where the structural similarity is not guaranteed, e.g., transportation networks and genetic networks. Another problem is that MTNE methods are designed only for undirected multiplex networks, so these methods cannot be used for directed multiplex networks.

2.2 Generative Model

Another approach for link prediction in multiplex networks is generative model. Generative models regard adjacency matrices for a multiplex network as one tensor. Then the models try to find the hidden parameters that generate the tensor assuming a particular probabilistic distribution.

MULTITENSOR [4] is a generative model used for multiplex networks. The authors extend the mixed-membership stochastic block model to multiplex networks. They assume that the layers share underlying common community structures, which determine the connectivity in all layers.

Formally, MULTITENSOR is similar to Poisson tensor factorization models. MULTITENSOR decomposes the adjacency tensor into three matrices, two membership matrices and one affinity matrix. After the decomposition, it regenerates the adjacency tensor, and uses the tensor for the tasks such as link prediction. This method can be used for directed and undirected networks. Link prediction experiments are conducted on three real networks, and MULTITENSOR performs the best in the experiment.

However, there is no guarantee that there are underlying community structures in any multiplex networks. Also there are no

guarantee that any multiplex networks are generated based on the Poisson distribution.

3 OUR METHOD

In this section, we explain our method, MELL. Firstly, we define notations and terms, and then we explain a basic embedding method for single-layer networks. After that, we illustrate the idea of our method and its algorithm.

3.1 Notation

In this paper, we use v for nodes, lower case letters (i.e., x except for v) for scalar values, lower case roman-bold letters (i.e., $\mathbf{x}$) for vectors, upper case roman-bold letters (i.e., $\mathbf{X}$) for matrices or tensors, upper case calligraphic letters (i.e., $\mathcal{X}$) for sets. The notation $\|\cdot\|_F$ denotes the Frobenius norm. A term $\mathbf{a}\cdot\mathbf{b}$ denotes the inner product of the vectors $\mathbf{a}$ and $\mathbf{b}$. A term $\mathbf{a}^\mathsf{T}$ denotes the transpose of the vector $\mathbf{a}$.

3.2 Terminology Definition

The networks studied in this paper are all multiplex networks. Multiplex network is defined as *Definition 3.1* .

Definition 3.1 (Multiplex Network). A multiplex network consists of a set of networks. Each of the networks is called a "layer" and all layers have the same set of nodes. Specifically, a multiplex network $\mathcal{G} = \{G^1, ..., G^l, ..., G^L\}$, where $G^l = (\mathcal{V}^l, \mathcal{E}^l)$ denotes the l-th layer in $\mathcal{G}$. L is the total number of layers. $\mathcal{V}^l$ denotes the set of nodes and $\mathcal{E}^l$ denotes the set of edges in the l-th layer. In other words, $\mathcal{E}^l \subset \mathcal{V}^l \times \mathcal{V}^l$ and $\mathcal{V}^l = \{v_1^l, ..., v_i^l, ..., v_N^l\}$, where v_i^l denotes the i-th node in the l-th layer. N denotes the number of nodes. For v_i^l, all i-th nodes in all other layers are its counterparts, e.g., $v_i^1, ..., v_i^L$ are the counterparts of each other. We use M for the total number of all existing edges, i.e., $M = \sum_l^L \left|\mathcal{E}^l\right|$. For undirected networks, $(v_i^l, v_j^l) = (v_j^l, v_i^l)$, while for directed networks, $(v_i^l, v_j^l) \neq (v_j^l, v_i^l)$. We use the term "head" for a node where a directed edge is from, and we use "tail" for a node where a directed edge is to, e.g., for (v_i^l, v_j^l), v_i^l is the head, and v_j^l is the tail.

In this paper, we study a method for predicting missing or potential edges in multiplex networks. This problem is defined as follows.

Definition 3.2 (Link Prediction in Multiplex Network Problem). Given a multiplex network $\mathcal{G}$, "link prediction in multiplex network problem" aims at inferring the missing or potential edges in any layers in $\mathcal{G}$. Formally, based on the given multiplex network $\mathcal{G}$, the objective is to generate an edge probability function $p : \mathcal{V}^l \times \mathcal{V}^l \to [0, 1]$ $(l = 1, ..., L)$. The output value of p shows how likely the edge between input nodes will be formed.

3.3 Inner Product Method

We introduce one of the basic embedding methods for single-layer networks, the inner product method [12] [17], which our method in this paper is based on. This method is also adopted in MTNE.

