Lecture 3: Node Embeddings

İsmail İlkan Ceylan
Overview, motivation, and applications of graph representation learning

2 lectures: Scope, context, and applications of graph representation learning

Shallow node embedding models

3 lectures: "Simple" node embedding models and their applications in knowledge graphs

Fundamentals of graph neural networks

4 lectures: Message passing neural networks, and popular graph neural network variants

Foundations, limitations, and extensions of graph neural networks

5 lectures: Limitations of graph neural networks in expressive power and their information bottlenecks.

Generative graph representation learning

3 lectures: Graph generation via variational, or autoregressive approaches.
Node Embeddings

L1 - L2: Scope, motivation, and applications of graph representation learning

L3 - L5: Shallow node embedding models

- **L3**: Node embeddings
- **L4**: Knowledge graph embeddings
- **L5**: Knowledge graph embedding models
Overview of the Lecture

Node Embedding Models

Node embedding models as instances of encoder-decoder framework.

Word Embeddings

A historical context for node embeddings

\[
\text{birds} = \begin{bmatrix}
0.51 \\
-0.22 \\
0.83 \\
-0.42 \\
-0.53 \\
0.14 \\
0.37 \\
0.88 \\
0.11 \\
0.04
\end{bmatrix}
\]

Node Embedding Models: Factorization

Approximate pairwise node similarity using matrix factorization

\[S \approx ZZ^T\]

Node Embedding Models: Random Walks

Nodes are similar if they co-occur on short random walks
A Glimpse of Word Embeddings
How to represent the words in a sentence?

- Sentence: “Birds of a feather flock together.”

- Vocabulary $V = \{\text{birds, ..., together}\}$.

- Represent each word as a 1-hot vector?

\[
\begin{align*}
\text{birds} &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
\text{of} &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\
\text{a} &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
\text{feather} &= \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\
\text{flock} &= \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\
\text{together} &= \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
\]
Word Representations

Problems:

• Each representation is independent: A “bird” and a “songbird” have orthogonal representations.

\[
\text{birds} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{of} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{a} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}
\]

• The representation of “finch” is as close to the representation of “bird” as of a “chair”.

\[
\text{feather} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{flock} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{together} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}
\]

• The dimensionality of these vectors is $V$: Encoding depends on the size of our vocabulary!
**Idea:** Word2Vec produces a dense embedding vector for each word in the corpus such that words with similar characteristics are in close proximity to one another in the embedding space.
The word representations computed using NNs are very interesting because the learned vectors explicitly encode many linguistic regularities and patterns. Somewhat surprisingly, many of these patterns can be represented as linear translations…

\[ \text{vec(“Madrid”) - vec(“Spain”) + vec(“France”) is closer to vec(“Paris”) than to any other word vector.} \]

(Mikolov et. al, 2013)

Figure 2 (Mikolov et. al, 2013): 2-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training no supervised information about what a capital city means is given.
Node Embedding Models
An Encoder-Decoder Perspective

\[ G = (V, E) \]

Encoder and Decoder: Let \( S[u, v] \) be a similarity measure between the nodes \( u, v \) and suppose:

\[
\text{Enc}: V \rightarrow \mathbb{R}^d \quad \text{Dec}: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+ \\
\]

Shallow encoder: A lookup function \( \text{Enc}(v) = Z[v]^T \), where \( Z: \mathbb{R}^{|V| \times d} \) is a matrix of \( d \)-dimensional embeddings.

Unsupervised: We do not use node labels or features and the resulting embeddings are task-independent!
Optimization

\[ G = (V, E) \]

**Optimization:** Given a dataset \( D = \{ (u_i, v_i) \mid 1 \leq i \leq n \} \), minimize the loss:

\[
\sum_{(u, v) \in D} f(\text{Dec}(z_u, z_v), S[u, v]),
\]

where \( f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \) (e.g., mean-squared error), measures the discrepancy between \( \text{Dec}(z_u, z_v) \) and \( S[u, v] \).
Characterizing a Node Embedding Model

Idea: Node embedding models produce an embedding vector for each node such that nodes with similar properties are in close proximity to one another in the embedding space.

