Graph Embedding for Deep Learning

Graph Learning and Geometric Deep Learning — Part 1

Flawnson Tong · Follow
Published in Towards Data Science
10 min read · May 6, 2019

> *Be sure to read an **overview of Geometric Deep Learning** and **the prerequisites** to become familiar with this niche in machine learning.*
>
> *Follow **my Twitter** and join the **Geometric Deep Learning subreddit** for latest updates in the space.*

There are alot of ways machine learning can be applied to graphs. One of the easiest is to turn graphs into a more digestible format for ML.

Graph embedding is an approach that is used to transform nodes, edges, and their features into vector space (a lower dimension) **whilst maximally preserving properties like graph structure and information.** Graphs are tricky because they can vary in terms of their scale, specificity, and subject.

A molecule can be represented as a small, sparse, and static graph, whereas a social network could be represented by a large, dense, and dynamic graph. Ultimately this makes it difficult to find a silver bullet embedding method. The approachess that will be covered each vary in performance on different datasets, but they are the most widely used in Deep Learning.

Graphs are the main focus of this series, but if the 3D imaging applications are more your thing, then I recommend this fantastic article by the Gradient.

Embedding Graph Networks

If we view embedding as a transformation to a lower dimension, embedding methods themselves are not a type of neural network model. Instead,they are a type of algorithm used in **graph pre-processing with the goal to turn a graph into a computationally digestible format.** This is because graph type data, by nature, are discrete.

> *Machine learning algorithms are tuned for continuous data, hence why embedding is always to a continuous vector space.*

As recent work has shown, there is a variety of ways to go about embedding graphs, each with a different level of granularity. Embeddings can be performed on the

node level, the sub-graph level, or through strategies like graph walks. These are some of the most popular methods.

DeepWalk — Perozzi et al

Deepwalk isn't the first of it's kind, but it is one of the first approaches that have been widely used as a benchmark in comparison with other graph learning approaches. Deepwalk belongs to the family of graph embedding techniques that uses walks, which are a concept in graph theory that enables the **traversal of a graph by moving from one node to another, as long as they are connected to a common edge.**

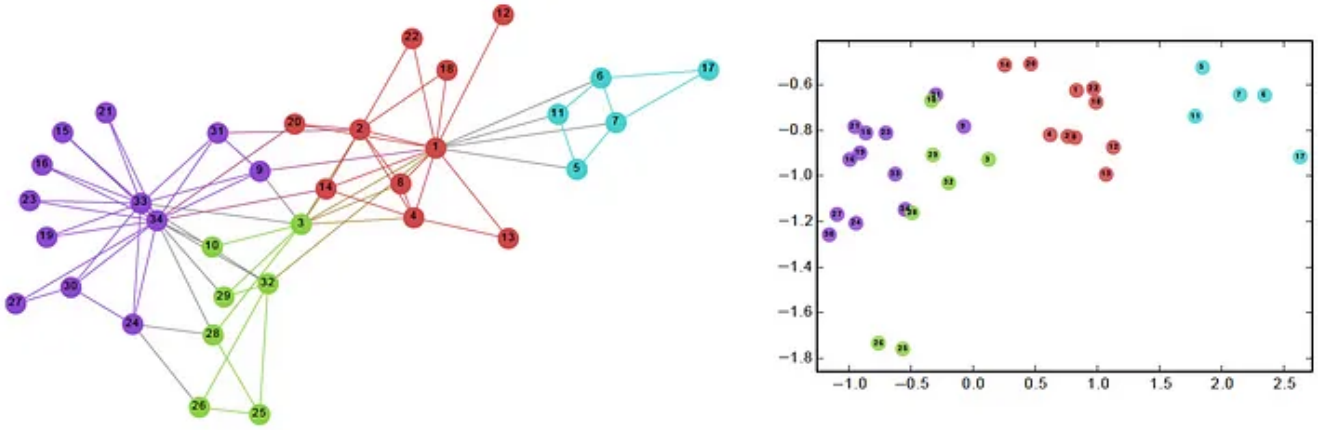
If you represent each node in a graph with an arbitrary representation vector, you can traverse the graph. **The steps of that traversal could be aggregated by arranging the node representation vectors next to each other in a matrix.** You could then feed that matrix representing the graph to a recurrent neural net. Basically, you can use the truncated steps of graph traversals as input for an RNN. This is analogous to the way word vectors in a sentence are put together.

The approach taken by DeepWalk is to complete a series of random walks using the equation:

$$\Pr\left(v_i \mid \left(\Phi(v_1), \Phi(v_2), \cdots, \Phi(v_{i-1})\right)\right)$$

The goal is to estimate the likelihood of observing node ***vi*** given all the previous nodes visited so far in the random walk, where ***Pr()*** is probability, Φ is a mapping function that represents the latent representation associated with each node **v** in the graph.

The latent representations is what becomes the input for a neural network. The neural network, based on what nodes and how often the nodes were encountered during the walk, can make a prediction about a node feature or classification.



The original graph and it's embedding (Courtesy of the DeepWalk research team)

The method used to make predictions is **skip-gram**, just like in Word2vec architecture for text. Instead of running along the text corpus, DeepWalk runs along the graph to learn an embedding. The model can take a target node to predict it's "context", which in the case of a graph, means it's connectivity, structural role, and node features.

Although DeepWalk is relatively efficient with a score of **O(|V|)**, this approach is **transductive**, meaning whenever a new node is added, the model must be retrained to embed and learn from the new node.

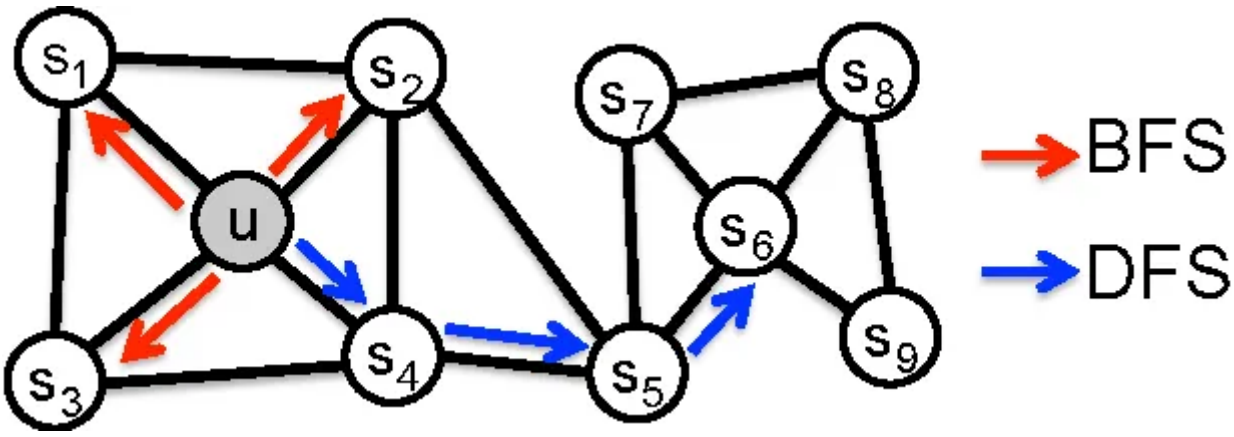
Node2vec — Grover et al

You've heard of Word2vec now prepare for... Node2vec (Aditya Grover et al)