In the inner product method, the edge probability that there exists an edge between node v_i and v_j is calculated as follows:

$$p(v_i, v_j) = \frac{1}{1 + \exp(-\mathbf{v}_i{}^\mathsf{T} \cdot \mathbf{v}_j)}, \tag{1}$$

where $\mathbf{v}_i$ and $\mathbf{v}_j$ denote the embedding vectors for v_i and v_j respectively. Given a network, output values of p for connected node pairs should be close to 1, and output values of p for unconnected node pairs should be 0. Thus this method learns the embedding vectors to maximize the likelihood of generating the original network. The likelihood is as follows:

$$\prod_{(v_i, v_j) \in \mathcal{E}} p(v_i, v_j) \prod_{(v_i, v_j) \notin \mathcal{E}} \Big(1 - p(v_i, v_j)\Big), \tag{2}$$

where $\mathcal{E}$ denotes the existing edge set. Usually for easy calculation, the log likelihood is used for the loss function, and a regularization term for the embedding vectors is added to it. The final loss function, which should be minimized, is as follows:

$$\begin{aligned} Loss\,(\mathcal{E}; \lambda) = &- \sum_{(v_i, v_j) \in \mathcal{E}} \log p(v_i, v_j) \\ &- \sum_{(v_i, v_j) \notin \mathcal{E}} \log \Big(1 - p(v_i, v_j)\Big) \\ &+ \lambda \|\mathbf{V}\|_F^2, \end{aligned} \tag{3}$$

where $\mathbf{V}$ denotes the matrix for embedding vectors, the i-th row in $\mathbf{V}$ is the embedding vector for v_i, λ denotes a regularization coefficient.

The inner product method embeds all nodes in the given single-layer network into the d-dimensional embedding space by minimizing Eq. (3). After the embedding, this method calculates $p(v_i, v_j)$ by Eq. (1) to predict whether the edge between v_i and v_j will be formed or not.

3.4 MELL

We propose MELL, a Multiplex network Embedding via Learning Layer vectors. MELL is based on the inner product method explained above, and we incorporate the following new ideas:

- MELL embeds nodes in each layer into a d-dimensional space, and enforces embedding vectors for the same node among layers to be close to each other in order to share all layer structures among embeddings.
- MELL learns layer vectors that characterize the layers' connectivity in order to differentiate the edge probabilities in each layer.
- For undirected multiplex networks, MELL uses two vectors for each node; the one is an embedding vector as a head, the other is an embedding vector as a tail.

The overall processes of our method are shown in **Figure 2**. MELL embeds nodes in each layer with learning layer vectors. After the embedding, it predicts links by calculating the edge probabilities.

The first idea is to embed each layer into the embedding space and enforce them to be close. By this enforcing, the embedding vectors are learned not only from one layer's connectivity but also from all other layers' connectivity as well. Specifically, MELL applies the inner product method to each layer. The standard inner

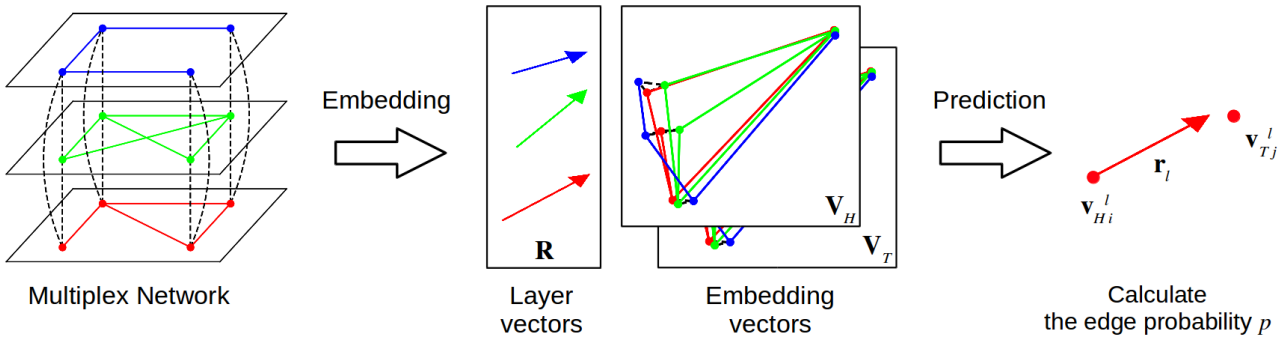


Figure 2: The overall processes for predicting links by MELL. Given a multiplex network, MELL embeds nodes in each layer into a lower dimensional space with learning layer vectors. After learning the embeddings, it predicts links by calculating the edge probabilities by Eq. (6).

