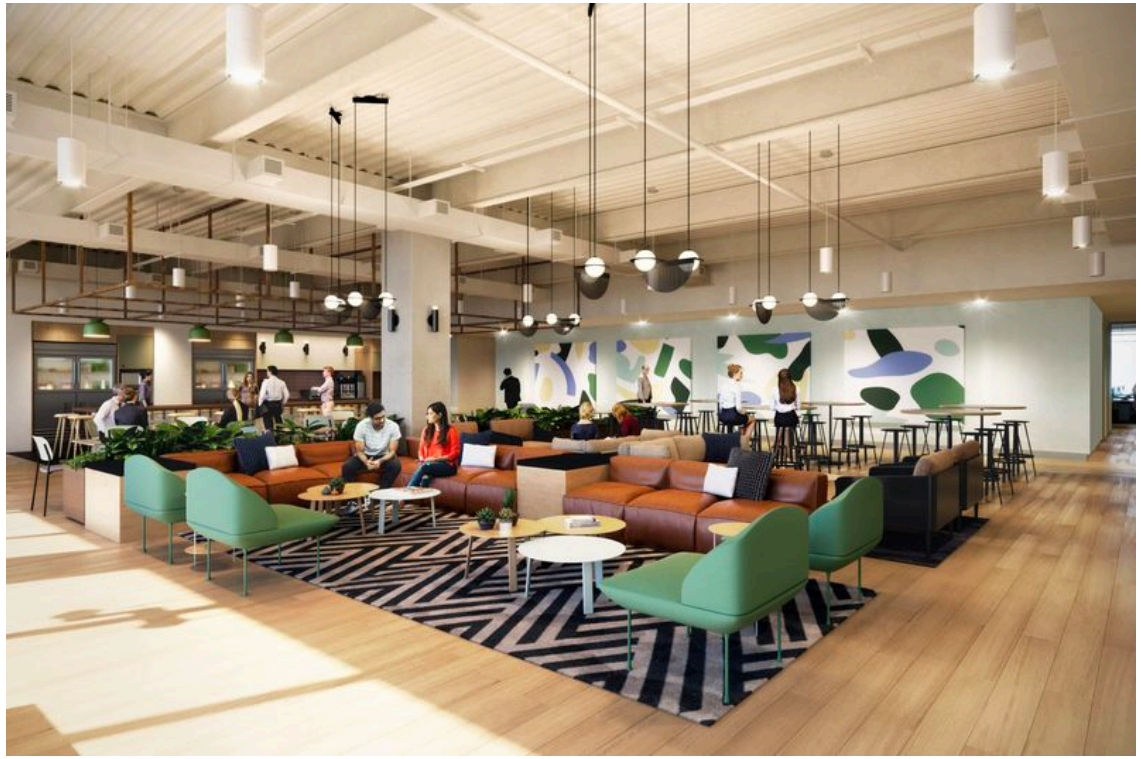
Machine Learning at WeWork:

Creating Community Through Graph Embeddings

Rev Summit | New York 2019

wework

PHYSICAL + DIGITAL DATA



On the Physical side, we use data to optimize how we construct, maintain and design our spaces

-reduce extraneous costs from construction and building logistics

-encourage physical connections through intelligent interior design

-maximize utilization

Physical Space:

Use data to optimize how we construct, maintain and design our spaces

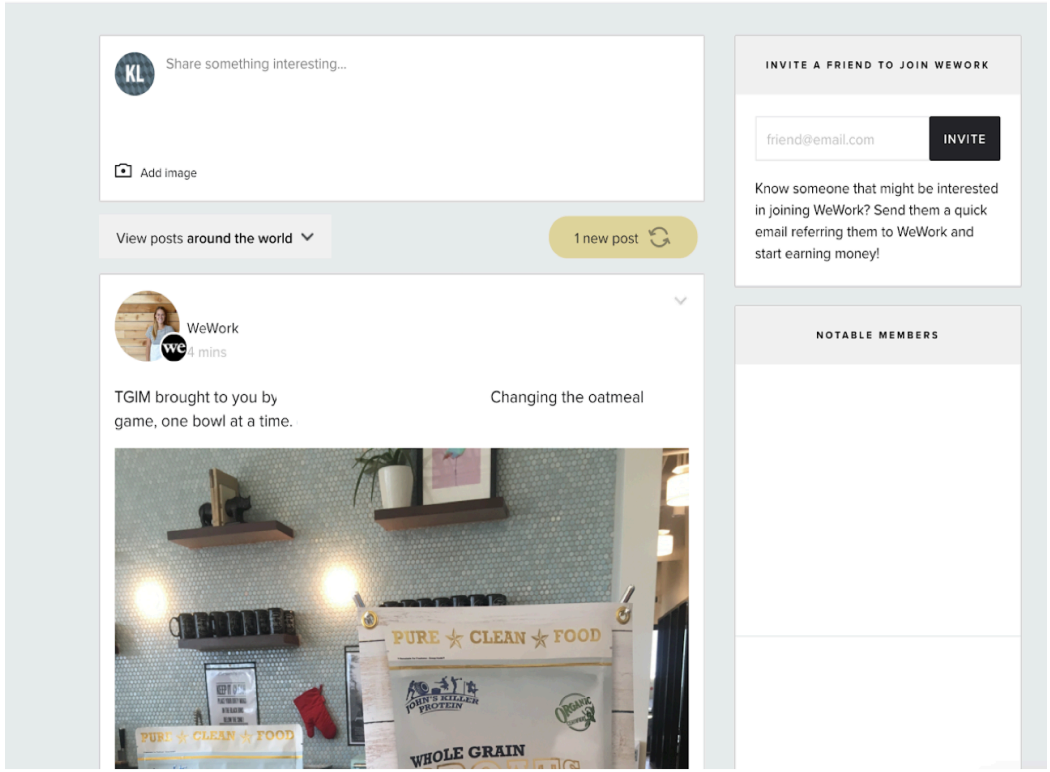
wework

On the Digital, we focus mostly on helping people make connections:

-promote their business

-request help from others

-simply make new friends and connections



Digital Space:

Focus on helping people make connections

wework

In order to better facilitate connections, WeWork started an ML team to focus on rec systems

wework

In order to better facilitate connections, WeWork started an ML team to focus on rec systems








wework

In order to better facilitate connections, WeWork started an ML team to focus on rec systems

