Machine Learning with Graphs: Representation learning 1/3 - Introduction, Random Walk based Embedding

Arlei Silva

Spring 2022

Representation learning on graphs

Representation learning consists of learning features (or embeddings) from data. This is an alternative to *feature engineering*, where feature generation relies on domain knowledge. Learned representations are represented as d-dimensional vectors. Representation learning can be either supervised or unsupervised, depending on whether labels are applied in the process. These representations can then be applied in so-called *downstream tasks* (e.g. classification, clustering, etc.). As examples, representations for words and images can be generated using models such as *Word2Vec* and *Convolutional Neural Networks (CNNs)*.

For the case of graph data, representation learning can be applied to learn node, edge, and (entire) graph representations. The most popular approach for supervised representation learning on graphs is *Graph Neural Networks (GNNs)*, which will be the focus of the next lecture. Here, we will focus on unsupervised embedding methods based on random walks.

Skipgram

The embedding methods we will discuss here were inspired by the *Skipgram model* for word embedding. The goal of word embedding is to learn *dense (or distributed) representations* for words such that semantic relationships between the words can be captured by distances/similarities between vectors. More specifically, skipgram is based on the following principle proposed by the linguist John Rupert Firth:

"You should know a word by the company it keeps"

Here, the "company" of a word is its *context*. The context of a word w is the set of words within a fixed distance/radius from a given occurrence w. We call the set of word occurrences and their contexts a *corpus*. We will associate

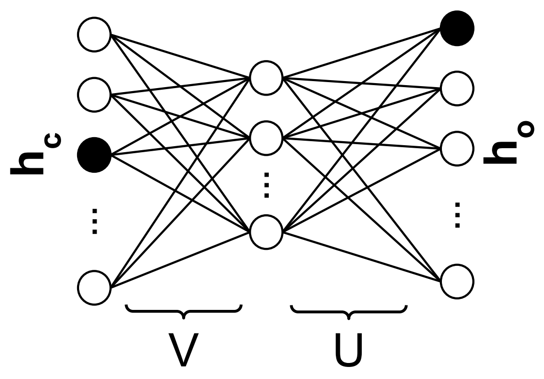

Figure 1: Skipgram as a neural network.

a corpus and word representations using the following *likelihood*:

$$J(\Theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m} p(w_{t+j} | w_t, \Theta)$$

where Θ—the set of parameters—contains representations for every word in the vocabulary.

We will learn the word representations via maximum likelihood estimation. Maximizing the above equation is the same as minimizing the following:

$$J'(\Theta) = -\sum_{t=1}^{T} \sum_{-m \le j \le m} p(w_{t+j} | w_t, \Theta)$$

Skipgram applies the *softmax* function as $p(w_{t+1}|w_t, \Theta)$:

$$p(w_o|w_c, \Theta) = \frac{\exp(\mathbf{u}_o^T \mathbf{v}_c)}{\sum_{w \in W} \exp(\mathbf{u}_w^T \mathbf{v}_c)} \tag{1}$$

where $\mathbf{u}_o$ is the output vector representation for word w_o, $\mathbf{v}_c$ is the input vector representation for word w_c and W is the vocabulary. Thus $\Theta = \mathbf{u}_1, \ldots \mathbf{u}_{|W|}$, $\mathbf{v}_1, \ldots \mathbf{v}_{|W|}$.

We can implement the skipgram model as a neural network, as shown in Figure 1. Let $\mathbf{h}_c$ and $\mathbf{h}_o$ be *one-hot* encodings for w_c and w_o, respectively. Moreover, let $U, V \in \mathbb{R}^{|W| \times D}$ be embedding matrices, where each row contains the D-dimensional embedding for a word in the vocabulary. Then, we can compute $\mathbf{v}_c = V^T \mathbf{h}_c$ and can predict $\sigma(U(V^T \mathbf{v}_c)) \approx \mathbf{u}_o$, where σ is the softmax activation function. Matrices U and V, which are weights in the network, can be learned by minimizing the *cross-entropy loss* for pairs (w_c, w_o) in the corpus using gradient descent.

One issue with the skipgram model from Figure 1 is the need for normalization in the softmax activation (see Equation 1—the denominator depends on the entire vocabulary. We will describe a more efficient alternative using *negative sampling*.

For an arbitrary pair of words w_c, w_o let's define a label z such that z is 1 if the pair occurs within a window in the corpus or 0, otherwise. Let $\mathcal{D}$ and $\mathcal{D}'$ be the multiset of pairs that occur and do not occur in the corpus, respectively. We will define label probabilities using a *sigmoid function*, as follows:

$$p(z = 1|w_c, w_o) = \sigma(\mathbf{u}_o^T \mathbf{v}_c) = \frac{1}{1 + \exp(-\mathbf{u}_o^T \mathbf{v}_c)}$$

$$p(z = 0|w_c, w_o) = \sigma(-\mathbf{u}_o^T \mathbf{v}_c) = \frac{1}{1 + \exp(\mathbf{u}_o^T \mathbf{v}_c)}$$