One of the more popular graph learning methods, Node2vec is one of the first Deep Learning attempts to learn from graph structured data. The intuition is similar to that of DeepWalk:

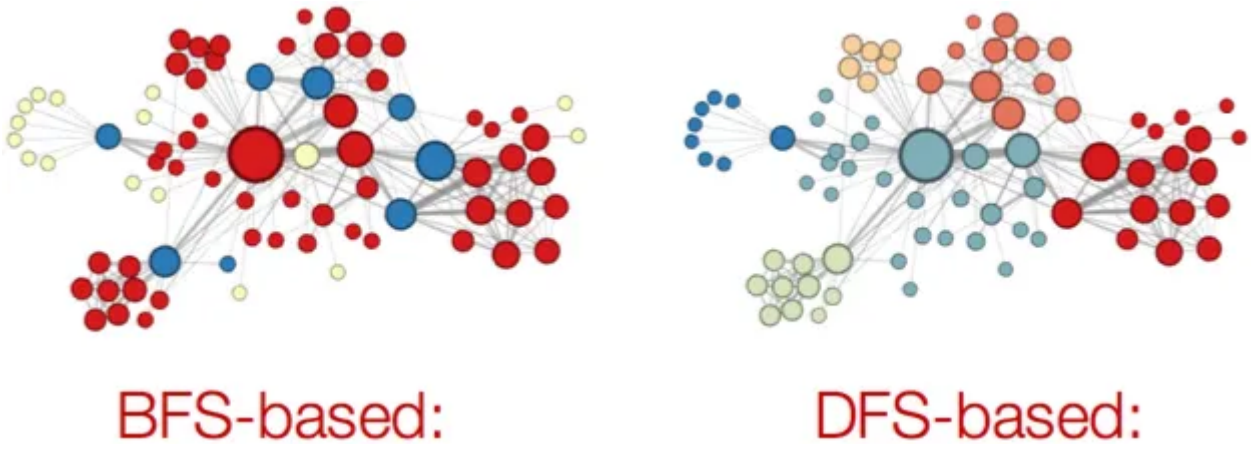
> *If you turn each node in a graph into an embedding as you would words in sentence, a neural network can learn representations for each node.*

The difference between Node2vec and DeepWalk is subtle but significant. Node2vec features a walk bias variable α, which is parameterized by p and q. The parameter p prioritizes a breadth-first-search (BFS) procedure, while the parameter q prioritizes a depth-first-search (DFS) procedure. The decision of where to walk next is therefore influenced by probabilities $1/p$ or $1/q$.



Both BFS and DFS are common algorithms in CS and graph theory (Courtesy of Semantic Scholar)

As the visualization implies, **BFS is ideal for learning local neighbors, while DFS is better for learning global variables.** Node2vec can switch to and from the two priorities depending on the task. This means that given a single graph, Node2vec can return different results depending on the values of the parameters. As per DeepWalk, Node2vec also takes the latent embedding of the walks and uses them as input to a neural network to classify nodes.



BFS vs DFS (Courtesy of SNAP Stanford)

Experiments demonstrated that **BFS is better at classifying according to structural roles (hubs, bridges, outliers, etc.) while DFS returns a more community driven classification scheme.**

Node2vec is one of the many graph learning project that have come out of Stanford's SNAP research group dedicated to graph analytics. Many of their works have been the origin of many great strides in geometric Deep Learning.

Graph2vec — Narayanan et al

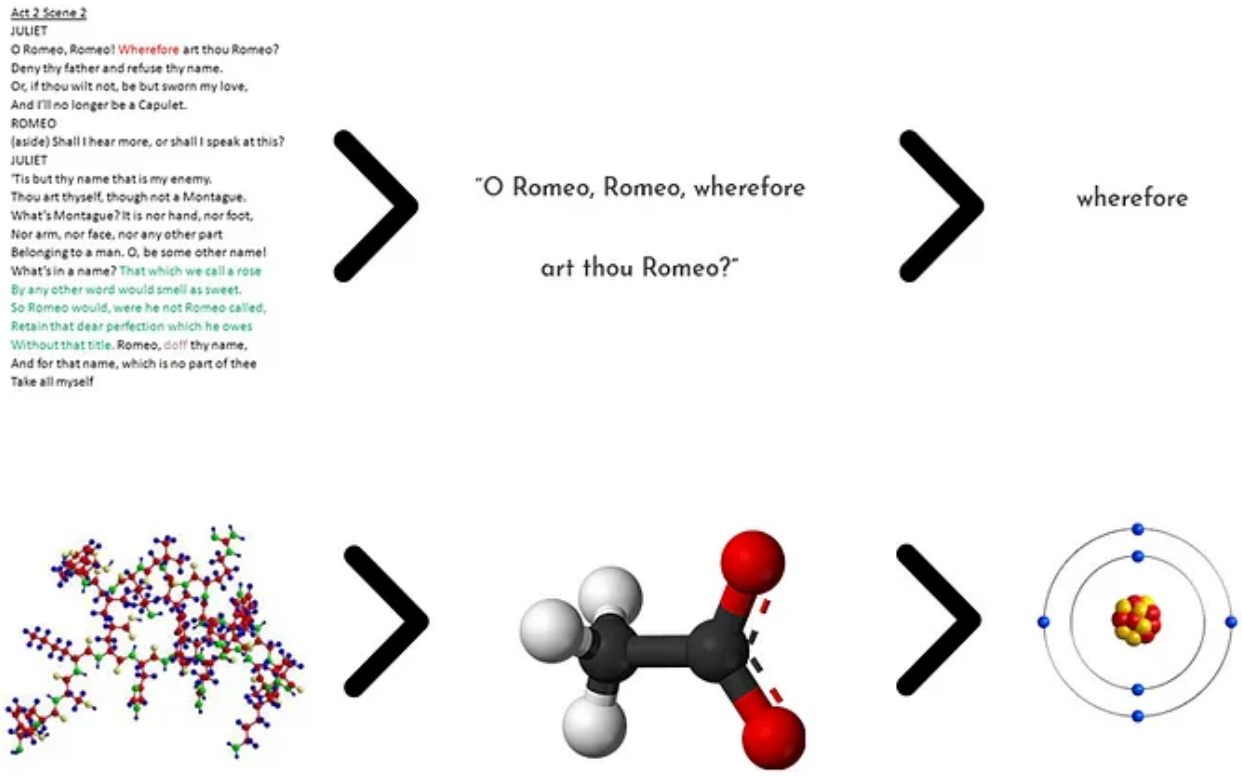
A modification to the node2vec variant, graph2vec essentially learns to embed a graph's sub-graphs. This is demonstrated by an equation that is used in doc2vec, a

closely related variant, and a point of inspiration for this paper.

