知乎 first published in Deep learning from proficiency to entry

Deep walk entry aids understanding and code analysis

classmate li
Former takeaway rider with good physical strength. In April 2018, he was the single champi...

0.intro

- random walk
- Word2vec entry aids understanding
- Analysis of deep walk papers
- deep walk code combat

What can deep walk do?

If your research is graph-based (road network, human relationship, server network, academic paper citation network), you may want to do something with the nodes in the graph, such as wanting to measure the similarity of points in the graph, specifically Click on a similarity study of stations on a road network. The traditional manual feature extraction method may do some linear combination of the degree of connectivity (degree) of the station on the road network and the distance between the stations. Intuitively, the stations at the core position extend in all directions, with high density, and rer
graph. Or social-network studies the

It is because the points in the graph cannot be directly quantified as the input of the model. Human eyes can get a rough idea of the points on the map. This map is directly given to the computer, but additional feature construction is required. The **original intention of deep walk is to automate this mapping process with neural networks** and express the nodes in a more effective way. neighborhood information.

In deep learning, Embedding specifically refers to using a low-dimensional vector to represent an entity, which can be a word, an item (Item2Vec), or a node in a network relationship (graph embedding). Deep walk is the beginning of graph embedding, and it is used as a baseline by follow-up research. This method uses **random walk** to generate sequences from graphs, uses sequences as sentences, and converts them into vectors through **word2vec to complete Embedding.** Random walk and word2vec will be introduced separately later.

Let's first look at a simple network structure: the character relationship diagram in Game of Thrones

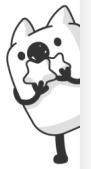
data

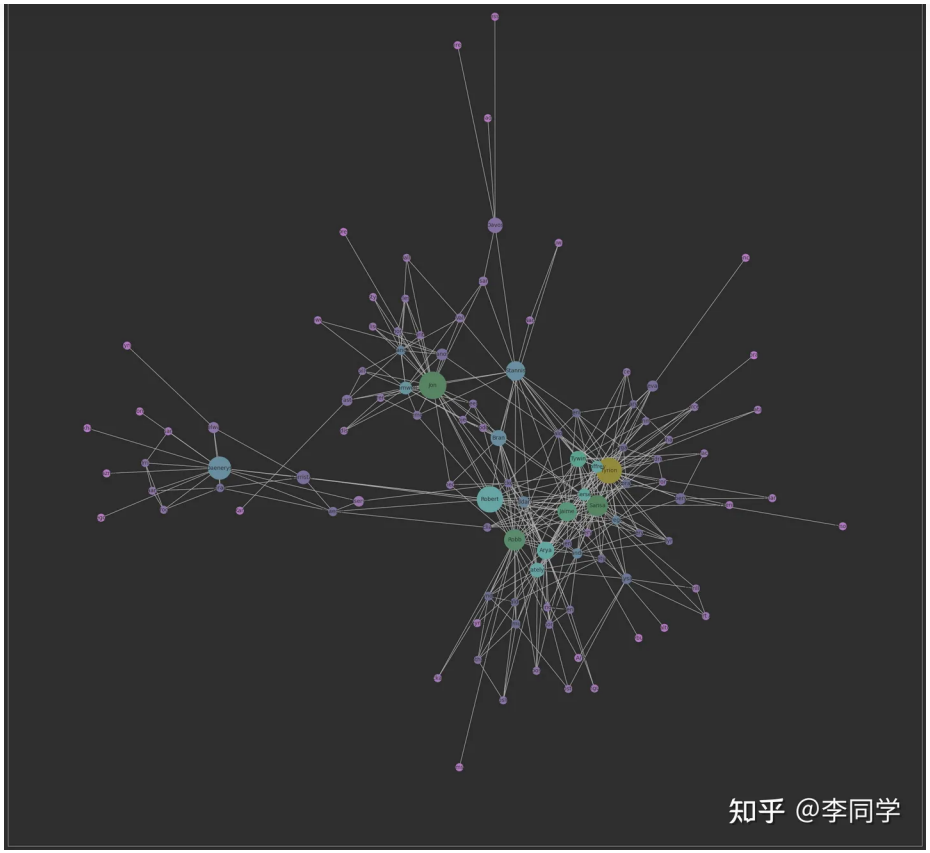
https://github.com/sandrohr95/FIMED-Analysis/blob/70532be76d4d4beaadb18fd350e2...
github.com/sandrohr95/FIMED-Analysis/blob/70532be7...