(1) What kind of decoder?      (2) What kind of node/graph similarity?         (3) Which loss function?

\[ G = (V, E) \]
Node Embeddings: Context
Context of a Word

Word2Vec:

• **Intuition**: A word’s meaning is given by its context, i.e., words that it co-occurs within a fixed-sized window.

• Deriving **context** for the word bird(s) from multiple sentences:

  “Birds of a feather flock together.”  “He heard the birds singing.”  “The early bird gets the worm.”
### Training Data: The Early Bird Gets the Worm

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Training data</th>
</tr>
</thead>
<tbody>
<tr>
<td>“The early bird gets the worm.”</td>
<td>(the, early)</td>
</tr>
<tr>
<td>“The early bird gets the worm.”</td>
<td>(early, the), (early, bird)</td>
</tr>
<tr>
<td>“The early bird gets the worm.”</td>
<td>(bird, early), (bird, gets)</td>
</tr>
<tr>
<td>“The early bird gets the worm.”</td>
<td>(gets, bird), (gets, the)</td>
</tr>
<tr>
<td>“The early bird gets the worm.”</td>
<td>(the, gets), (the, worm)</td>
</tr>
<tr>
<td>“The early bird gets the worm.”</td>
<td>(worm, the)</td>
</tr>
</tbody>
</table>

For every target word in a sentence, training data $D$ consists of all combinations (target word, context word) within a fixed window. The example shown above uses window size three. Trained to predict the probability of a context word being present when an input word is present: $P(\text{early} \mid \text{bird})$. 
Word Embeddings

Continuous skip-gram model (Mikolov et. al, 2013):

- **Input**: Large corpus of text.

- **Training data**: Go through each position in text, yielding a target word $w$, and pick each context word $c$, obtaining a set of word-context pairs $(w, c)$.

- **Encoding**: Every word $w$ is represented as a vector $\mathbf{w}$.

- **Predicting the context**: Use the similarity between the embeddings $\mathbf{w}$ and $\mathbf{c}$ to determine $p(c \mid w)$.

- **Optimization**: Keep updating the word embeddings to maximise this probability.

\[
\begin{align*}
\text{birds} &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, & \text{of} &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, & \text{a} &= \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \\
\text{feather} &= \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, & \text{flock} &= \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, & \text{together} &= \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
\end{align*}
\]
What is the Context of a Node?

$G = (V, E)$

**Sequence vs graph:** Graphs are not linearly ordered structures, so there is no node before/after.

**Context:** There are nodes “around” a node, so a node’s context is given by its neighbourhood.

**Intuition:** Two nodes are similar if their neighbourhoods are similar according to some notion of neighbourhood.

**Optimization:** Reconstruct the local neighbourhood of a node, using gradient-based optimization.
Node Embeddings: Factorization
**Matrix Factorization**

**Similarity:** View $S$ is an approximation and generalization of $A$.

**Goal:** Learn $S$ using matrix factorization techniques, i.e., decode to recover $S$. 
Matrix Factorization: Inner Product

**Decoder**: Similarity between two nodes is proportional to the dot product of their embeddings:

$$\text{Dec}(z_u, z_v) = z_u^T z_v$$

**Loss**:

$$\mathcal{L} = \sum_{(u,v)\in D} \|\text{Dec}(z_u, z_v) - S(u, v)\|_2^2$$

Mapping back to the matrix $Z$ of node embeddings, reveals the connection to matrix factorization:

$$\mathcal{L} \approx \|ZZ^T - S\|_2^2.$$
Matrix Factorization: Inner Product

In its simplest form, we can set $S = A$ and minimise

$$\mathcal{L} = \sum_{(u,v) \in D} \| \text{Dec}(z_u, z_v) - A(u, v) \|_2^2$$

This objective approximately recovers the graph:

$$\mathcal{L} \approx \| ZZ^T - A \|_2^2.$$ 

To capture multi-hops, we can set a similarity defined over:

$$A[u, v], \ldots, A^k[u, v].$$

Decoder (i.e., any pairwise similarity) and accordingly the target similarity (neighbourhood overlap measures) can vary...
Matrix Factorization: Other Approaches

Similarity can be defined in terms of generalizations of other matrices, i.e., the graph Laplacian.

We can decode differently, i.e., based on the $L_2$-distance: $\text{Dec}(z_u, z_v) = \|z_u - z_v\|_2^2$.