-personalized newsfeed
-text classifiers
-entity extraction
-onboarding skills suggestion
-conference rooms recommender
-experiment platform

wework

Graph Embeddings with node2vec

Or: How to recommend members for fun and profit

wework

Graph Embeddings with node2vec

1) Business Use Case

2) Data

3) Model

wework

Use Case: Member Needs

Our members have broadly expressed the desire to:
-meet other members in the community
-make social and professional connections
-feel a sense of “belonging”

wework

Use Case: Member Needs

We are thus interested in facilitating the meeting of members

wework

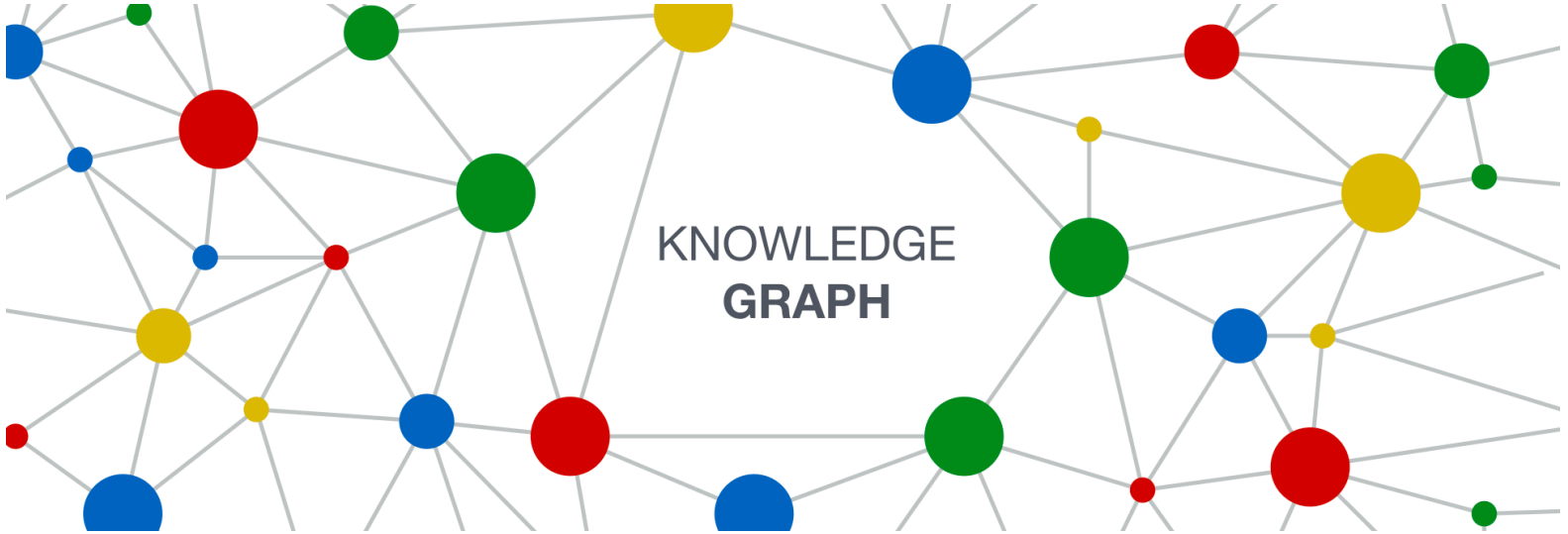
Use Case: Member Needs

How do we create an intelligent member-to-member recommendation service?

?

wework

Data Structure: Member Knowledge Graph



wework

Member Knowledge Graph

A central repository of information about our members

wework

Member Knowledge Graph

A central repository of information about our members

wework

Member Knowledge Graph

A central repository of information about our members

wework

Member Knowledge Graph

A central repository of information about our members

wework

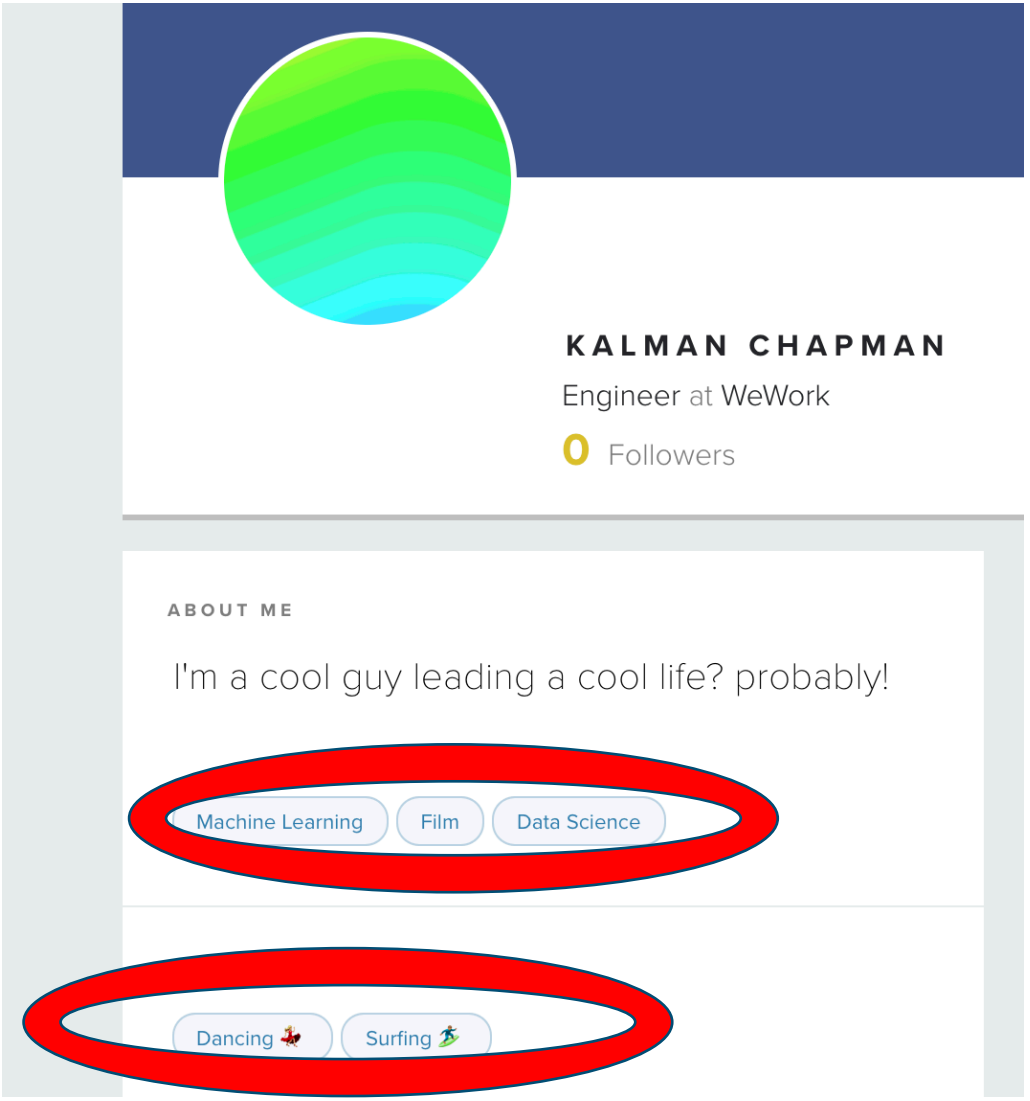
Member Knowledge Graph

We collect data from:

-member profiles
-member interactions on the app
    -posts you've liked
    -events you've bookmarked
-community team notes
-external data sources that we match to our internal data

wework

Member Knowledge Graph



wework

Member Knowledge Graph

All this information is expressed in graph form

wework

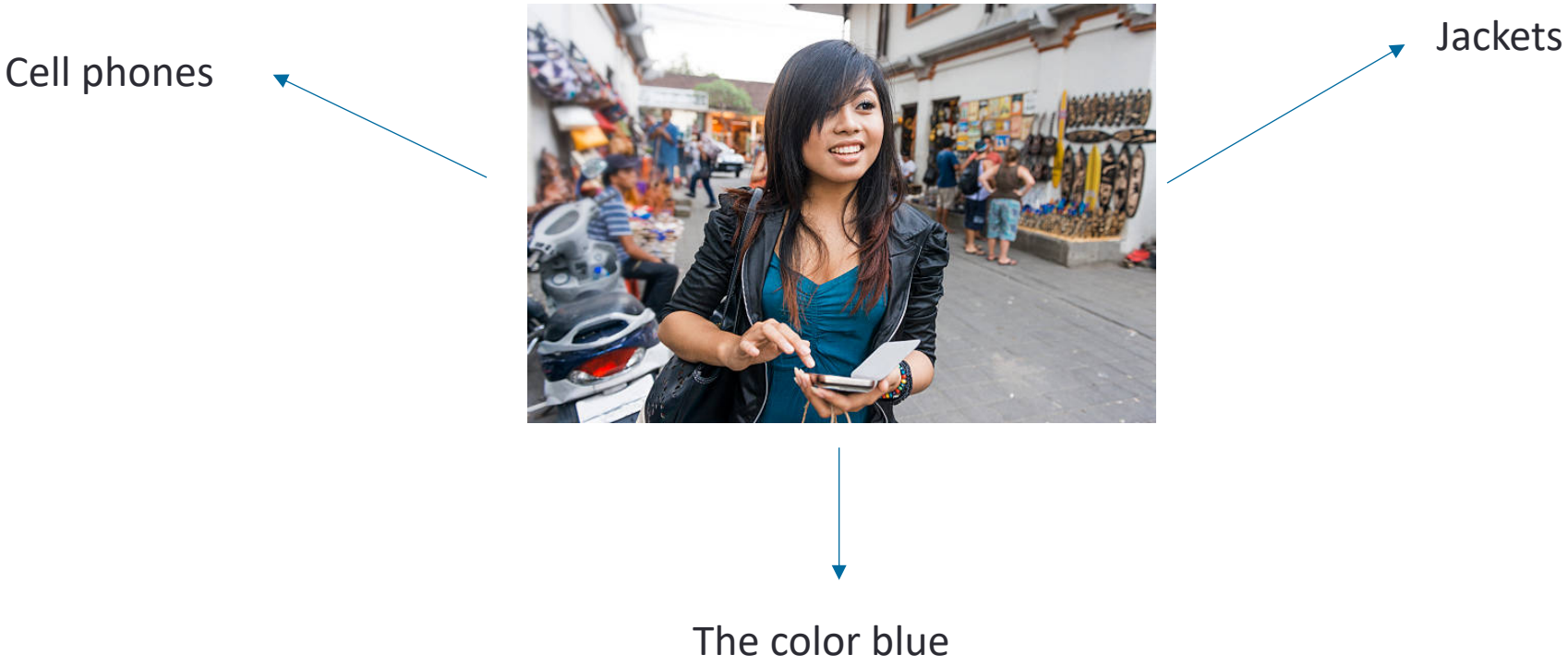
Member Knowledge Graph

All this information is expressed in graph form

wework

Member Knowledge Graph

All this information is expressed in graph form



wework

Member Knowledge Graph

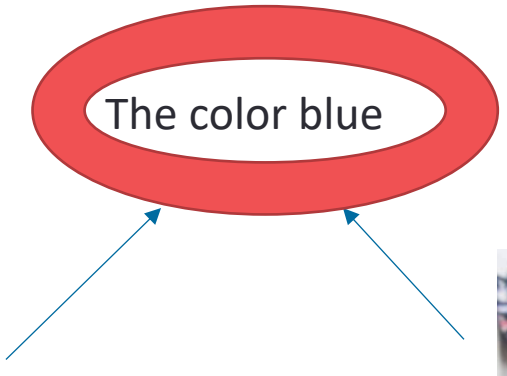
Connecting members through shared skills or interests

The color blue

wework

Member Knowledge Graph

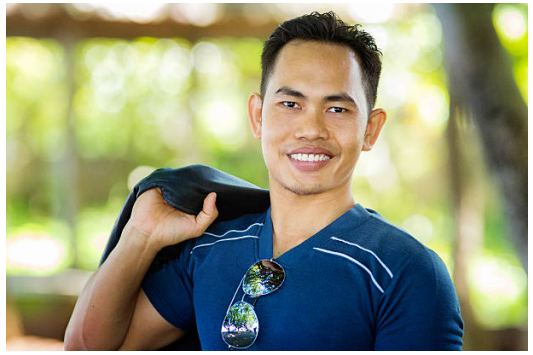
Connecting members through shared skills or interests



wework

Member Knowledge Graph

Connecting members through shared skills or interests



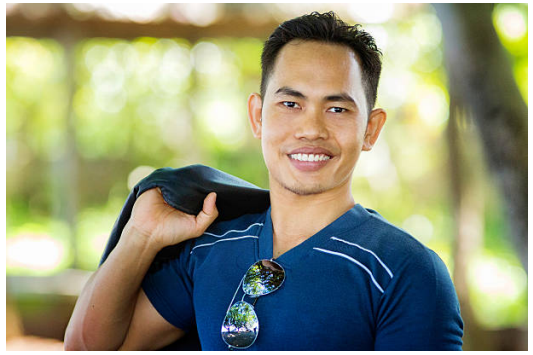
Laundry
The color blue

The color blue

wework

Member Knowledge Graph

We assume that people who have lots of shared skills or interests may be good recommendations for each other



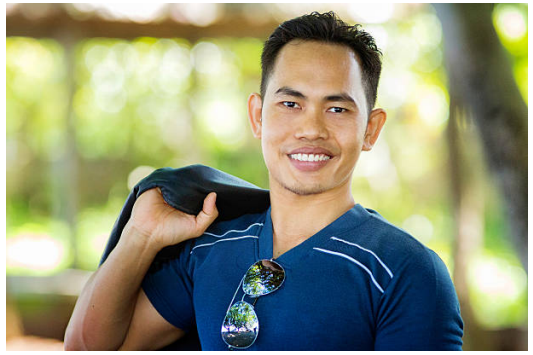
Laundry
The color blue

The color blue

wework

Member Knowledge Graph

How do we calculate how member similarity using common skills or interests?