The characters in the first two columns, and the third column is the number of times these two characters have been in contact in the novel. The more the characters, the closer the relationship.

```
import networkx as nx
import matplotlib.pyplot as plt

# data/got.txt是数据位置
G_got = nx.read_weighted_edgelist("data/got.txt", create_using = nx.Graph(), n
pos = nx.spring_layout(G_got)
betCent = nx.betweenness_centrality(G_got, normalized=True, endpoints=True)
node_color = [20000.0 * G_got.degree(v) for v in G_got]
node_size =  [v * 10000 for v in betCent.values()]
plt.figure(figsize=(30,30))
nx.draw_networkx(G_got, pos=pos,font_size = 10 ,node_color=node_color,
                 node_size=node_size,font_color='white')
```



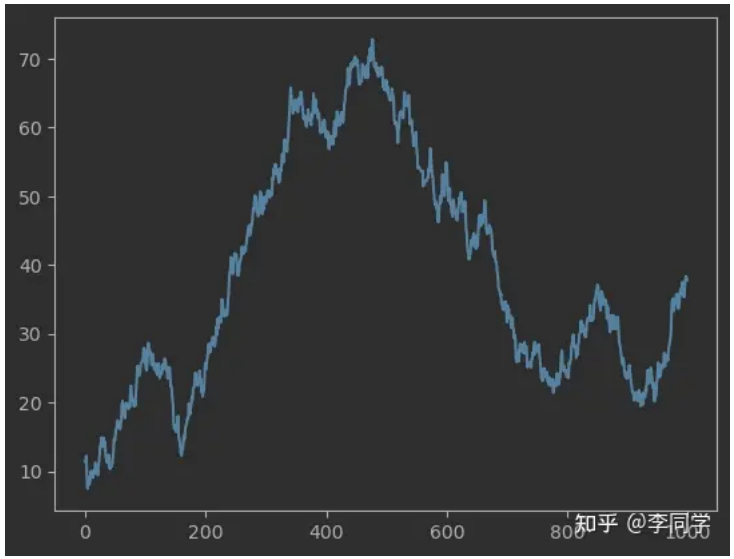
1. random walk

Random walk is a random process. This process can occur on structures such as real number lines and graphs. Continuous random walks constitute a trajectory. Random walks can be used to explain many observed phenomena, such as stock prices, animal walking paths, and the motion of molecules in the air, and are fundamental statistical models for recording random activity.

Example 1: 1D Random Walk - Stock Volatility

For example, a stock is initially priced at 10 yuan, a random walk of 1000 trading days, and the price changes follow a normal distribution.

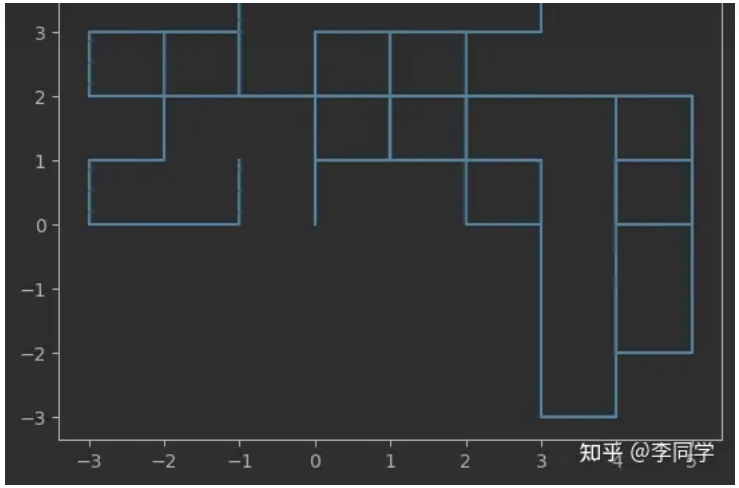
```
prices = []
for i in range(1000):
    price += np.random.randn()
    prices.append(price)
plt.plot(prices)
```