$$Pr(w_j|d) = \frac{exp(\vec{d} \cdot \vec{w_j})}{\sum_{w \in V} exp(\vec{d} \cdot \vec{w})}$$

In plain english, this equation can be written as: the probability of the word (**wj**) appearing in context given document (**d**) equals the exponential of the document embedding matrix (***d~***) multiplied by the word embedding matrix (***w~j*** is sampled from the document), divided by the sum of all the exponentials of the document embedding matrix multiplied by the word embedding matrix for each word in the vocab list (**V**) across all documents.

Using an analogy with word2vec, **if a document is made of sentences (which is then made of words), then a graph is made of sub-graphs (which is then made of nodes).**



Everything is made of smaller things

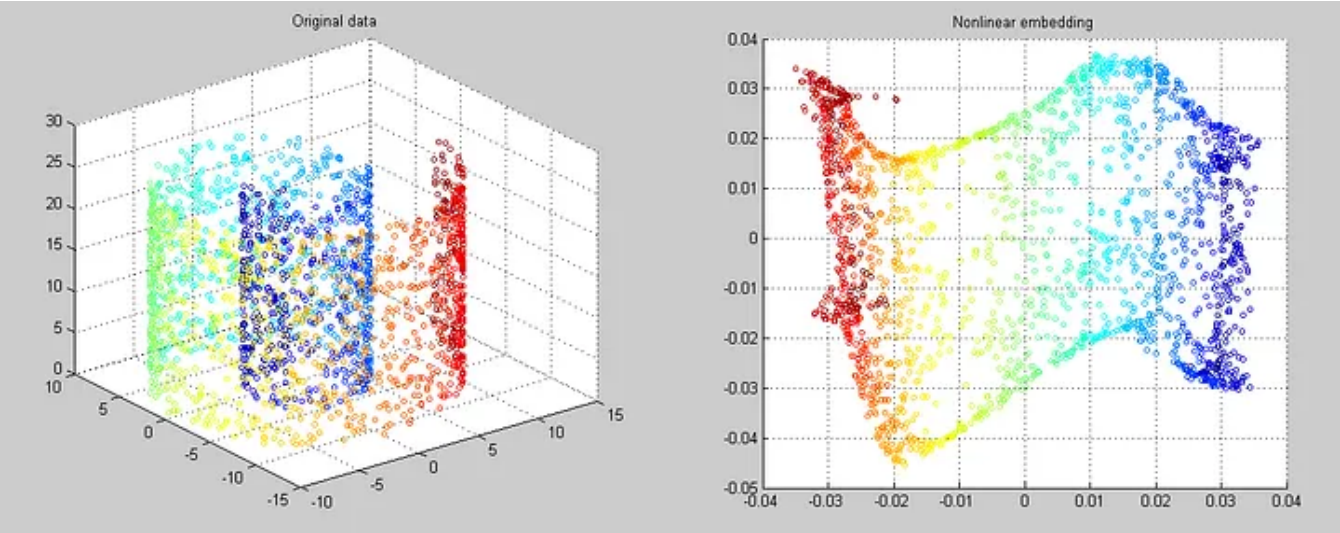
These predetermined sub-graphs have a set number of edges, as specified by the user. Once again, it is the latent sub-graph embeddings that are passed into a neural network for classification.

Structural Deep Network embedding (SDNE) — Wang et al

Unlike the previous embedding techniques, SDNE does not use random walks. Instead, it tries to learn from two distinct metrics:

- **First-order proximity:** two nodes are considered similar if they share an edge (pairwise similarity)
- **Second-order proximity:** two nodes are considered similar if they share many neighboring/adjacent nodes

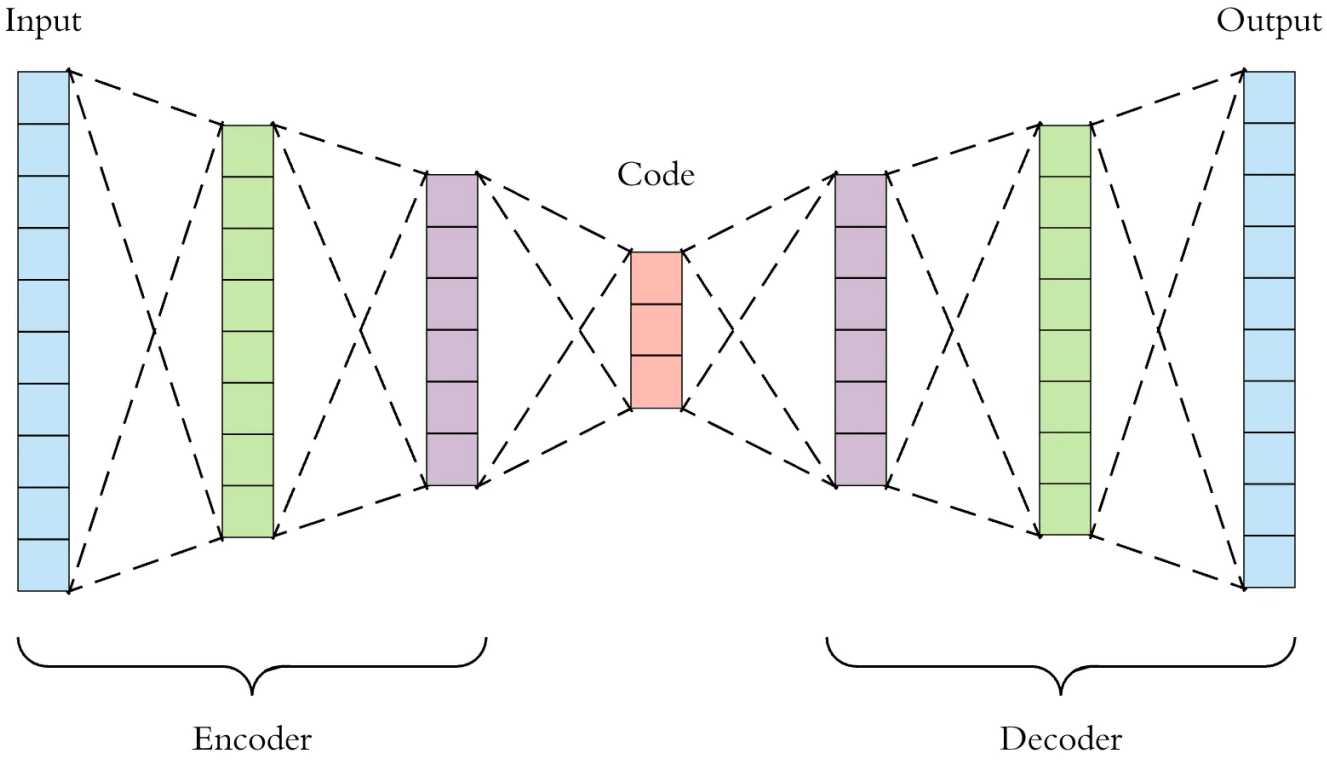
The ultimate goal is to capture highly non-linear structures. This is acheived by using **deep autoencoders (semi-supervised) to preserve the first order (supervised) and second order (unsupervised) network proximities.**