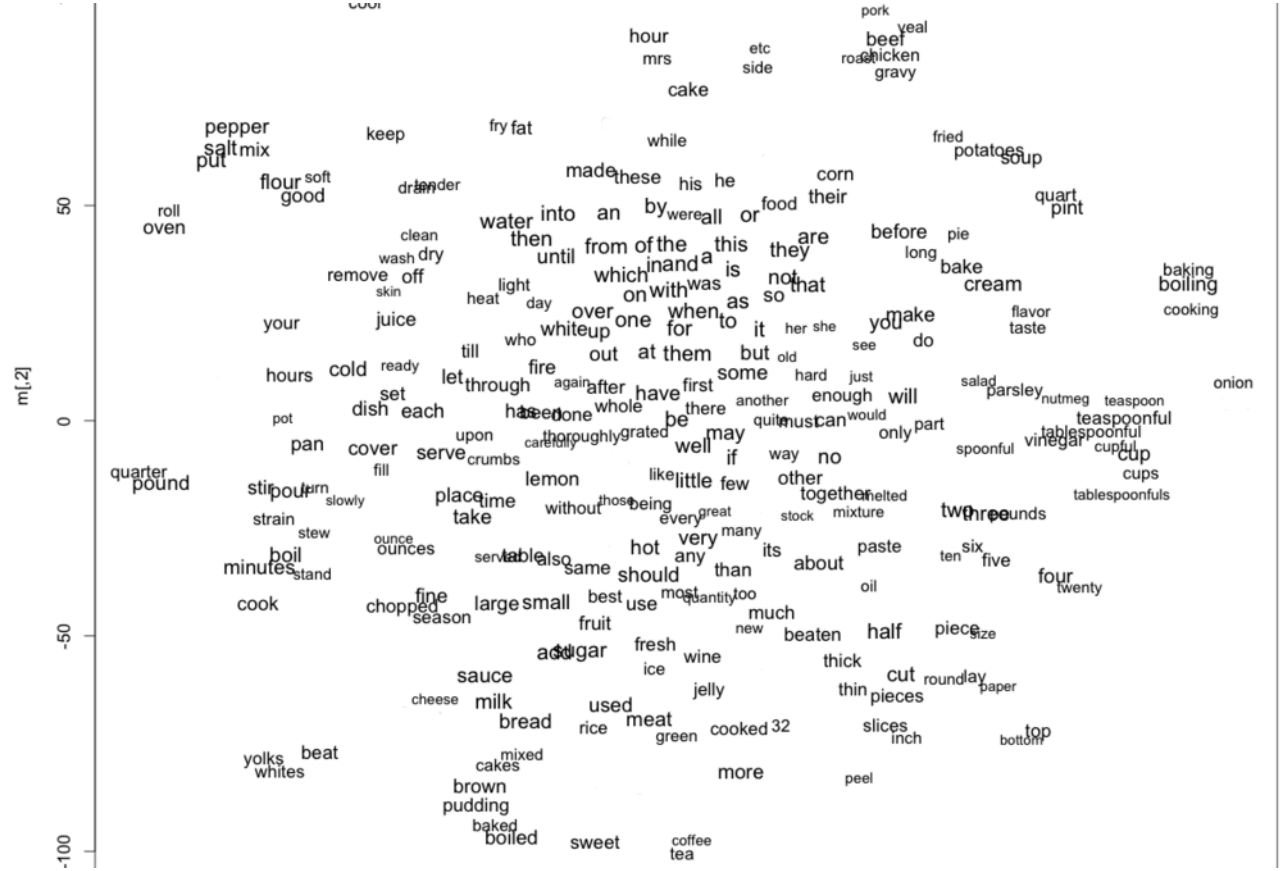


Laundry
The color blue

The color blue

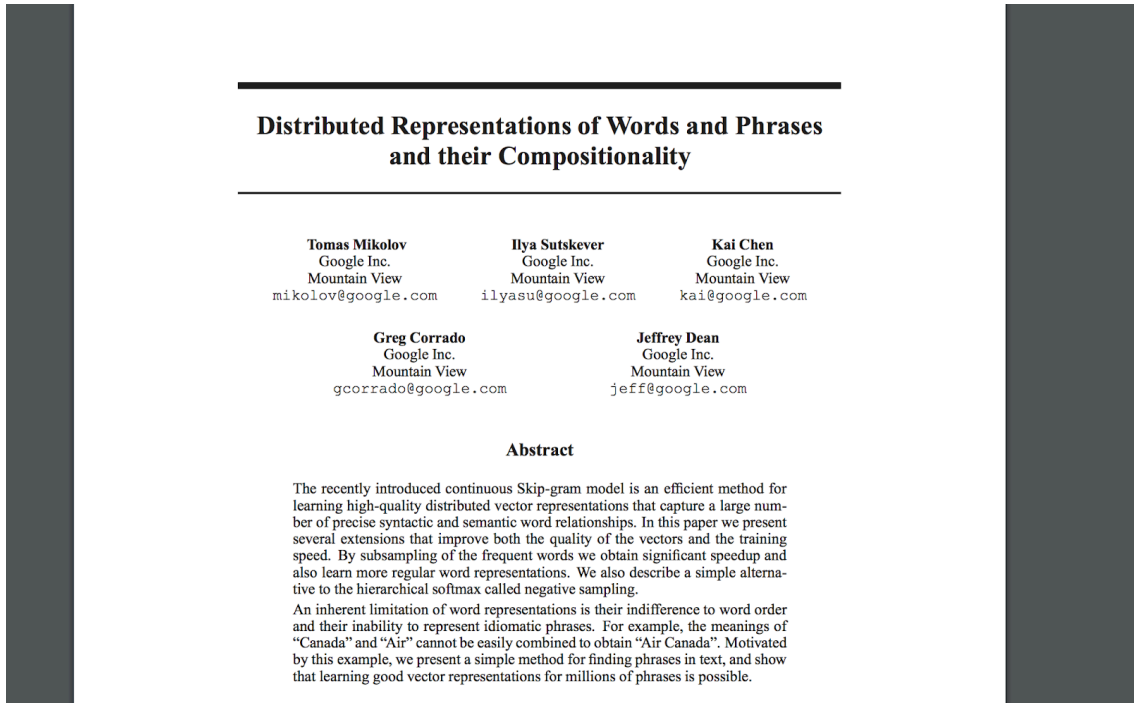
wework

Embeddings



wework

Word Embeddings for Text



Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several extensions that improve both the quality of the vectors and the training speed. By subsampling of the frequent words we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling.

An inherent limitation of word representations is their indifference to word order and their inability to represent idiomatic phrases. For example, the meanings of "Canada" and "Air" cannot be easily combined to obtain "Air Canada". Motivated by this example, we present a simple method for finding phrases in text, and show that learning good vector representations for millions of phrases is possible.