Example 2: Two-dimensional random walk

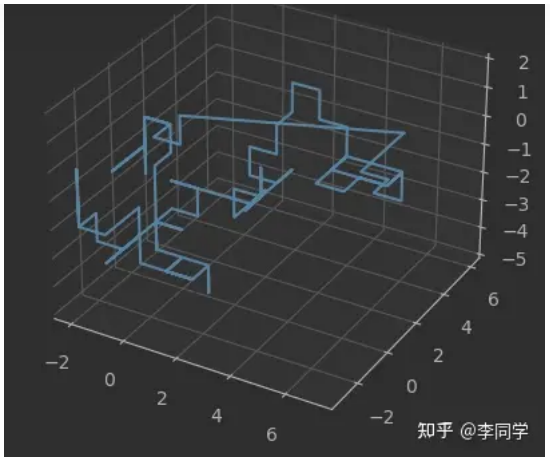
Randomly choose a direction walk on a trellis graph

```
def dunk_walk(step_num):
    pos_x = 0
    pos_y = 0
    actions = [[0, 1], [0, -1], [1, 0], [-1, 0]]
    trace_x = [pos_x]
    trace_y = [pos_y]
    for i in range(step_num):
        pos_x,pos_y = np.array((pos_x,pos_y)) + actions[np.random.choice(4)]
        trace_x.append(pos_x)
        trace_y.append(pos_y)
    plt.plot(trace_x,trace_y)
    plt.show()
dunk_walk(100)
```



Example 3: 3D random walk

```
def dunk_bird(step_num):
    cor = [0,0,0]
    trace = [cor]
    actions = [-1, 1]
    for i in range(step_num):
        change_dim = np.random.choice(3)
        cor[change_dim] += np.random.choice(actions)
        # copy,否则复制的是内存地址
        trace.append(cor.copy())
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.plot3D([cor[0] for cor in trace],[cor[1] for cor in trace],[cor[2] for c
    ax.legend()
    plt.show()
dunk_bird(100)
```



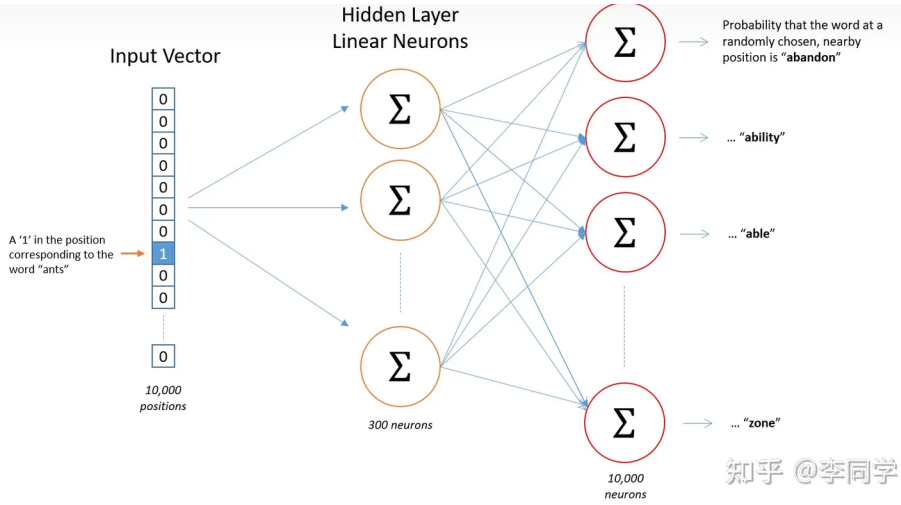
> A drunk man can come home, but a drunk bird can be lost forever