product method merely learns the structural connections in one layer. Thus, in order to exploit all layers' connectivity for each embedding, MELL enforces the embedding vectors for the same node to be close to each other. MELL realizes this idea by adding a regularization term to the loss function. Formally, this term is designed as the variance of embedding vectors. The enforcing term is as follows:

$$\begin{aligned}(\text{The enforcing term}) &= \beta \mathbb{V}[\mathbf{V}], \qquad (4)\\ \text{where } \mathbb{V}[\mathbf{V}] &= \frac{1}{L}\sum_{l=1}^{L} \|\mathbf{V}[l] - \mathbb{E}[\mathbf{V}]\|_F^2,\\ \mathbb{E}[\mathbf{V}] &= \frac{1}{L}\sum_{l=1}^{L} \mathbf{V}[l],\end{aligned}$$

and β denotes the regularization coefficient, $\mathbf{V} \in \mathbb{R}^{L\times N\times d}$ denotes the embedding tensor, which contains all embedding vectors for the given multiplex network. $\mathbf{V}[l]$ denotes the $N \times d$ embedding matrix for the nodes in the l-th layer. L, N, d denote the number of layers, nodes and the embedding space's dimension, respectively. $\mathbb{E}[\mathbf{V}]$ calculates the average embedding matrix among layers and $\mathbb{V}[\mathbf{V}]$ calculates the variance of embedding vectors.

However, even though layer structures in a multiplex network are different from each other, the enforcing prevents the embedding vectors from being different. This causes poor link predictions because even if some layers have different structures, a method based on the embedding will predict an edge between the same pair of nodes in all layers with almost the same probabilities. In order to differentiate edge probabilities in each layer, we incorporate the idea of layer vector. Layer vector is a vector which determines the edge probabilities for all node pairs in the layer. Formally, the edge probability between v_i^l and v_j^l is calculated as follows using a layer vector $\mathbf{r}_l$:

$$p(v_i^l, v_j^l) = \frac{1}{1+\exp\left(-(\mathbf{v}_i^l + \mathbf{r}_l)^\top \cdot \mathbf{v}_j^l\right)}, \qquad (5)$$

where $\mathbf{v}_i^l$ and $\mathbf{v}_j^l$ denote the embedding vector for v_i^l and v_j^l respectively. $\mathbf{r}_l$ denotes the layer vector for the l-th layer. The layer vectors actually learn the layers' connectivity. By the enforcing and Eq. (5), if layers are similar to each other, the layer vectors are also similar to each other, and if they are different, the layer vectors are different as well. Therefore, layer vectors capture and characterize connectivity in each layer, comparing the structure with other layer structures.

The disadvantage of the layer vector is its asymmetric nature. The original inner product method is for undirected network because p in Eq. (1) is symmetric, i.e., $p(v_i, v_j) = p(v_j, v_i)$. In Eq. (5), $p(v_i^l, v_j^l)$ is not equal to $p(v_j^l, v_i^l)$ because the layer vector is added to the head embedding vector. This asymmetric nature is not a problem for directed networks because they require an asymmetric edge probability. To make our method applicable to undirected networks as well, we introduce two vectors for each node, one is as a head and the other is as a tail. This idea is inspired by APP [18], which employ an asymmetric edge probability like Eq. (5). Since two embedding vectors should be used in order to incorporate this idea, we redefine $p(v_i^l, v_j^l)$ as follows:

$$p(v_i^l, v_j^l) = \frac{1}{1+\exp\left(-(\mathbf{v}_H{}_i^{\,l} + \mathbf{r}_l)^\top \cdot \mathbf{v}_T{}_j^{\,l}\right)}, \qquad (6)$$

where $\mathbf{v}_H{}_i^{\,l}$ denotes the head embedding vector for v_i^l and $\mathbf{v}_T{}_j^{\,l}$ denotes the tail embedding vector for v_j^l. In directed networks, a vector as a head and that as a tail are always the same, i.e., $\mathbf{v}_H{}_i^{\,l} = \mathbf{v}_T{}_i^{\,l}$ in any l and any i.