Dimensionality reduction with LE (Courtesy of MathWorld)

To preserve first order proximity, the model also a variation of **Laplacian Eigenmaps**, a graph embedding/dimensionality reduction technique. The Laplacian Eigenmap embedding algorithm **applies a penalty when similar nodes are mapped far from each other in the embedded space**, thus allowing for optimization by

loss function it must minimize.



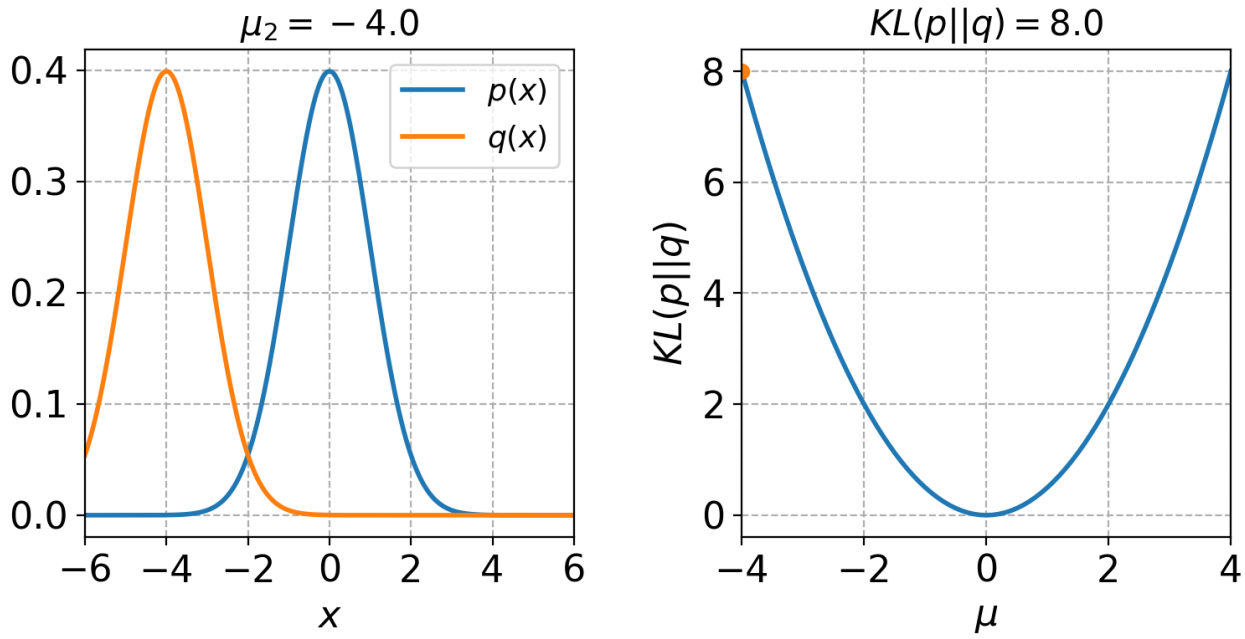
An autoencoder (Courtesy of Arden Dertat)

Together, the first order proximity loss function and the second order reconstruction loss function are jointly minimized to return a graph embedding. The embedding is than learned from by a neural network.

Large-scale Information Network Embedding (LINE) — Tang et al

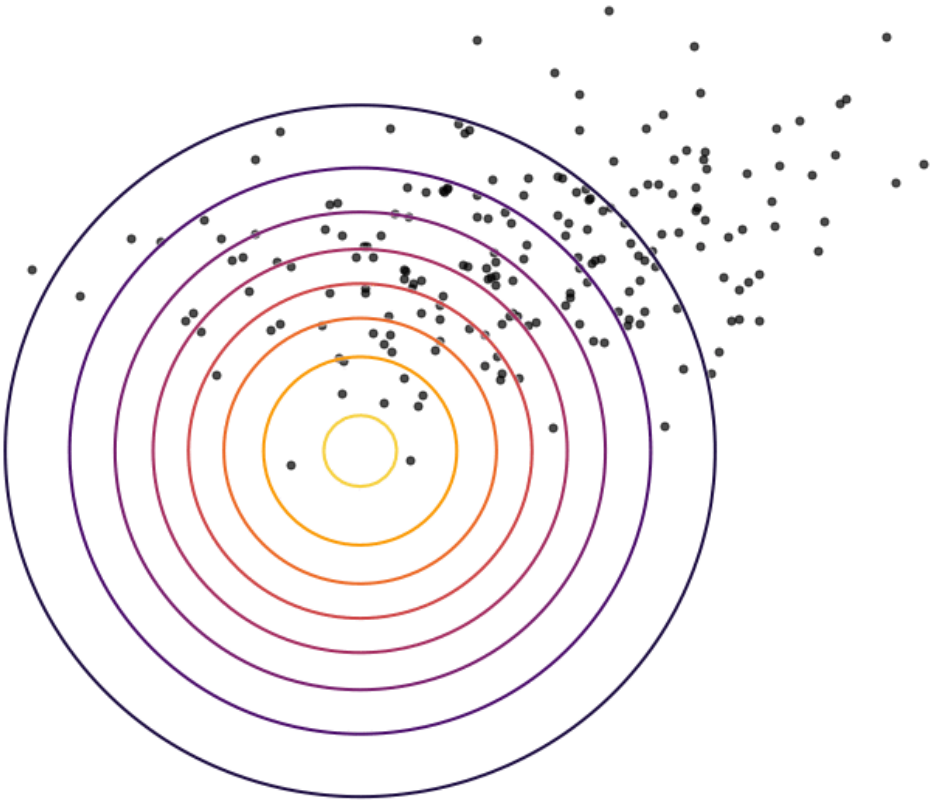
LINE (Jian Tang et al) explicitly defines two functions; one for **first order proximity** and another for **second order proximity.** In the experiments conducted by the original research, second order proximity performed significantly better than first, and it was implied that including higher orders may level off the improvements in accuracy.