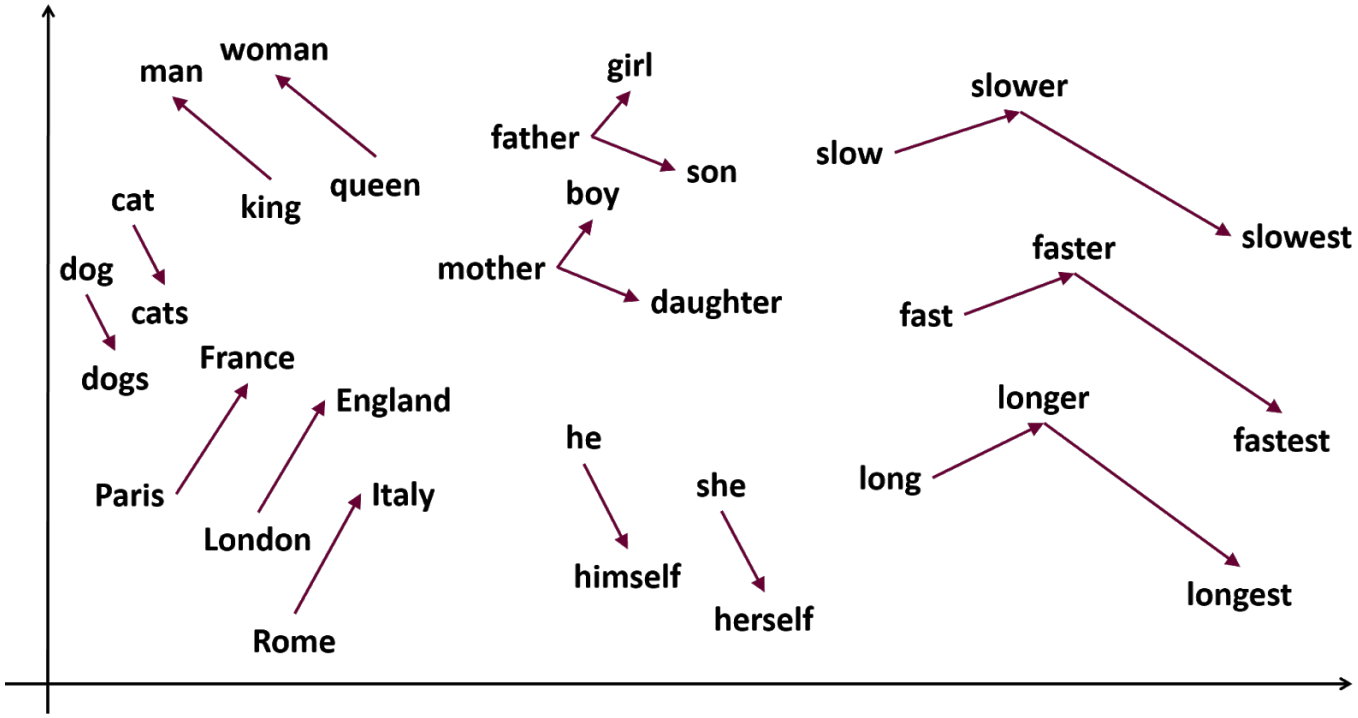
Learn vector representations of words that capture semantic meanings

wework

Word Embeddings for Text

Process: train a neural network

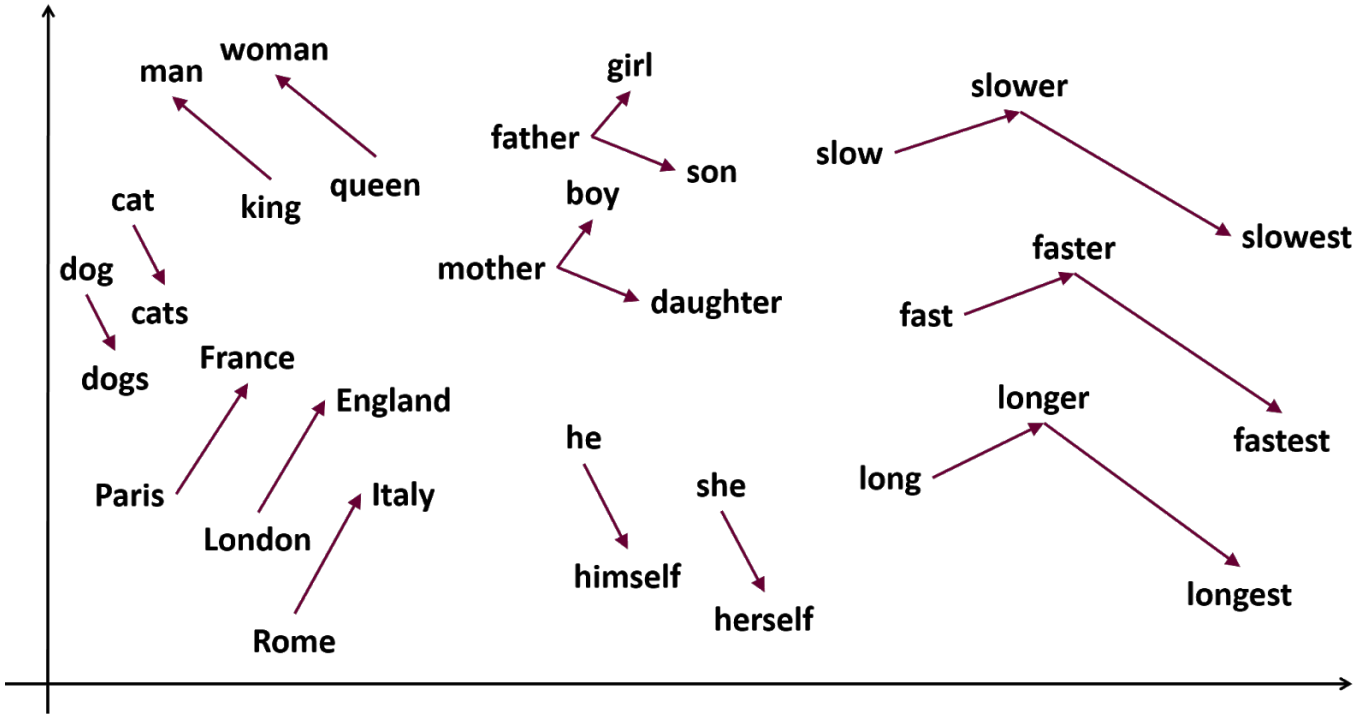
Outcome: each word can be represented by an N-dimensional vector