Considering these three ideas: enforcing, incorporating layer vector, and using two vectors for undirected multiplex networks, the loss function to optimize the embedding vectors and layer vectors is designed as follows:

$$Loss(\mathcal{E}_{pos}, \mathcal{E}_{neg}; \lambda, \beta, \gamma) = - \sum_{(v_i^l, v_j^l) \in \mathcal{E}_{pos}} \log p(v_i^l, v_j^l) - \sum_{(v_i^l, v_j^l) \in \mathcal{E}_{neg}} \log \left(1 - p(v_i^l, v_j^l)\right) + \beta \left(\mathbb{V}[\mathbf{V}_H] + \mathbb{V}[\mathbf{V}_T]\right) + \frac{\lambda}{N}(\|\mathbf{V}_H\|_F^2 + \|\mathbf{V}_T\|_F^2) + \gamma \|\mathbf{R}\|_F^2, \quad (7)$$

where $\mathbf{V}_H, \mathbf{V}_T \in \mathbb{R}^{L \times N \times d}$ denote the head and tail embedding tensor respectively, which contain all head and tail embedding vectors. $\mathbf{R} \in \mathbb{R}^{L \times d}$ denotes the layer vectors. $\mathcal{E}_{pos}$ denotes the all existing edges in the given multiplex network, i.e., $\mathcal{E}_{pos} = \mathcal{E}^1 \cup ... \cup \mathcal{E}^L$. $\mathcal{E}_{neg}$ is the set of negative samples. They are uniformly and randomly sampled from all unconnected node pairs in the given multiplex network. The number of negative samples is $k|\mathcal{E}_{pos}|$ using a parameter k. The third term is basically designed as the enforcing term in Eq. (4), but is modified for using $\mathbf{V}_H$ and $\mathbf{V}_T$. β denotes the regularization coefficient for this enforcing term. λ and γ denote regularization coefficients for embedding vectors and for layer vectors respectively. The fourth term is divided by N to normalize the impact of λ.

In order to minimize *Loss*, our method learns the optimal $\mathbf{V}_H, \mathbf{V}_T$ and $\mathbf{R}$. After learning, we use Eq. (6) to predict edges. Due to the layer vectors, our method is basically for directed multiplex networks. Thus we introduce additional processes for undirected networks; we regard one undirected edge as two directed edges. Firstly, for learning embedding vectors, $\mathcal{E}_{pos}$ is set as $\mathcal{E}^1 \cup ... \cup \mathcal{E}^L \cup \mathcal{E}'^1 \cup ... \cup \mathcal{E}'^L$, where $\mathcal{E}'^l = \left\{(v_j^l, v_i^l) \middle| (v_i^l, v_j^l) \in \mathcal{E}^l\right\}$. Secondly, after learning embedding vectors, we use the average value of $p(v_i^l, v_j^l)$ and $p(v_j^l, v_i^l)$ for predicting an edge (v_i^l, v_j^l).

Our method has five parameters altogether: embedding space's dimension d, a negative sampling rate k, regularization coefficient for embedding vectors λ, regularization coefficient for the variance β and regularization coefficient for layer vectors γ. These parameters should be decided by a grid search.

We implement our method using Python and TensorFlow. We use Adam [10] for fast optimizing, but the other standard optimization methods, such as gradient descent, can be used as well. The overall MELL processes to learn $\mathbf{V}_H, \mathbf{V}_T$ and $\mathbf{R}$ are shown in **Algorithm 1**.

3.5 Complexity Analysis

The total time complexity of MELL is $O(td(kM + NL))$, where t denotes the number of iterations, k denotes the negative sampling rate, L, N, M denotes the number of layers, nodes, edges in a given multiplex network, respectively.

4 EXPERIMENTS

In this section, we explain the experiments we conducted. First, we explain details of data sets, experimental setting, baseline methods to be compared and the evaluation metric. After that we show the experimental results. We also report the parameter sensitivity of our method.

Algorithm 1 MELL algorithm

Input: Multiplex network $\mathcal{G} = \{G^1, ..., G^L\}$, embedding space's dimension d, negative sampling rate k, regularization coefficient for embedding vectors λ, regularization coefficient for the variance β, regularization coefficient for layer vectors γ

Output: embedding tensors for head and tail $\mathbf{V}_H, \mathbf{V}_T \in \mathbb{R}^{L \times N \times d}$, layer tensor $\mathbf{R} \in \mathbb{R}^{L \times d}$

```
function MELL(G, d, k, λ, β, γ)
    initialize V_H, V_T, R with random values
    E_pos ← E^1 ∪ ... ∪ E^L
    while (not converge) do
        E_neg ← negative samples, |E_neg| = k|E_pos|
        Calculate Loss(E_pos, E_neg; λ, β, γ) by Eq. (7)
        update V_H, V_T, R
    end while
    return V_H, V_T, R
end function
```

4.1 Data Set

We test our method and baseline methods using five data sets from different domains: social, co-authorship, transportation, neuronal and genetic domains. The basic statistics of the data sets are shown in **Table 1**. L, N, M denotes the number of layers, nodes and all exiting edges, respectively. All data sets are obtained from CoMuNe lab's web site[1]. Explanations of the data sets are as follows:

- CS-Aarhus [11] : This undirected multiplex social network consists of five kinds of online and offline relationships (Facebook, Leisure, Work, Co-authorship, Lunch) between the employees of Computer Science department at Aarhus.
- Pierre Auger Collaboration [5] : This undirected co-authorship multiplex network consists of 16 types of different working tasks within the Pierre Auger Collaboration. The layers represent Neutrinos, Detector, Enhancements and other tasks. This network is originally weighted, but we ignore the weights for our experiments.
- EU Air Transportation [1] : This undirected transportation multiplex network is composed by 37 different layers each one corresponding to a different airline operating in Europe. The nodes are airports, and the edges are routes, respectively.
- C.Elegans Connectome [2] [6] : This directed neuronal multiplex network, Caenorhabditis Elegans Connectome, consists of three different synaptic junctions: electric, chemical monadic, and polyadic.
- Xenopus [15] [6] : This directed genetic multiplex network consists of five types of different interactions for organisms: association, direct interaction, physical association, colocalization, and suppressive genetic interaction defined by inequality. The authors use the Biological General Repository for Interaction Data sets 3.2.108 (BioGRID, thebiogrid.org, updated 1 Jan 2014) for Xenopus laevis.

[1] https://comunelab.fbk.eu/data.php

Table 1: Properties of the data sets. L, N, M denotes the number of layers, the number of the nodes, the total number of all existing edges, respectively. Type shows the type of the data set.

Data set	L	N	M	type	directed
CS-Aarhus	5	61	620	social	undirected
Pierre Auger Collaboration	16	514	7153	co-authorship	undirected
EU Air Transportation	37	450	3588	transportation	undirected
C.Elegans Connectome	3	279	5863	neural	directed
Xenopus	5	461	620	genetic	directed

4.2 Experimental Setting

Given a multiplex network from the data sets, first we randomly split all existing edges into five parts each of which has 20% of the edges of each layer. We vary training edge rate α_t from 20% to 80% in 20% increments. Based on α_t, we pick one to four parts of the split edge parts and use them as training edges. The other parts are used as positive testing edges. This positive testing edges are hidden with other unconnected node pairs, so that methods can not tell the testing edges and truly unconnected node pairs apart. For testing, we sample truly unconnected node pairs as negative testing edges. These edges are sampled uniformly and randomly, and the number of positive testing edges and negative testing edges are the same. We use both the positive and negative testing edges for testing. We change the picking combination for the training edge parts uniformly five times, and report the average score as the final results. When α_t is 20%, this experimental setting is the same as the five-fold cross validation.

For the evaluation, we use AUC score, a well-known and standard metric for link prediction tasks.

4.3 Baseline Method

In this experiments, we test 4 existing methods and our method: APP, MTNE-R, MTNE-R Cons., MULTITENSOR and MELL. Except for MULTITENSOR, which is not an embedding method, we use the same dimension for embedding, $d = 128$ for fair comparison. For other parameters, such as regularization coefficient, we do grid search for each data set to find optimal parameters. The explanations of the baseline methods are as follows:

- APP [18] : This method embeds directed and undirected single-layer networks. In this experiment, we naively apply this method to each layer, and thus this method only uses one layer, ignoring the other layers.
- MTNE-R [17] : This method is for embedding undirected multiplex networks. It embeds each layer to the embedding space, and then it calculates a consensus embedding. Using the consensus embedding, it optimizes each layer's embedding, and uses each embedding for link prediction in each layer. We use all unconnected node pairs as negative samples for learning.
- MTNE-R Cons. [17] : This method is a variant of MTNE-R. Different from MTNE-R, this method uses the consensus embedding for link prediction for all layers, ignoring differences among layers.
- MULTITENSOR [4] : This is a generative model based on the Poisson distribution. It decomposes the adjacency tensor and uses obtained matrices for link prediction. Originally, it uses a part of layers for link prediction, however, all layers are used in this experiment for a fair comparison.

4.4 Results

We set α_t to 20%, 40%, 60% and 80%. For each α_t, we test baseline methods and our method using all data sets. Experimental results are shown in **Table 2**. The values in the table are AUC scores. For MTNE-R and MTNE-R Cons., the results on C.Elegans Connectome and Xenopus are set to "-" because they are directed multiplex networks.

First of all, APP, which totally ignores other layer structures, shows the worst scores in all data sets. For EU Air Transportation network, APP predicts links almost randomly. On the other hand, other methods, that use structural connectivity in other layers, performs better than APP.