The goal of LINE is to minimize the difference between the input and embedding distributions. This is achieved using KL divergence:



A simple case of KL-divergence minimization

The visualizations are simple, the math less so. Aurélien Géron has a great video on the subject. On another note, Géron is also one of the few researchers in graph learning, and has combined knowledge graphs and Deep Learning to improve video recommendations while working at YouTube.



LINE defines two joint probability distributions for each pair of nodes then minimizes the KL divergence of the distributions. The two distributions are the adjacency matrix and the dot product of node embedding. KL Divergence is an important similarity metric in information theory and entropy. The algorithm is used in probabilistic generative models like Variational Autoencoders, which embed inputs of an autoencoder into a latent space, which becomes the distribution.

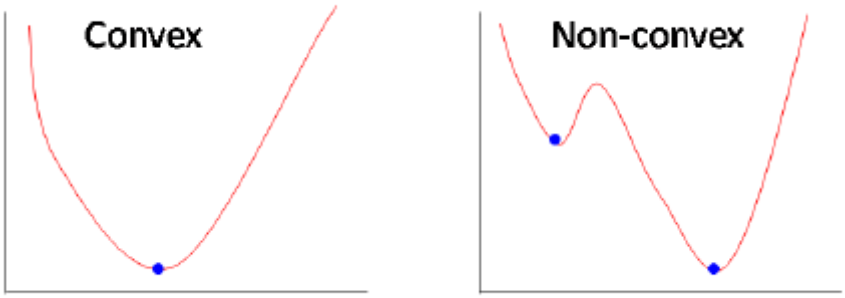
Since the algorithm has to define new functions for each increasing order of proximity, **LINE doesn't perform very well if the application needs an understanding of node community structure.**

Nevertheless, the simplicity and effectiveness of LINE are just a couple reasons why it was the most cited paper on WWW of 2015. This work helped inspire interest in

Graph Learning as a niche in Machine Learning and eventually Deep Learning in specific.

Hierarchical Representation Learning for Networks — Chen et al

HARP is an improvement to the previously mentioned embedding/walking based models. Previous models risked getting stuck in local optima since their objective functions are non-convex. Basically this means, the ball can't roll to the absolute bottom of the hill.



Gradient decent isn't perfect (Courtesy of Fatih Akturk)

Therefore, the intended **motive**:

> *Improve the solution and avoid local optima by better weight inizialization.*

and the proposed **method**:

> *Use graph coarsening to aggregate related nodes into "supernodes"*

was made.

HARP is essentially a **graph-preprocessing step that simplifies the graph** to make for faster training.

> ***After coarsening the graph, it then generates an embedding of the coarsest "supernode", followed by an embedding of the entire graph (which itself is made of supernodes).***

This strategy is followed for each "supernode" in the entire graph.

Since HARP can be **used in conjunction** with previous embedding algorithms like LINE, Node2vec and DeepWalk. The original paper reported marked improvements

of up to *14%* in classification tasks when combining HARP with various graph embedding methods: a significant leap forward.

In Essence

I've definitely missed a bunch of algorithms and models, especially since the recent explosion of interest in Geometric Deep Learning and Graph Learning has led to new contributions popping up in publications almost daily.

In any case, Graph embedding methods are a simple but very effective method of transforming graphs into the optimal format for a machine learning task. Due to their simplicity, they are often quite **scalable** (at least compared to their convolutional counterparts), and are easy to implement. They can be applied to most networks and graphs without sacrificing performance or efficiency. **But can we do better?**

Next up is a dive into the complex and elegant world of **Graph Convolutions**!

Key Takeaways

- Graph embedding techniques **take graphs and embed them in a lower dimensional continuous latent space** before passing that representation through a machine learning model.
- Walk embedding methods **perform graph traversals with the goal of preserving structure and features** and aggregates these traversals which can then be passed through a recurrent neural network.
- Proximity embedding methods use **Deep Learning methods and/or proximity loss functions to optimize proximity**, such that nodes that are close together in the original graph are likewise in the embedding.
- Other approaches use methods like **graph coarsening to simplify the graph before applying an embedding technique on the graph**, reducing complexity while preserving structure and information.

Need to see more content like this?

*Follow me on **LinkedIn**, **Facebook**, **Instagram**, and of course, **Medium** for more content.*

*All my content is on **my website** and all my projects are on **GitHub***

*I'm always looking to meet new people, collaborate, or learn something new so feel free to reach out to **flawnsontong1@gmail.com***

Upwards and onwards, always and only 🚀

Machine Learning | Artificial Intelligence | Entrepreneurship | Startup | Technology

Follow

Written by Flawnson Tong

1.2K Followers · Writer for Towards Data Science

Using machine learning to accelerate science one step at a time :)