wework

Word Embeddings for Text

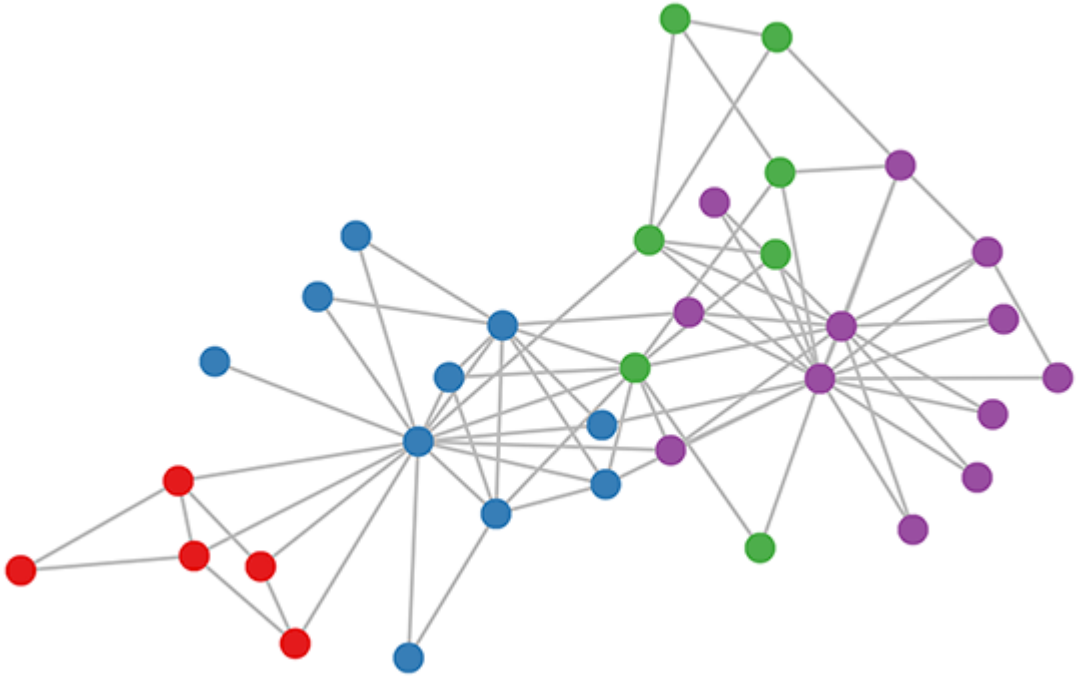
Ultimately the results of this model can be used for many NLP tasks such as word similarity, clustering, classification



wework

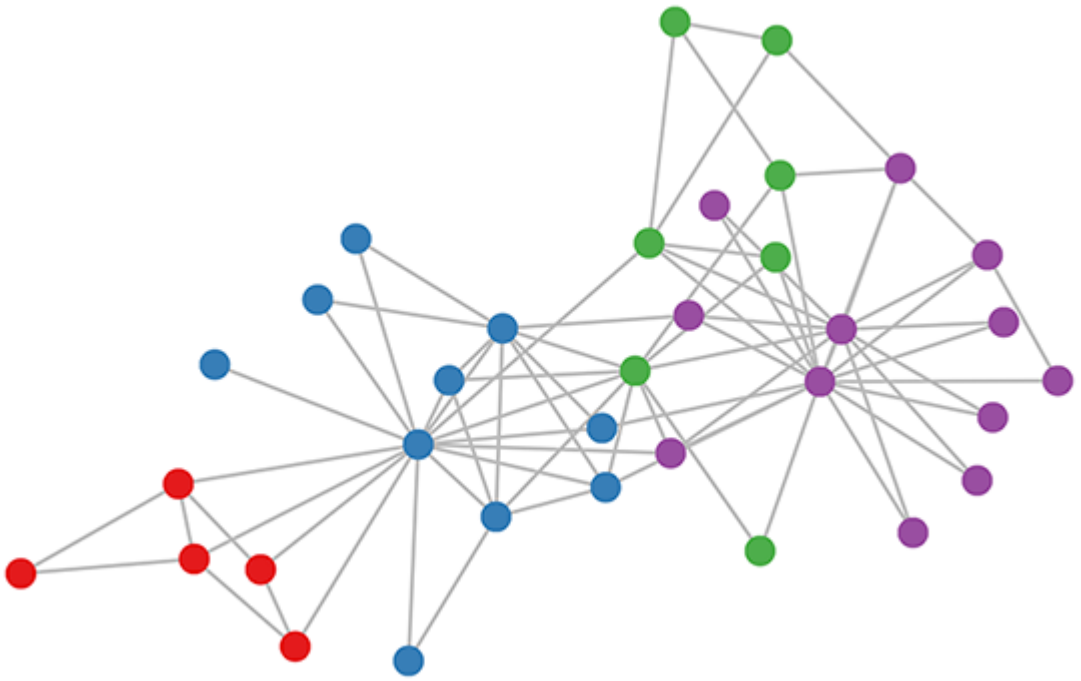
Node Embeddings for Graphs

Can we do something similar for graph networks?