MELL outperforms all existing methods, APP, MTNE-R, MTNE-R Cons. and MULTITENSOR. For directed networks, C.Elegans Connectome and Xenopus, MELL shows significantly higher AUC scores, compared to only one baseline method for multiplex networks. Especially for Xenopus data sets, the AUC scores of MELL are at least 20% higher than MULTITENSOR, and 25% higher at $\alpha_t = 80\%$. For undirected network, MELL also shows the higher AUC scores. These results show that MELL effectively exploits the structural connectivity in all layers in the multiplex networks.

We expect that MTNE models do not work on multiplex networks where the layer structures are not similar with each other. As we expected, MTNE-R and MTNE-R Cons. shows comparable AUC scores on CS-Aarhus and Pierre Auger Collaboration. They are a social network and a co-authorship network, respectively. Probably they have similar layer structures. On the other hand, AUC scores for EU Air Transportation are significantly inferior to MULTITENSOR and MELL. This is because transportation networks are different from social networks, and their layer structures are complementary. MTNE-R Cons., which performs better than MTNE-R in original paper's experiments, shows almost the same AUC scores with MTNE-R, and they are not remarkably different.

4.5 Parameter Sensitivity

We vary parameters of MELL, in order to test the sensitivity of the parameters. MELL requires five parameters; embedding space's dimension d, negative sampling rate k, and three regularization coefficients : λ for embedding vectors, β for the variance, and γ for layer vectors. We set $d = 128, k = 4.0, \lambda = 10.0, \beta = 1.0, \gamma = 1.0$ as default parameters, and pick up one parameter and vary that

Table 2: Experimental Results. The values in the table are AUC scores. α_t is the rate of training edges to all existing edges. Since MTNE-R and MTNE-R Cons. are for undirected multiplex networks, the scores are set to "-" in the table. The bold face is the best value for the same α_t in each data set.

data set	method	training rate α_t 20%	40%	60%	80%
CS-Aarhus	APP	0.5461	0.6630	0.7376	0.7757
	MTNE-R	0.6977	0.7788	0.7921	0.8492
	MTNE-R Cons.	0.6943	0.7645	0.8189	0.8370
	MULTITENSOR	0.6645	0.7827	0.8862	0.9071
	MELL	**0.8173**	**0.8877**	**0.9224**	**0.9417**
Pierre Auger Collaboration	APP	0.6060	0.7913	0.8894	0.9434
	MTNE-R	0.9214	0.9525	0.9765	0.9873
	MTNE-R Cons.	0.9217	0.9542	0.9775	0.9874
	MULTITENSOR	0.9451	0.9705	0.9817	0.9893
	MELL	**0.9811**	**0.9918**	**0.9956**	**0.9979**
EU Air Transportation	APP	0.4364	0.4904	0.5536	0.5828
	MTNE-R	0.5535	0.6891	0.7446	0.7817
	MTNE-R Cons.	0.5550	0.7010	0.7440	0.7683
	MULTITENSOR	0.8516	0.9041	0.9344	0.9463
	MELL	**0.9806**	**0.9897**	**0.9941**	**0.9944**
C.Elegans Connectome	APP	0.5875	0.6800	0.7287	0.7518
	MTNE-R	-	-	-	-
	MTNE-R Cons.	-	-	-	-
	MULTITENSOR	0.7167	0.8019	0.8434	0.8836
	MELL	**0.8360**	**0.9144**	**0.9430**	**0.9701**
Xenopus	APP	0.5135	0.5260	0.5536	0.5442
	MTNE-R	-	-	-	-
	MTNE-R Cons.	-	-	-	-
	MULTITENSOR	0.5469	0.5921	0.6441	0.6664
	MELL	**0.7575**	**0.8159**	**0.8818**	**0.9130**

parameter fixing others to check the parameter impact to the link prediction. The results are shown in **Figure 3**. The training rate α_t is fixed to 60% in this experiment. The same as the previous experiment, the AUC scores are the averages of five times results.

The embedding dimension d does not affect much for AUC scores compared to other parameters. The negative sampling rate k affects AUC scores of Xenopus. Xenopus prefers higher k because Xenopus is the sparsest network in the data sets. Thus in order to increase the true negatives of the link prediction results, Xenopus requires higher k. The regularization coefficient λ greatly affects the AUC scores of CS-Aarhus, C.Elegans Connectome and Xenopus. They show the highest AUC scores at 10, 100 and 1.0 respectively.

The regularization coefficient β controls how much MELL enforces the embedding vectors among layers to be close. Too small β makes embeddings not to learn the other layer structures, and too big β prevents the embeddings from being different. The results show the highest AUC scores from $\beta = 1$ to $\beta = 100$, and show lower scores at $\beta = 0.1$ and $\beta = 1000$. The regularization coefficient γ is for layer vectors, and it affects the results of Xenopus significantly. Xenopus shows the highest at 10, but the score hugely decreases at 100.