As the dimension increases, the probability of returning to the starting point will become lower and lower

2. word2vec

2.1 Principle

Word2vec is an algorithm used in NLP. By mapping words into vectors, it captures the context of words before and after, word similarity and other properties. It can be used for information retrieval, topic discrimination, sentiment analysis, etc. After mapping, it can be easily fed into machine learning algorithms, just like using RGB values to represent pictures. Word2vec can be imagined as a combination of words converted into N-dimensional values . The initialization state of Embedding can be understood as a set of random numbers. With the optimization of the algorithm and continuous iterative update, when the network reaches the convergence state, the parameters of each layer in the network will be relatively solidified, thus obtaining the hidden layer weight table. At this point, it is equivalent to getting the embedding we want. In practical applications, the weight parameter of the penultimate layer of the neural network is usually regarded as the embedding of the corresponding sample.



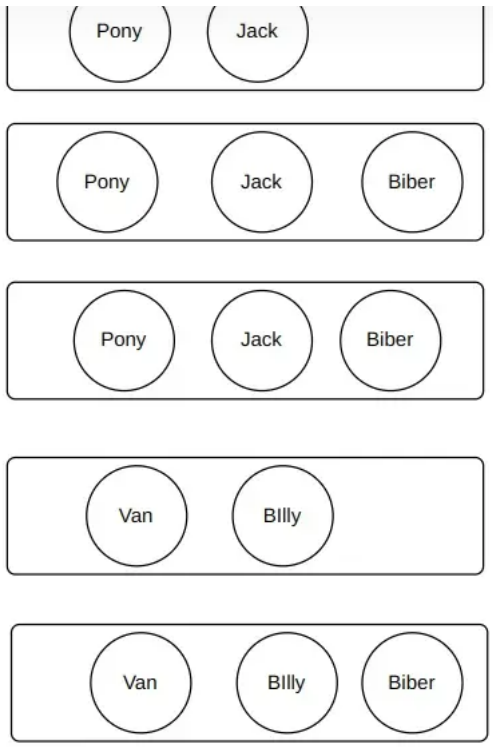
When training the network, the input is a one-hot vector representing the input word, and the training output is also a one-hot vector representing the output word. The output vector will actually be a probability distribution (that is, a bunch of logits). The trained hidden layer parameters can be used as embedding, which is similar to the autoencoder. The logical basis of word2vec is that words adjacent to a sentence are related to each other. After a large number of sentences are trained, the hidden layer parameters are taken out as word vectors. Imagine that after the NN without activation function is trained, it can reflect the logic of human sentences. If the logic is correct, the word vector is also correct.

2.2 Assisted understanding

After reading several materials, it seems that there is nothing very straightforward. I personally draw a few pictures to show the understanding process

Word: Pony, Jack, Biber, Van, Billy

sentences: The following are scenes of different characters appearing in different videos