Putting them together:

$$p(z|w_c, w_o) = \left(\frac{1}{1 + \exp(-\mathbf{u}_o^T \mathbf{v}_c)}\right)^z \left(\frac{1}{1 + \exp(\mathbf{u}_o^T \mathbf{v}_c)}\right)^{1-z}$$

The likelihood function for this model can be computed as:

$$J(\Theta) = \prod_{w_c, w_o \in \mathcal{D} \cup \mathcal{D}'} \left(\frac{1}{1 + \exp(-\mathbf{u}_o^T \mathbf{v}_c)}\right)^z \left(\frac{1}{1 + \exp(\mathbf{u}_o^T \mathbf{v}_c)}\right)^{1-z}$$

And the negative log-likelihood, to be minimized, is:

$$J'(\Theta) = -\sum_{w_c, w_o \in \mathcal{D}} \log\left(\frac{1}{1 + \exp(-\mathbf{u}_o^T \mathbf{v}_c)}\right) - \sum_{w_c, w_o \in \mathcal{D}'} \log\left(\frac{1}{1 + \exp(\mathbf{u}_o^T \mathbf{v}_c)}\right)$$

The negative log-likelihood for each pair of words in $\mathcal{D} \cup \mathcal{D}'$ is as follows:

$$J_t'(\Theta) = -z \log \sigma(\mathbf{u}_o^T \mathbf{v}_c) - (1 - z) \log \sigma(-\mathbf{u}_o^T \mathbf{v}_c)$$

However, notice that while sampling pairs in $\mathcal{D}$ is easy, the set $\mathcal{D}'$ is much larger. Word2vec proposes the use of *negative sampling* to efficiently sample pairs that are likely to be in $\mathcal{D}'$. More specifically, k random words are used as negative samples for each unique pair:

$$J'_{w_c, w_o}(\Theta) = -\#(w_c, w_o)(\log \sigma(\mathbf{u}_o^T \mathbf{v}_c) - k\mathbb{E}_{w \sim P(w)} \log \sigma(\mathbf{u}_w^T \mathbf{v}_c)) \tag{2}$$

where $\#(w_c, w_o)$ is the number of times the pair occurs in the corpus $P(w)$ is the word probability in the corpus.

Computing representations that minimize the negative log-likelihood for the entire corpus is also straightforward. Word2vec applies *stochastic gradient descent*[6] to learn representations in an online fashion (one pair of words at a time). For more details on word2vec see [4].

Random-walk based graph embedding

Deepwalk, node2vec, and other similar graph embedding methods are inspired by word2vec [3]. The idea is to apply a random walk process to extract patterns similar to sentences in the text case. A random walk on a graph is a *Markov Chain* $X_1, X_2, \ldots X_T$ that starts at a given vertex and transitions through the edges with probability given by:

$$p(X_t = v | X_{t-1} = u) = \frac{1}{|N(u)|} \tag{3}$$

where we have assumed that the graph is unweighted—this can be easily extended to weighted graphs.

Random-walk based graph embedding methods sample random walks starting from each vertex in the graph. By treating each walk as a sentence, we can apply the same algorithm described in the last section. Other variants skipgram models—e.g. using hierarchical softmax—can also be applied.

Embedding via matrix factorization

Both the skipgram model and RW-based graph embedding can be formulated as matrix factorization. In particular, skipgram with negative sampling is related to the *Pointwise Mutual Information Similarity (PMI)* between words w_o, w_c in the vocabulary:

$$PMI(w_o, w_c) = \log \frac{p(w_o, w_c)}{p(w_o)p(w_c)} = \log \frac{\#(w_o, w_c)}{\#(w_o)\#(w_c)}$$

where $\#(w_o, w_c)$ and $\#(w_o)$ are counts of pairs and words, respectively.

By setting the derivative of Equation 2 with respect to $\mathbf{u}_o^T \mathbf{v}_c$ to zero, we get:

$$\mathbf{u}_o^T \mathbf{v}_c = PMI(w_o, w_c) - \log k$$

For the case of graph embedding, we can sample the initial vertex according to the stationary distribution $p(v_c) = \pi \in [0, 1]$ of the random walk, where n is the number of vertices in the graph and $\sum_{v \in V} \pi_v = 1$. The value of π_v is the probability that the walker is at vertex v for an infinitely long walk. Then $p(v_c, v_o) = p(v_c))p(v_o|v_c)$. Let us define the transition matrix $M \in [0, 1]^{n \times n}$, where $M_{uv} = p(v|u)$. Then, we can write the PMI matrix of the random walk as:

$$PMI = \log \Pi M^\tau - \log \pi^T \pi$$

where $\Pi = diag(\pi)$ and τ is the length of the walk.

We can compute embedding matrices U and V using the Singular Value Decomposion $PMI = UV^T = U' \Lambda V'^T$ by setting $U = U' \sqrt{\Lambda}$ and $V^T = \sqrt{\Lambda} V'^T$.

References

[1] Christopher M Bishop. *Pattern recognition and machine learning.* Springer, 2006.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning.* Springer, 2009.

[3] Zexi Huang, Arlei Silva, and Ambuj Singh. A broader picture of random-walk based graph embedding. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 685–695, 2021.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[5] Kevin Murphy. An introduction to graphical models. *Rap. tech*, 96:1–19, 2001.

[6] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.