More from Flawnson Tong and Towards Data Science

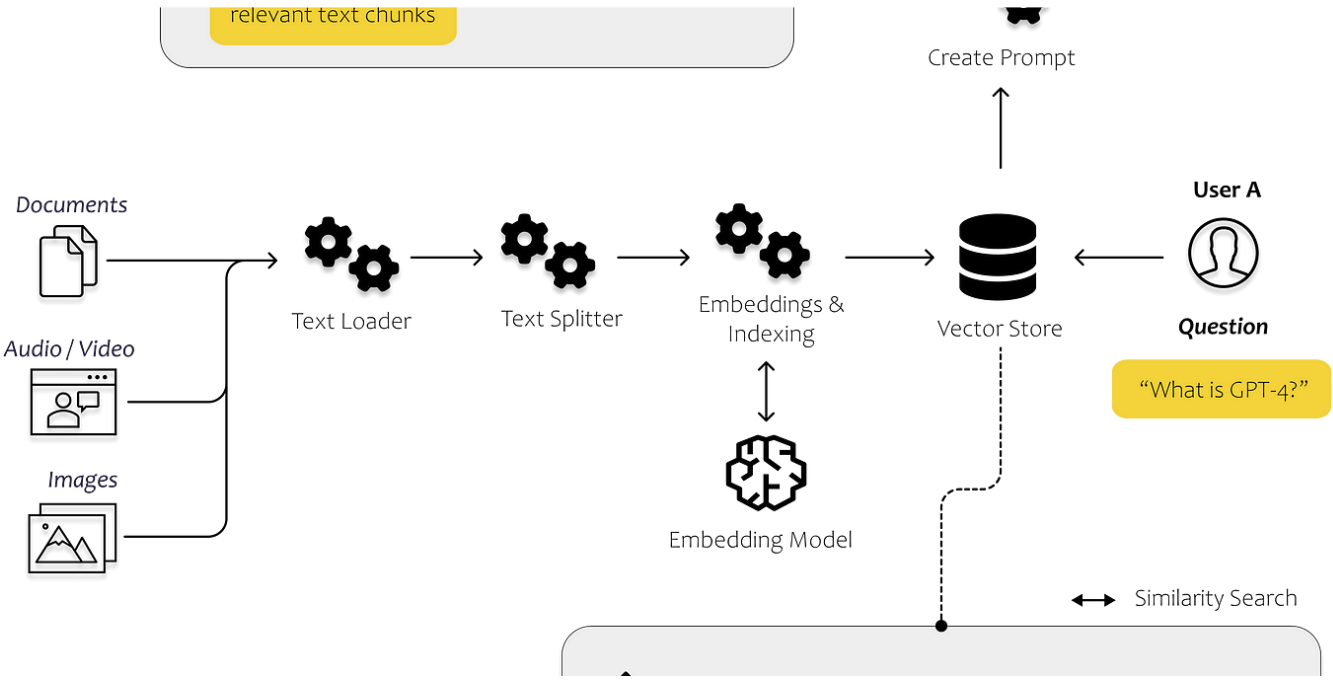
Flawnson Tong in Towards Data Science

Graph Theory and Deep Learning know-hows

Graph Learning and Geometric Deep Learning — Part 0

12 min read · Apr 23, 2019

1.7K 11



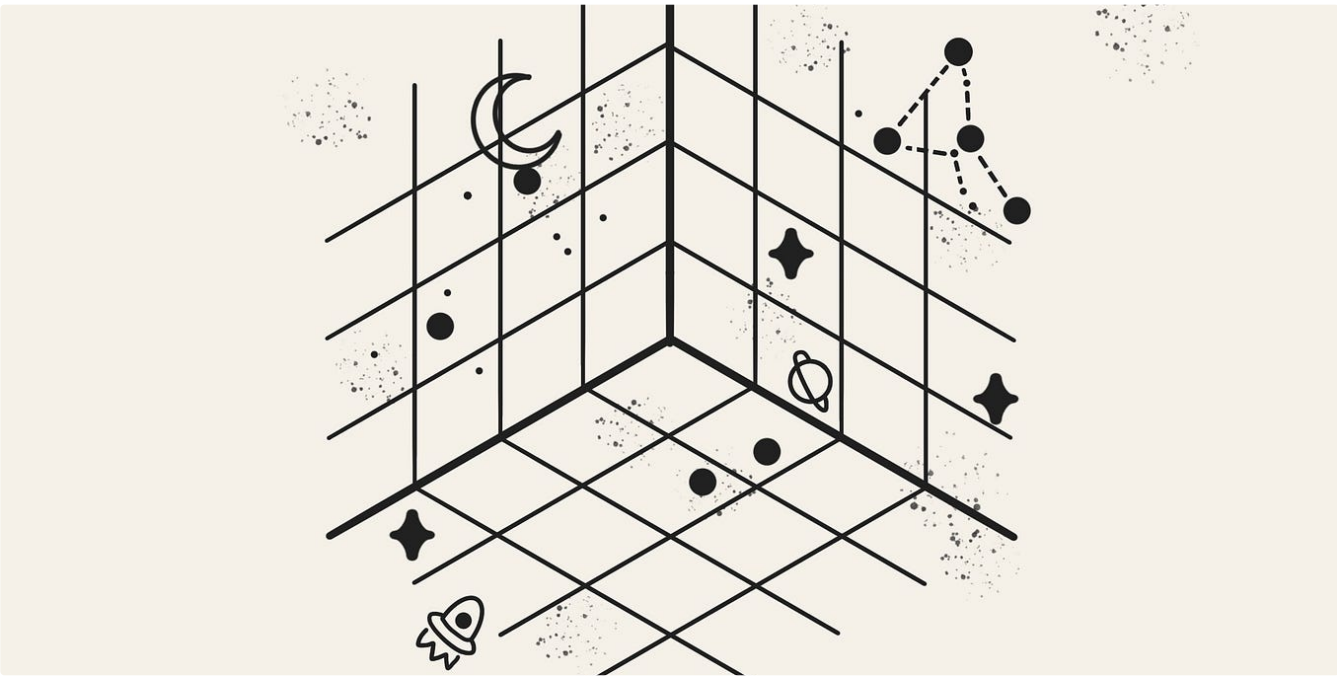
Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