Embedding task: given sentences, embedding to three-dimensional vector

Prediction task: use the current character to predict its next character

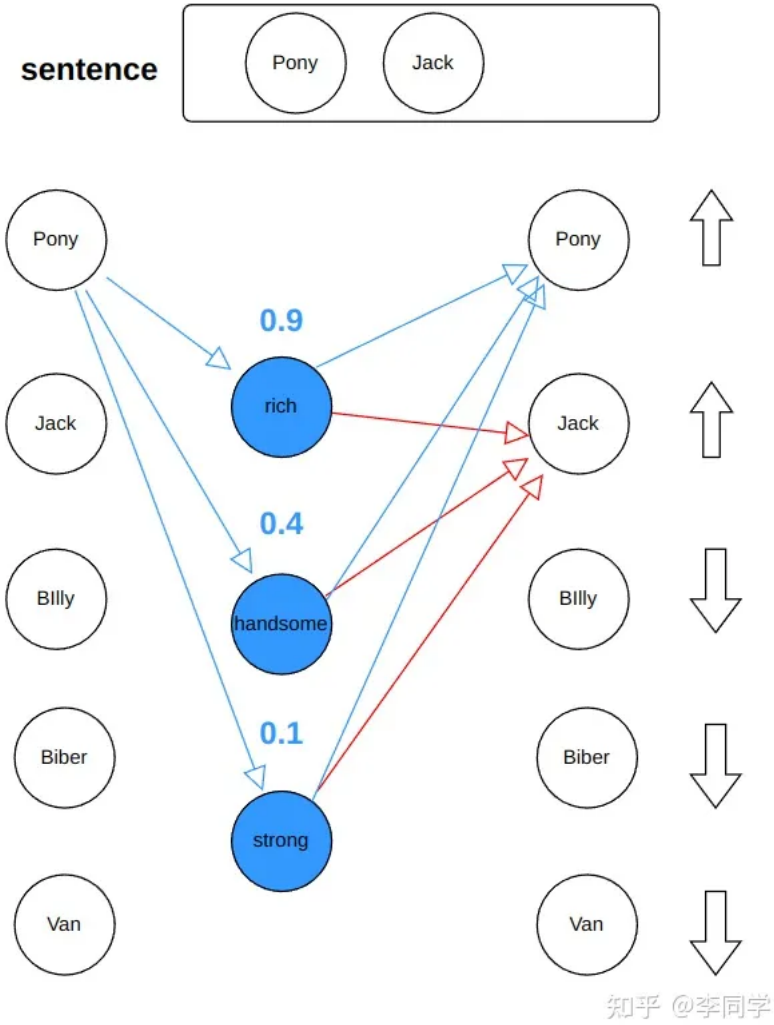
Input: one-hot

Output: logit, which is the probability distribution. After training, input Jack, output Jack's logit is the largest, and do softmax

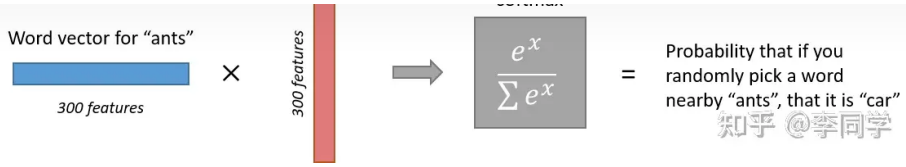
Suppose we know from the perspective of God that the three features after embedding represent rich, handsome, and strong.

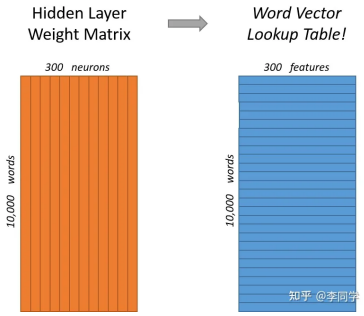
The picture below is a certain training, input pony: (1,0,0,0,0)

Because pony and jack often appear in the same sentence, the logit of a well-trained jack cannot be small. At this time, Jack ir



On the Pony feature, Jack's output weight has also been trained, because they are similar and often appear in a sentence, and Jack wears Pony shoes to train the output weight more often, and in turn, their input weight matrix will become more and more Similar, the picture above should be seen together with the picture below





But one-hot has a problem. When there are many words, the input layer-hidden layer, hidden layer-output layer weights are too large to explode, and the gradient drops slowly and is easy to overfit. Therefore, there are still tricks, such as negative sampling and hierarchical softmax.

Then there are skip-gram and CBOW two training methods, one is to use the surrounding words to predict the words in the middle, and the other is to use a word to predict the surrounding words.