wework

Node Embeddings for Graphs

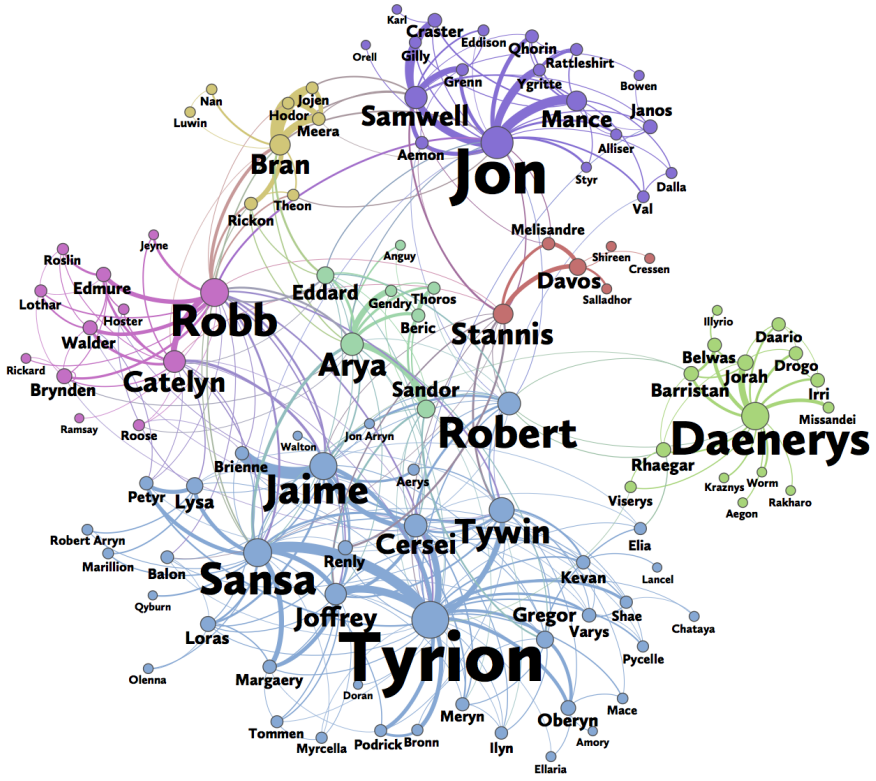


Instead of words we have nodes

Each node can be represented with a vector after training a neural network model

wework

Node Embeddings for Graphs



Node = person

Edge = Social interaction between 2 people

OR

Edge = Similarity between 2 people's skills

OR

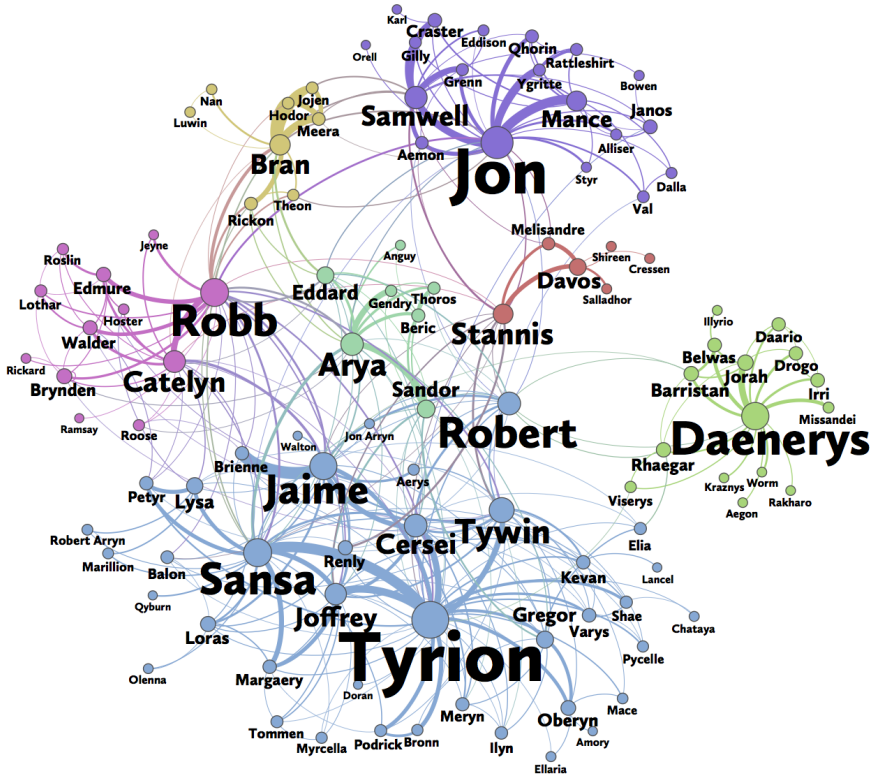
Edge = Similarity between 2 people's interests

OR

Edge = Some other function…

wework

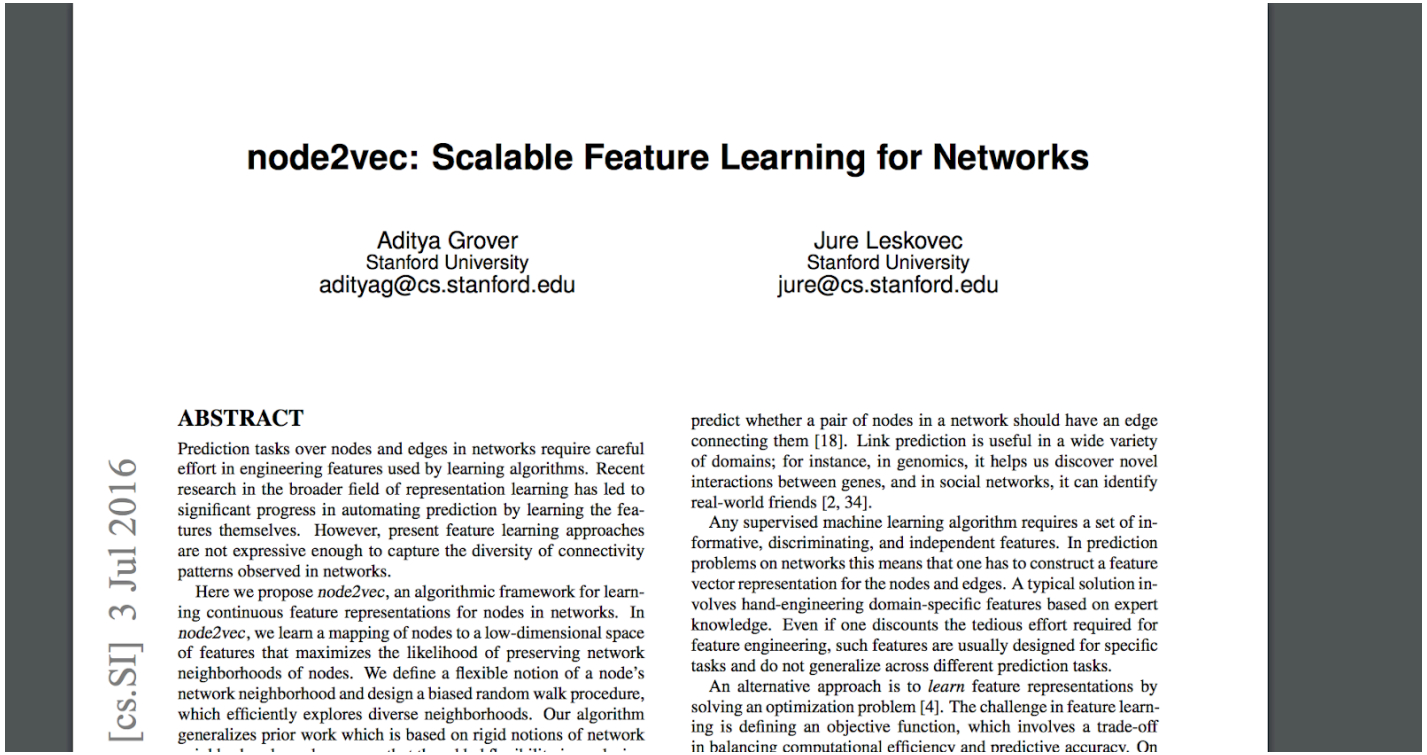
Node Embeddings for Graphs



Nodes with similar vectors (measured by something like cosine similarity) should be clustered close together

wework

Node Embeddings for Graphs



node2vec: Scalable Feature Learning for Networks

Aditya Grover
Stanford University
adityag@cs.stanford.edu

Jure Leskovec
Stanford University
jure@cs.stanford.edu

ABSTRACT

Prediction tasks over nodes and edges in networks require careful effort in engineering features used by learning algorithms. Recent research in the broader field of representation learning has led to significant progress in automating prediction by learning the features themselves. However, present feature learning approaches are not expressive enough to capture the diversity of connectivity patterns observed in networks.

Here we propose *node2vec*, an algorithmic framework for learning continuous feature representations for nodes in networks. In *node2vec*, we learn a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes. We define a flexible notion of a node's network neighborhood and design a biased random walk procedure, which efficiently explores diverse neighborhoods. Our algorithm generalizes prior work which is based on rigid notions of network

predict whether a pair of nodes in a network should have an edge connecting them [18]. Link prediction is useful in a wide variety of domains; for instance, in genomics, it helps us discover novel interactions between genes, and in social networks, it can identify real-world friends [2, 34].

Any supervised machine learning algorithm requires a set of informative, discriminating, and independent features. In prediction problems on networks this means that one has to construct a feature vector representation for the nodes and edges. A typical solution involves hand-engineering domain-specific features based on expert knowledge. Even if one discounts the tedious effort required for feature engineering, such features are usually designed for specific tasks and do not generalize across different prediction tasks.