· 26 min read · Jun 22

3.3K 28



Leonie Monigatti in Towards Data Science

Explaining Vector Databases in 3 Levels of Difficulty

From noob to expert: Demystifying vector databases across different backgrounds

· 8 min read · Jul 4

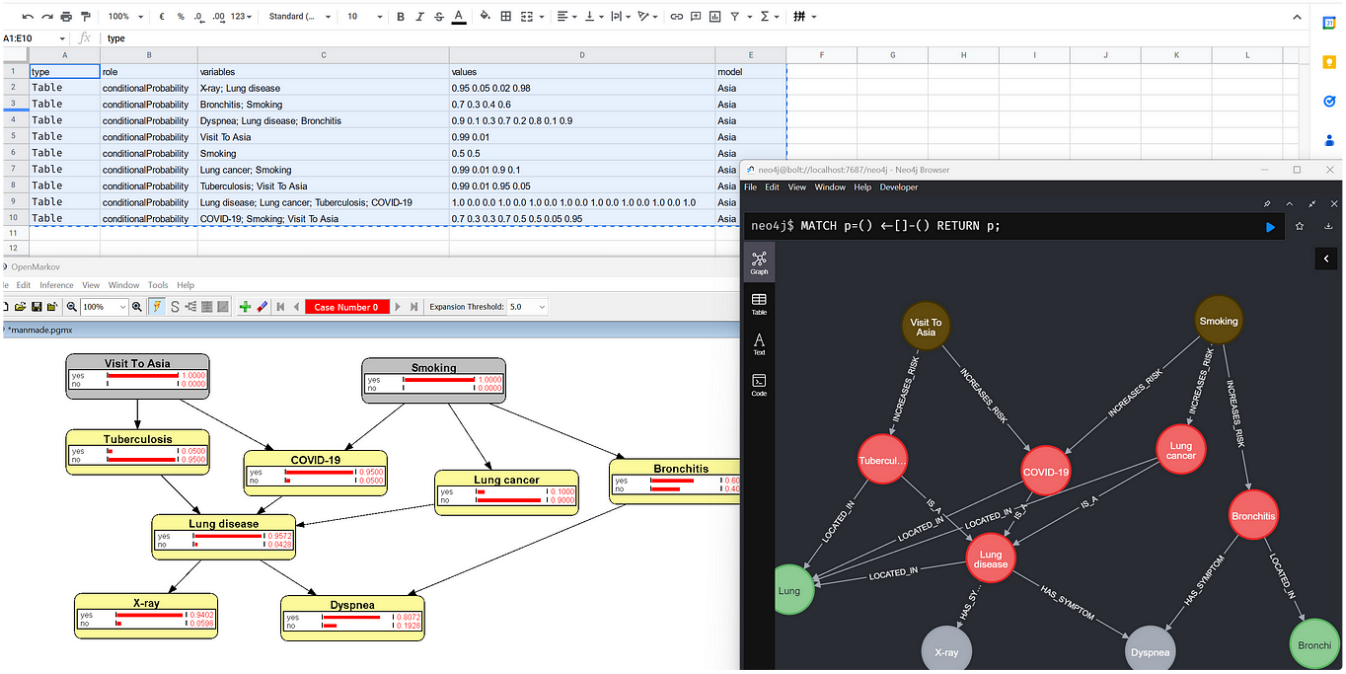
1.7K 18

Flawnson Tong

What is Geometric Deep Learning?

Deep Learning 🤖 on graphs and in 3D

9 min read · Apr 18, 2019

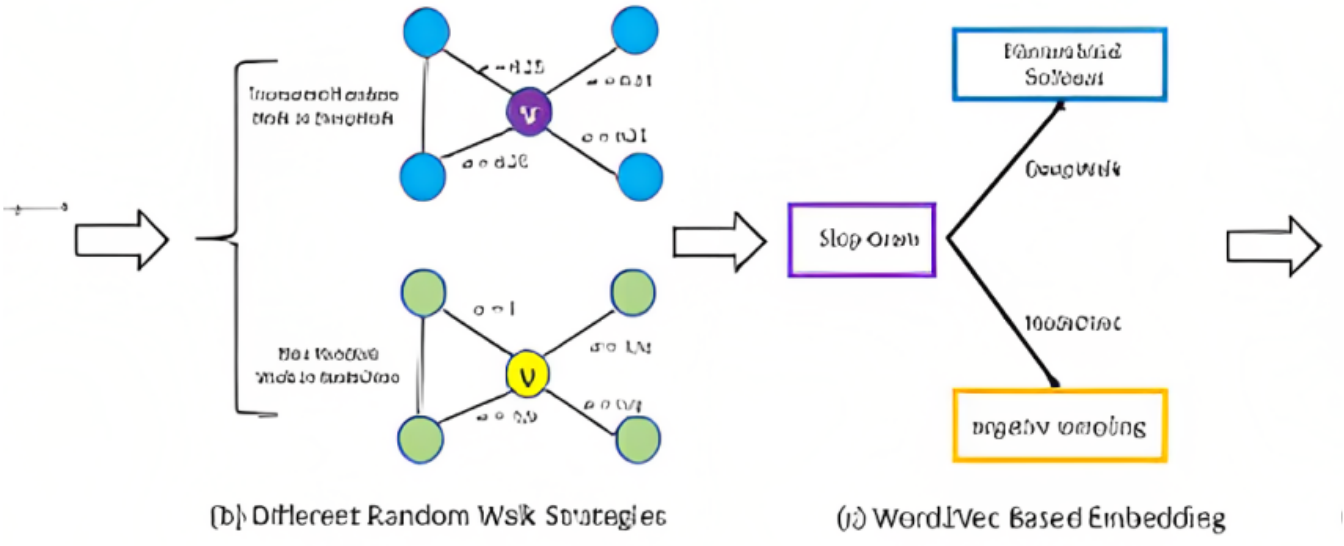


Sixing Huang in Geek Culture

How to Build a Bayesian Knowledge Graph

Store data in Google Sheets, visualize the knowledge graph in Neo4j, and do Bayesian reasoning with OpenMarkov

· 10 min read · Feb 5



(b) Different Random Walk Strategies

(c) Word2Vec Based Embedding

Tejpal Kumawat

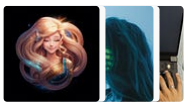
Deep Walk and Node2Vec: Graph Embeddings

Investigating Node2Vec and DeepWalk to extract embeddings from graphs

6 min read · Mar 14

10

Lists



ChatGPT prompts

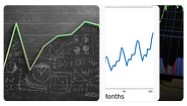
22 stories · 165 saves

AI Regulation

6 stories · 49 saves

ChatGPT

21 stories · 64 saves



Predictive Modeling w/ Python

18 stories · 169 saves

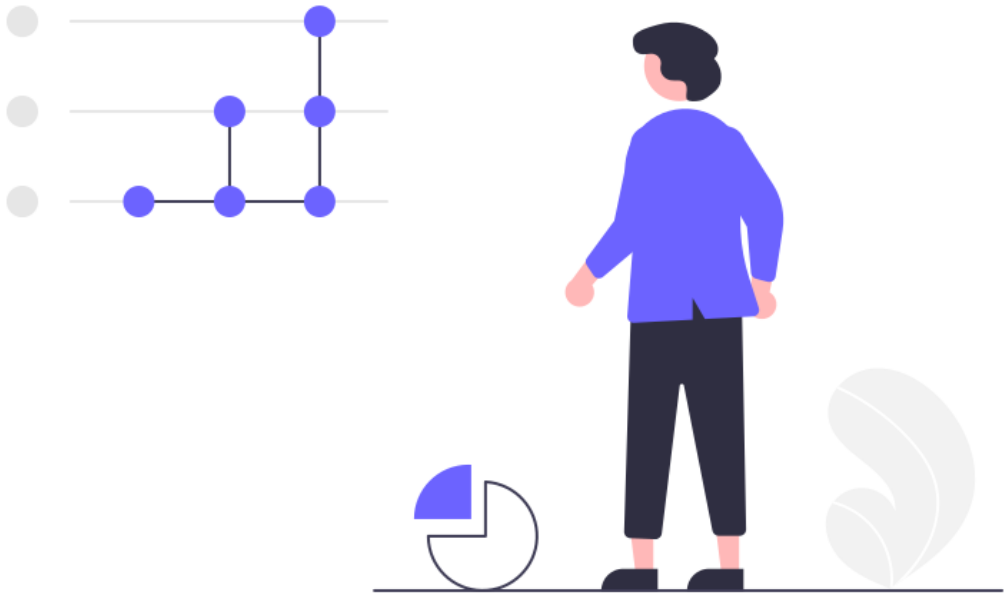
Prof. Simon J. Preis in Towards Data Science

Are Expert Systems Dead?