2.3 Small demo

The word2vec introductory understanding has come to this level first, and without further ado, quickly adjust the package and start : use gensim's built-in word2vec to train words into 2-dimensional data and visualize them

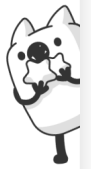
The corpus randomly creates 6 sent

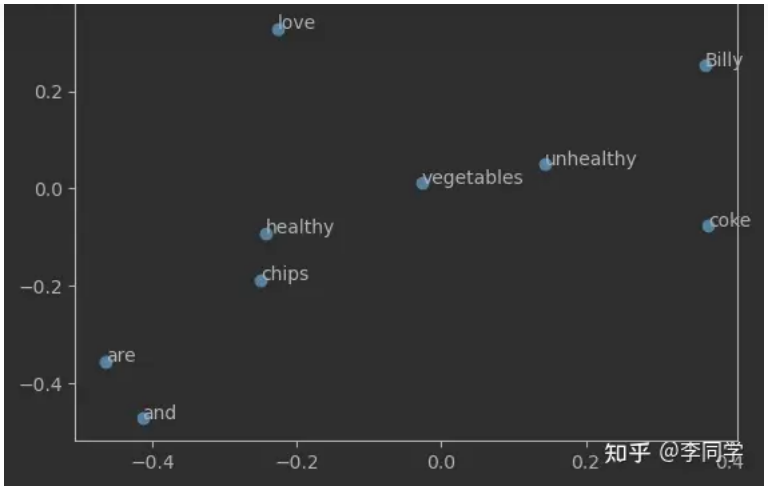
```
    ['I','love', 'chips'],
    ['I','hate', 'vegetables'],
    ['Billy','hate', 'vegetables'],
    ['chips','and','coke','are','unhealthy'],
    ['vegetables','are','healthy']
]


import gensim
import matplotlib.pyplot as plt

model = gensim.models.Word2Vec(sentences=sentences, vector_size=2, min_count=1,

enum_word = set()
for sentence in sentences:
    for word in sentence:
        enum_word.add(word)
cor_2d = []
labels = []
for word in enum_word:
    labels.append(word)
    cor_2d.append(model.wv[word].tolist())
fig, ax = plt.subplots()
x = [c[0] for c in cor_2d]
y = [c[1] for c in cor_2d]
ax.scatter(x, y)
for i, wd in enumerate(labels):
    ax.annotate(wd, (x[i], y[i]))
```



The small case is purely for entertainment, and the corpus is too small. If you build tasks yourself, you need to do a reasonable amount.

From the official documentation:

> To observe strong regularities in the word vector space, it is needed to train the models on large data set, with sufficient vector dimensionality.

Then use someone else's training to download a model trained by Google through a large number of articles, which is trained by 200 billion words

Because the vector output by this model is 300-dimensional, it cannot be directly visualized. First, draw a correlation matrix between words to see, and also judge whether your words are i the Google corpus.

```
from matplotlib.colors import Normalize
# 模型所在文件夹 /home/li/
wv = gensim.models.KeyedVectors.load_word2vec_format("/home/li/GoogleNews-vect

lines = []
for lb in labels:
    line = []
    for lb2 in labels:
```

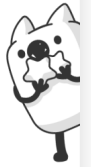
```
fig, ax = plt.subplots()
cmap = 'cool'
im = ax.imshow(lines, cmap=cmap)
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")
cmap = plt.colormaps[cmap]
fig.colorbar(plt.cm.ScalarMappable(norm=Normalize(0, 1), cmap=cmap),
             ax=ax)
ax.set_xticks(range(len(lines)), labels=labels)
ax.set_yticks(range(len(lines)), labels=labels)
fig.tight_layout()
```

This time it is much more reasonable, healthy and unhealthy, love and hate are more similar.

Then use sklearn's PCA and TSNE to further reduce the dimensionality for visualization.

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

data = []
```

```
X_tsne = TSNE(n_components=2,perplexity = 5).fit_transform(np.array(data))
X_pca = PCA(n_components=2).fit_transform(np.array(data))


plt.figure(figsize=(10, 5))
plt.subplot(121)
plt.scatter(X_pca[:, 0], X_pca[:, 1],label = "PCA")
for i, wd in enumerate(labels):
    plt.annotate(wd, (X_pca[:, 0][i], X_pca[:, 1][i]))
plt.legend()
plt.subplot(122)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=['r' for _ in range(len(X_tsne[:, 1])
plt.legend()
for i, wd in enumerate(labels):
    plt.annotate(wd, (X_tsne[:, 0][i], X_tsne[:, 1][i]))