Finally, we can define objectives ensuring that similar nodes are close to each other in the space.
Node Embeddings: Random Walks
Matrix Factorization: Random Walks

**Similarity:** Two nodes have similar embeddings if they tend to co-occur on short random walks over the graph.

**Decode:** The probability of visiting the node $v$ on a random walk starting from $u$, i.e., $p(v \mid u)$, parametrised via softmax:

$$p(v \mid u; \theta) = \text{Dec}(z_u, z_v) = \frac{e^{z_u^\top z_v}}{\sum_{k \in V} e^{z_u^\top z_k}}$$

**Loss:** Let $D = \{(u_i, v_i) \mid 1 \leq i \leq n\}$ be the multiset sampled from some random walk distribution $p_{rw}(v_i \mid u_i)$, minimize:

$$\mathcal{L} = \sum_{(u,v) \in D} - \log(\text{Dec}(z_u, z_v))$$
A rough picture of random walk algorithms:

- Sample a starting node $u \in V$ according to some distribution over vertices (e.g., uniform, or any stationary distribution).
- Sample random walks $u, v_1, \ldots, v_L$ starting from $u \in V$.
- Using a window size $T$, generate the multiset of pairs:
  \[
  D = \{ ((u, v_1), (u, v_2), (v_1, v_2), \ldots) \}
  \]
- These are target node and context pairs - we can apply the skip gram algorithm!
**From Word to Random Walk Embeddings**

**Word embeddings:** Let \( c, d \in \mathbb{R}^d \) be embeddings of the context words \( c, d \), and \( w \in \mathbb{R}^d \) the embedding of the word \( w \). Predict the context \( c \) of a word \( w \):

\[
p(c \mid w; \theta) = \frac{e^{c^\top w}}{\sum_{d \in C} e^d}
\]

**Random walk embeddings:** The probability of visiting the node \( v \) on a random walk starting from the node \( u \):

\[
p(v \mid u; \theta) = \text{Dec}(z_u, z_v) = \frac{e^{z_u^\top z_v}}{\sum_{k \in V} e^{z_u^\top z_k}}
\]
Matrix Factorization: Random Walks

**Problem**: Optimization can be inefficient, since the normalization factor in softmax can be expensive to compute:

\[
\mathcal{L} = \sum_{(u,v) \in D} - \log(\text{Dec}(\mathbf{z}_u, \mathbf{z}_v)) = \sum_{(u,v) \in D} - \log\left(\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in V} e^{\mathbf{z}_u^\top \mathbf{z}_k}}\right)
\]

This has been noted in **Word2Vec** (Mikolov et al., 2013)!

- **DeepWalk** uses hierarchical softmax (Perozzi et al., 2014).
- **Node2Vec** uses the negative sampling (Grover and Leskovec, 2016).

These algorithms deviate from the original objective and the same is true for the skip gram algorithm (Goldberg and Levy, 2014).
A Negative Sampling Approach

The probability \( p((u, v) \in D) \) that \((u, v)\) comes from \( D \).

\[
p((u, v) \in D; \theta) = \frac{1}{1 + e^{-z_u^\top z_v}}
\]

The probability \( p((u, v) \notin D) \) that \((u, v)\) does not come from \( D \).

\[
p((u, v) \notin D; \theta) = \frac{1}{1 + e^{z_u^\top z_v}}
\]

Node2Vec minimizes the loss:

\[
\mathcal{L} = \sum_{(u, v) \in D} -\log \sigma(z_u^\top z_v) - \gamma \mathbb{E}_{k \in p(V)} \left[ \log \sigma(-z_u^\top z_k) \right]
\]

...where \( \gamma > 0 \) is an hyper-parameter, and \( p(V) \) is a distribution over nodes.
DeepWalk and Node2Vec

Key differences between DeepWalk and Node2Vec

- DeepWalk can also be used with negative sampling.
- DeepWalk employs uniformly random walks to define $p(v \mid u)$.
- Node2Vec allows for biased $p(v \mid u)$ through hyper parameters and focuses on 2nd-order random walks.
Random Walks Embeddings Factorize Matrices