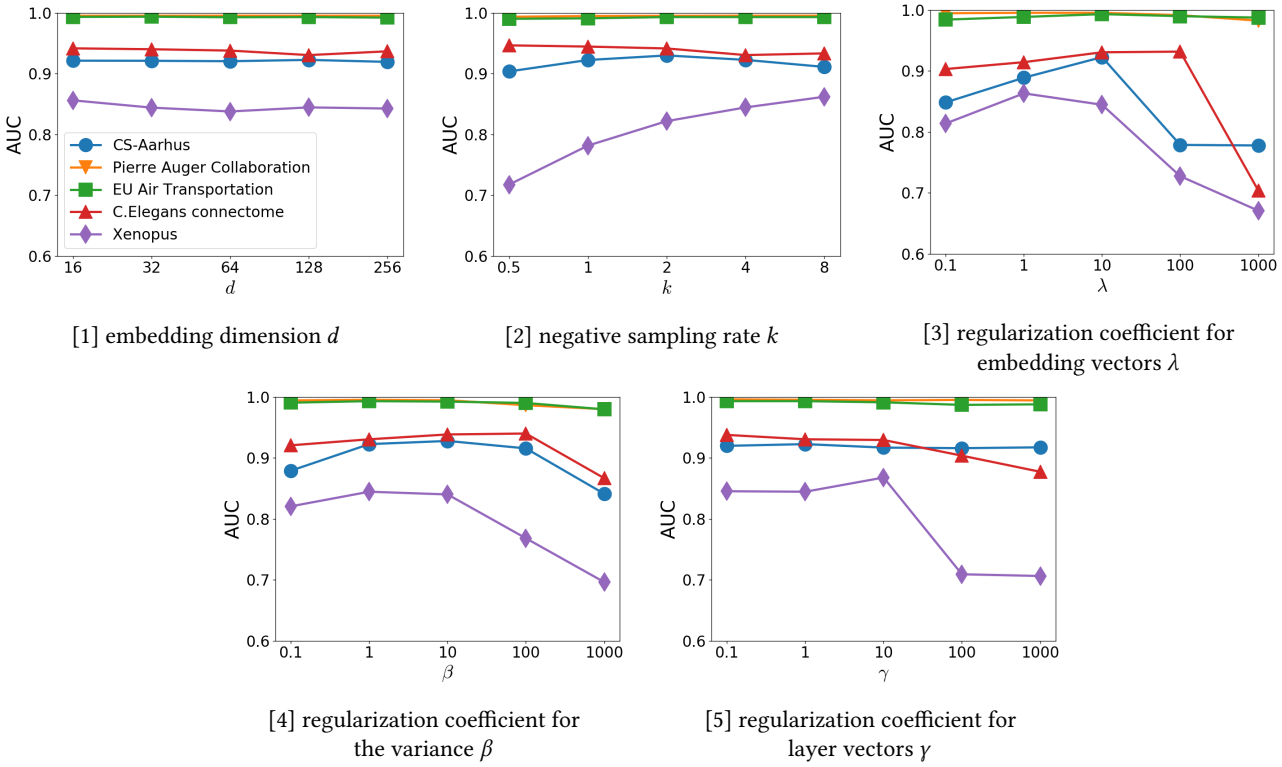
However, no parameters affect the AUC scores for Pierre Auger Collaboration and EU Air Transportation much. Their AUC scores are nearly 1.0, so that MELL can stably predict links in them, regardless of the changes of parameters.

5 CONCLUSION

In this paper, we propose a novel embedding method for multiplex networks, named MELL, which incorporates an idea of layer vector that characterizes the connectivity in a layer. MELL embeds nodes in each layer into the lower embedding space using all layer structures, and incorporates layer vectors to differentiate edge probabilities in the layers. We test our method for link prediction tasks, using five data sets from different domains. We evaluate our methods and baseline methods by AUC scores. The experimental results show that our method outperforms all of the baseline methods for all of the data sets, indicating that MELL learns the overall structures of multiplex networks more effectively than the existing methods. For future work, we will extend our method for visualization and multi-label classification tasks.

ACKNOWLEDGMENTS

This work was supported by Tokyo Tech - Fuji Xerox Cooperative Research (Project Code KY260195), JSPS Grant-in-Aid for Scientific Research(B) (Grant Number 17H01785) and JST CREST (Grant Number JPMJCR1687).