plt.show()
```

把刚才那个简单小例子做成一个有向图(不是特别准确的图)，是不是能感觉到deepwalk要在图上干什么了

3.deep walk

3.1.deep walk 论文

原始论文地址

deep walk 是基于random walk的图转向量算法，俗称图的embedding。Embedding（嵌入）在数学上表示的是一个映射关系（函数）。映射需要满足injective和structure-preserving。

- Injective（单射函数），对于每个 Y 只有唯一的 X 对应
- structure-preserving（结构保存），映射前在X空间X1<X2，映射后在Y空间上Y1<Y2。

给定一个图，用embedding压缩成高维向量，比如下面这个空手道成员关系图，压缩到2维之后可视化放到平面上。可以看到原来的社区信息，成员关系似乎很好地保留了下来。

如果连通图的度分布遵循幂律分布（无标度网络,少数节点拥有极其多的连接），则在短随机游走中顶点出现的频率也遵循幂律分布（密度函数是幂函数）。自然语言中的词频也遵循类似的分布。

语言模型里面的任务是估计一个特定单词序列在语料库中出现的可能性。比如用前几个单词预测最后一个单词。一句话里面有i个词 W^n_1=(w_0,w_1,...,w_{n-1}) 建模的目标就是在所有训练语料库中使Pr(w_n|(w_1,w_2,...,w_{n-1})) 最大化。将上述概念扩展到网络中,就是在给定之前的随机游走中访问过的所有顶点的情况下，估计观测到顶点vi可能性最大化Pr(v_i|(v_1,v_2,...,v_{i-1})) 。

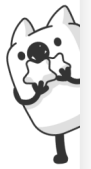
节点无法计算，改用映射（Skip-Gram），损失函数如下

minimize -logPr(\{v_{i-w},...,v_{i+w}\} / v_i|\Phi(v_i))

In a random walk process, given vi, the probability of points appearing within the w window, there is no order between these vertices, and the author said that this is very suitable for social relations.

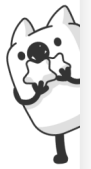
The sentences in word2vec before are formed by random walks in the graph. The construction of a sentence is a random walk starting from the nodes of the graph, and each walk is equivalent to "saying a sentence".

deep walk algorithm



The build binary tree above is to build Hierarchical Softmax, convert the corpus into a Huffman tree, and assign short binary codes to frequent words to reduce complexity. If the corpus is very large, the output calculation softmax and other operations are too expensive, and the hierarchical softmax is used. Treat the vertices as leaves of the Huffman tree, maximizing the path. This is a partial engineering application (in the company, it is reasonable to say that it is a development activity), so I won't go in-depth here.

The SkipGram algorithm updates the Φ matrix every time a probability is calculated within the window

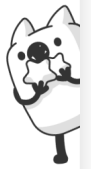


A specific example from the text

Select the fourth vertex as the starting point, swim out of 4-3-1-5-1, the window is 1, when it is the turn of 1 point, the one-hot of the v1 vertex becomes Φ(v1) through the input vector Φ , now to maximize the conditional probabilityPr(u_k|\Phi(v_j)) , v3 and v5 are in the window.

3.2.deep walk code analysis

I found one based on gensim, with more stars



First solve the problem of running, because the time is relatively long, the API of some packages has changed, so the code is changed as follows

```
graph.py  from collections import Iterable 改成from collections.abc import Ite
skipgram.py from collections import Mapping 改成 from collections.abc import M
```

Run deepwalk, the input is karate.adjlist (karate club data, the first in each line is the personnel number, and the others are other people this person knows), and the embedding result is output

```
python -m deepwalk --input example_graphs/karate.adjlist --output karate.embedd
```

The main logic is in this function, look at it step by step, and write in the comments