<table>
<thead>
<tr>
<th>Model</th>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINE</td>
<td>( \log \left( \text{vol}(G)D^{-1}AD^{-1} \right) - \log b )</td>
</tr>
<tr>
<td>DeepWalk (skip gram-negative sampling)</td>
<td>( \log \left( \text{vol}(G) \left( \frac{1}{T} \sum_{1 \leq t \leq T} (D^{-1}A)^t D^{-1} \right) \right) - \log b )</td>
</tr>
<tr>
<td>Node2Vec</td>
<td>( \log \left( \text{vol}(G) \left( \frac{1}{2T} \sum_{1 \leq t \leq T} \left( \frac{\sum_u X_{w,u} P_{c,w,u}^t + \sum_u X_{c,u} P_{w,c,u}^t}{\sum_u X_{w,u} \sum_u X_{c,u}} \right) \right) \right) - \log b )</td>
</tr>
</tbody>
</table>

Random walk embeddings implicitly approximate the shown matrices under certain assumptions, e.g., conditions for ensuring stationary distribution are imposed (Qui et al., 2018).
A Closer Look at DeepWalk

\[
S_{DW} = \log \left( \text{vol}(G) \left( \frac{1}{T} \sum_{1 \leq t \leq T} A_t \right) - \log b \right) - \log b
\]

\[
\sum_i \sum_j A_{ij} = P
\]

transition probabilities of a random walker over a graph

DeepWalk factorizes the matrix \( S_{DW} \)

LINE is a special case of DeepWalk with \( T = 1 \).
A Closer Look at Node2vec

\[ \sum_i \sum_j A_{ij} = P^t \]

Tensor of the transition probabilities of 2nd order random walks:

\[ p(w_{j+1} = c \mid w_j = w, w_{j-1} = u) \]

\[ S_{n2v} = \log \left( \text{vol}(G) \left( \frac{1}{2T} \sum_{1 \leq t \leq T} \left( \sum_u X_{w,u} P^t_{c,w,u} + \sum_u X_{c,u} P^t_{w,c,u} \right) \right) \right) - \log b \]

Context window size

#negative samples

Stationary distribution of 2nd order random walks.

Node2Vec factorizes the matrix \( S_{n2v} \)
Use and Limitations of Node Embeddings
Node Embeddings for Downstream Tasks

\[ G = (V, E) \]

Unsupervised: We do not use node labels or features and the resulting embeddings are task-independent!

- **Node-level**: Feed the learned node embeddings to a standard classifier/regressor/clusterer.
- **Link-level**: Derive edge features from node representations, e.g., \( z_{u,v}(i) = \left( z_u(i) + z_v(i) \right)/2 \).
- **Graph-level**: Aggregate node features to yield a graph representation, e.g., \( z_G = \sum_{u \in V} z_u \).
Unsupervised: We do not use node labels or features and the resulting embeddings are task-independent!

- Node embeddings are (still) widely used in the industry due to their simplicity and scalability.
- We focused on node embeddings over undirected graphs, but the ideas can generalise to other graphs.
- Node embeddings can be adapted for multi-relational graphs; see, e.g., Metapath2vec.
Limitations of Shallow Embeddings

The embedding of the nodes do not share any parameters, i.e., hard to model dependencies.

It is hard to capture certain structural similarities, e.g., $u_1$ and $u_{10}$.

Node/graph-level features cannot be utilised effectively.

Hard to capture graph-level, global properties, hence worse on graph-level tasks.

Transductive: No embeddings for new nodes, unseen during training.
Summary and Outlook

Overview, motivation, and applications of graph representation learning

- 2 lectures: Scope, context, and applications of graph representation learning.

Shallow node embedding models

- 3 lectures: “Simple” node embedding models and their applications in knowledge graphs

Fundamentals of graph neural networks

- 4 lectures: Message passing neural networks, and popular graph neural network variants

Foundations, limitations, and extensions of graph neural networks

- 5 lectures: Limitations of graph neural networks in expressive power and their information bottlenecks.

Generative graph representation learning

- 3 lectures: Graph generation via variational, or autoregressive approaches.
Lecture 3: Node Embeddings

- Node embeddings inspired from word embeddings.
- Context: A node’s local neighbourhood.
- Embeddings via matrix factorization or random walks
- Use of node embeddings and their limitations.