A review of recent trends, use cases and technologies

12 min read · Mar 16

33



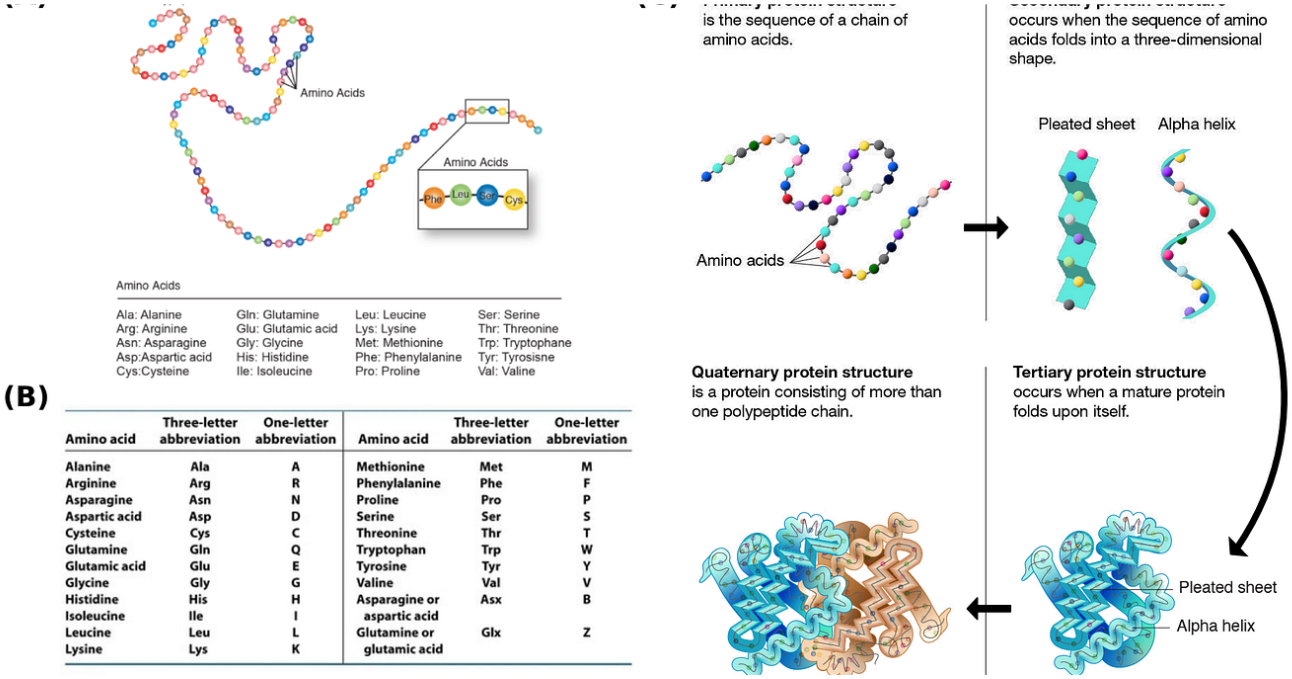
Ravenspike in Dev Genius

Analyzing graph networks: Utilizing advanced methods

Story overview

· 16 min read · Jun 5

29



Amino acid	Three-letter abbreviation	One-letter abbreviation	Amino acid	Three-letter abbreviation	One-letter abbreviation
Alanine	Ala	A	Methionine	Met	M
Arginine	Arg	R	Phenylalanine	Phe	F
Asparagine	Asn	N	Proline	Pro	P
Aspartic acid	Asp	D	Serine	Ser	S
Cysteine	Cys	C	Threonine	Thr	T
Glutamine	Gln	Q	Tryptophan	Trp	W
Glutamic acid	Glu	E	Tyrosine	Tyr	Y
Glycine	Gly	G	Valine	Val	V
Histidine	His	H	Asparagine or aspartic acid	Asx	B
Isoleucine	Ile	I			
Leucine	Leu	L	Glutamine or glutamic acid	Glx	Z
Lysine	Lys	K			

A Aviv Korman in Stanford CS224W GraphML Tutorials

Prediction of Protein Movement Upon Ligand Binding Using Equivariant Graph Neural Networks

Harnessing Graphein to transform protein structures into PyTorch-Geometric-ready residue-level spatial graphs and deploying equivariant...

12 min read · May 14

Algorithm 1 Learning TransE

input Training set $S = \{(h, \ell, t)\}$, entities and rel. sets E and L, margin γ, embeddings dim. k.

initialize $\boldsymbol{\ell} \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each $\ell \in L$

$\boldsymbol{\ell} \leftarrow \boldsymbol{\ell} / \|\boldsymbol{\ell}\|$ for each $\ell \in L$

$\mathbf{e} \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each entity $e \in E$

loop

  $\mathbf{e} \leftarrow \mathbf{e} / \|\mathbf{e}\|$ for each entity $e \in E$

  $S_{batch} \leftarrow \text{sample}(S, b)$ // sample a minibatch of size b

  $T_{batch} \leftarrow \emptyset$ // initialize the set of pairs of triplets

  for $(h, \ell, t) \in S_{batch}$ **do**

    $(h', \ell, t') \leftarrow \text{sample}(S'_{(h,\ell,t)})$ // sample a corrupted triplet

    $T_{batch} \leftarrow T_{batch} \cup \{((h, \ell, t), (h', \ell, t'))\}$

  end for

  Update embeddings w.r.t. $\sum_{((h,\ell,t),(h',\ell,t')) \in T_{batch}} \nabla \left[\gamma + d(\boldsymbol{h} + \boldsymbol{\ell}, \boldsymbol{t}) - d(\boldsymbol{h'} + \boldsymbol{\ell}, \boldsymbol{t'})\right]_+$

end loop

S Shreyash Pandey

Comparison of Knowledge Graph Embeddings (KGE) models

Knowledge Graph Embeddings (KGE) are models that attempt to learn the embeddings, and vector representation of nodes and edges, by taking...

9 min read · Feb 11

 18