```
def process(args):
  #根据入参的文件格式构建Graph，支持三种 一对多(邻接表)，一对一（edge）,matlab矩阵文件
  if args.format == "adjlist":
    G = graph.load_adjacencylist(args.input, undirected=args.undirected)
  elif args.format == "edgelist":
    G = graph.load_edgelist(args.input, undirected=args.undirected)
  elif args.format == "mat":
    G = graph.load_matfile(args.input, variable_name=args.matfile_variable_name
  else:
    raise Exception("Unknown file format: '%s'.  Valid formats: 'adjlist', 'edg

  print("Number of nodes: {}".format(len(G.nodes())))

  #random walk数 = node数 * 每个node Walk数
  num_walks = len(G.nodes()) * args.number_walks

  print("Number of walks: {}".format(num_walks))

  # 根据步数和总共走多少个sequence计算数据量
  data_size = num_walks * args.walk_length

  print("Data size (walks*length): {}".format(data_size))
  #数据量不超过最大阈值就正常处理
  if data_size < args.max_memory_data_size:
    print("Walking...")
    # random walk生成Sentence
    walks = graph.build_deepwalk_corpus(G, num_paths=args.number_walks,
                                        path_length=args.walk_length, alpha=0,
    print("Training...")
    # sentence投喂 Word2vec
    model = Word2Vec(walks, v
  else:
```

```
    #数据量超过最大阈值，溢写磁盘，防止OOM
    walks_filebase = args.output + ".walks"
    # 磁盘IO序列化
    walk_files = serialized_walks.write_walks_to_disk(G, walks_filebase, num_pa
                                        path_length=args.walk_length, alpha=0,
                                        num_workers=args.workers)

    # 底下这点Counting 不用看了，原因见 https://github.com/phanein/deepwalk/issues/
    print("Counting vertex frequency...")
    if not args.vertex_freq_degree:
      vertex_counts = serialized_walks.count_textfiles(walk_files, args.workers
    else:
      # use degree distribution for frequency in tree
      vertex_counts = G.degree(nodes=G.iterkeys())

    print("Training...")
    walks_corpus = serialized_walks.WalksCorpus(walk_files)
    model = Skipgram(sentences=walks_corpus, vocabulary_counts=vertex_counts,
                     vector_size=args.representation_size,
                     window=args.window_size, min_count=0, trim_rule=None, work

  model.wv.save_word2vec_format(args.output)
```

create corpus function

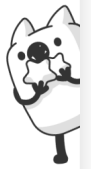
Input parameter G, that is, the constructed graph, the parent class is a dictionary, the key is a point, and the value is the number of other points connected to the changed point

```
def build_deepwalk_corpus(G, num_paths, path_length, alpha=0,
                          rand=random.Random(0)):
    walks = []

    nodes = list(G.nodes())
    #num_paths 是每个Node要走多少次
    for cnt in range(num_paths):
        # 打乱顺序
        rand.shuffle(nodes)
        for node in nodes:
            walks.append(G.random_walk(path_length, rand=rand, alpha=alpha, st

    return walks

def random_walk(self, path_le
```

```
    if start:
        path = [start]
    else:
        # Sampling is uniform w.r.t V, and not w.r.t E
        path = [rand.choice(list(G.keys()))]
    # walk
    while len(path) < path_length:
        #上一个节点
        cur = path[-1]
        # 上一个节点有相邻节点
        if len(G[cur]) > 0:
            #相邻节点随机挑一个
            if rand.random() >= alpha:
                path.append(rand.choice(G[cur]))
            #重新开始 不知是干什么，既然alpha=0，这段也没用
            else:
                path.append(path[0])
        else:
            #上一个节点没有相邻节点了（死胡同：有向图可能出现，或者是孤立节点）
            break
    return [str(node) for node in path]
```

Published on 2023-02-22 20:38 · IP Dependency Japan

Graph Neural Network (GNN) Indicates learning Embedding

The article is included in the following columns

Deep learning from proficiency to entry
Deep learning, distributed, quantitative trading, depending on the mood, maybe a eunuch

recommended reading


New Walkman with 30Kg weight loss——Conclusion...

Ren Ziyu

Guru Appreciation | As I Walked Out One Evening

English Guru

She Walks in Beauty

English poetry

5 Walking bass modes suitable for different piar

Oops sir Posted in Adults,