[1] embedding dimension d

[2] negative sampling rate k

[3] regularization coefficient for embedding vectors λ

[4] regularization coefficient for the variance β

[5] regularization coefficient for layer vectors γ

Figure 3: Parameter sensitivity of MELL. These graphs show the AUC scores for link prediction with different parameters, $d, k, \lambda, \beta, \gamma$. The default parameters are $d = 128, k = 4.0, \lambda = 10.0, \beta = 1.0, \gamma = 1.0$, and we vary one parameter fixing others.

REFERENCES

[1] Alessio Cardillo, Jesús Gómez-Gardeñes, Massimiliano Zanin, Miguel Romance, David Papo, Francisco del Pozo, and Stefano Boccaletti. 2013. Emergence of network features from multiplexity. *Scientific Reports* 3 (Feb. 2013), 1344. https://doi.org/10.1038/srep01344

[2] Beth L. Chen, David H. Hall, and Dmitri B. Chklovskii. 2006. Wiring optimization can relate neuronal structure and function. *Proceedings of the National Academy of Sciences of the United States of America* 103, 12 (2006), 4723–4728. https://doi.org/10.1073/pnas.0506806103

[3] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2017. A Survey on Network Embedding. *ArXiv e-prints* (Nov. 2017). arXiv:1711.08752

[4] Caterina De Bacco, Eleanor A. Power, Daniel B. Larremore, and Cristopher Moore. 2017. Community detection, link prediction, and layer interdependence in multilayer networks. *Physical Review E* 95, Article 042317 (April 2017), 10 pages. Issue 4. https://doi.org/10.1103/PhysRevE.95.042317

[5] Manlio De Domenico, Andrea Lancichinetti, Alex Arenas, and Martin Rosvall. 2015. Identifying Modular Flows on Multilayer Networks Reveals Highly Overlapping Organization in Interconnected Systems. *Physical Review X* 5, Article 011027 (March 2015), 11 pages. Issue 1. https://doi.org/10.1103/PhysRevX.5.011027

[6] Manlio De Domenico, Mason A. Porter, and Alex Arenas. 2015. MuxViz: a tool for multilayer analysis and visualization of networks. *Journal of Complex Networks* 3 (2015), 159–176. Issue 2. https://doi.org/10.1093/comnet/cnu038

[7] Manlio De Domenico, Albert SolÃĺ-Ribalta, Emanuele Cozzo, Mikko KivelÃđ, Yamir Moreno, Mason A. Porter, Sergio GÃşmez, and Alex Arenas. 2013. Mathematical Formulation of Multilayer Networks. *Physical Review X* 3, Article 041022 (Oct. 2013), 15 pages. Issue 4. https://doi.org/10.1103/PhysRevX.3.041022

[8] Palash Goyal and Emilio Ferrara. 2017. Graph Embedding Techniques, Applications, and Performance: A Survey. *ArXiv e-prints* (May 2017). arXiv:1705.02801

[9] Aditya Grover and Jure Leskovec. 2016. node2Vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. 855–864. https://doi.org/10.1145/2939672.2939754

[10] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations* (May 2015).

[11] Matteo Magnani, Barbora Micenkova, and Luca Rossi. 2013. Combinatorial Analysis of Multiple Networks. *ArXiv e-prints* (March 2013). arXiv:1303.4986

[12] Tong Man, Huawei Shen, Shenghua Liu, Xiaolong Jin, and Xueqi Cheng. 2016. Predict Anchor Links Across Social Networks via an Embedding Approach. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*. 1823–1829. http://dl.acm.org/citation.cfm?id=3060832.3060876

[13] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. 701–710. https://doi.org/10.1145/2623330.2623732

[14] Luis Solá, Miguel Romance, Regino Criado, Julio Flores, Alejandro García del Amo, and Stefano Boccaletti. 2013. Eigenvector centrality of nodes in multiplex networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 23, 3, Article 033131 (Sept. 2013), 10 pages. https://doi.org/10.1063/1.4818544

[15] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. 2006. BioGRID: a general repository for interaction datasets. *Nucleic Acids Research* 34, suppl_1 (Jan. 2006), D535–D539. https://doi.org/10.1093/nar/gkj109

[16] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. 1067–1077. https://doi.org/10.1145/2736277.2741093

[17] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S. Yu. 2017. Multi-task Network Embedding. In *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA '17)*. 571–580. https://doi.org/10.1109/DSAA.2017.19

[18] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable Graph Embedding for Asymmetric Proximity. *31st AAAI Conference on Artificial Intelligence* (Feb. 2017). https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14696/14500