An alternative approach is to *learn* feature representations by solving an optimization problem [4]. The challenge in feature learning is defining an objective function, which involves a trade-off in balancing computational efficiency and predictive accuracy. On

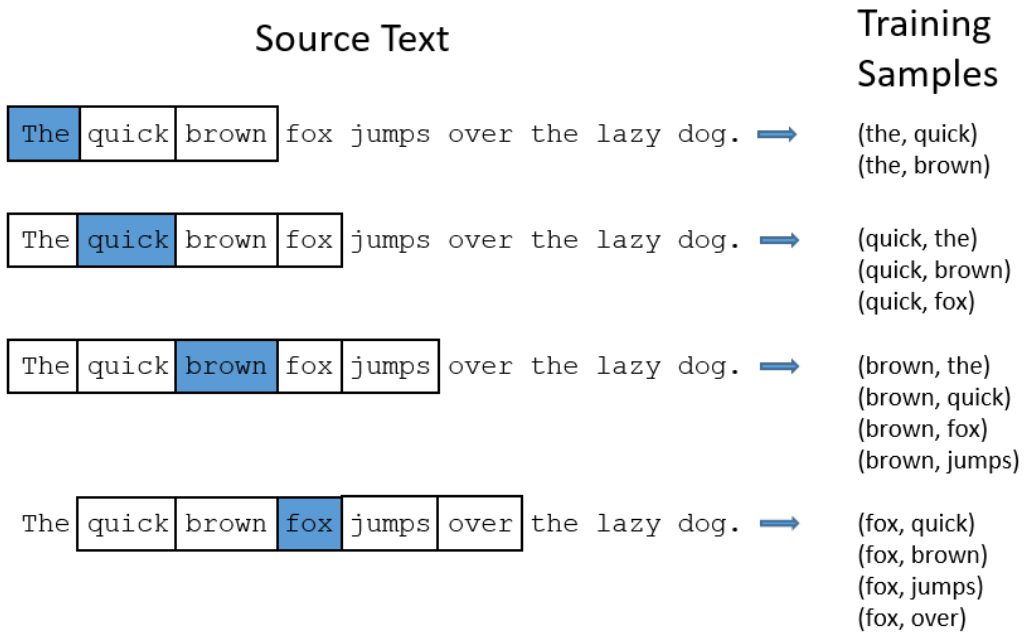
Luckily, someone's already written a paper about this…

wework

word2vec vs Node2vec

word2vec:

Given a corpus, maximize *P(next word | context words)*

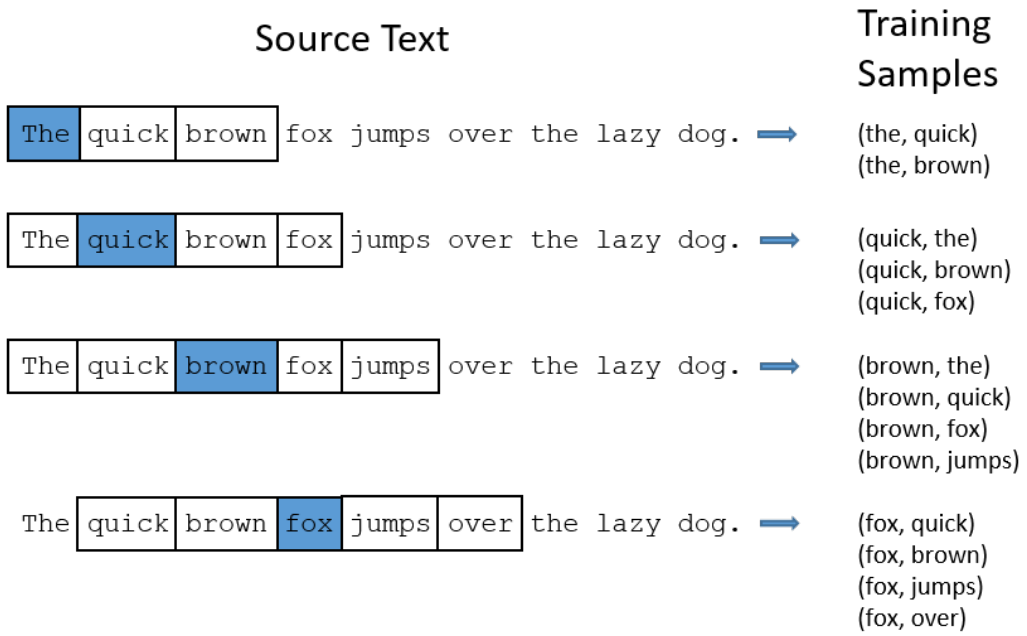


wework

word2vec vs Node2vec

word2vec:

Given a corpus, maximize *P(next word | context words)*



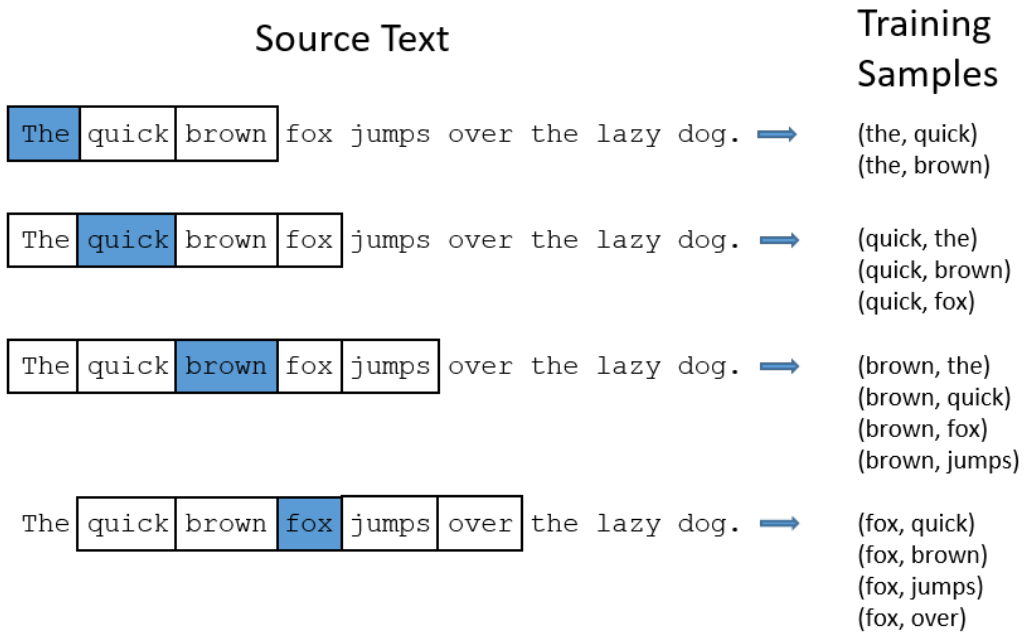
CBOW

wework

word2vec vs Node2vec

word2vec:

Given a corpus, maximize *P(next word | context words)*



CBOW

Skip-gram

wework

word2vec vs Node2vec

node2vec:

For node2vec, we want to maximize the log-probability of observing a network neighborhood $N_s(u)$ for a node u conditioned on its feature representation

$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u)).$$

wework

word2vec vs Node2vec

node2vec:

In order to make this problem tractable, we assume:

- Conditional independence of neighborhood nodes

$$Pr(N_S(u)|f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i|f(u)).$$

wework

word2vec vs Node2vec

node2vec:

In order to make this problem tractable, we assume:

- Symmetry in feature space (a source node and neighborhood node have a symmetric effect over each other)
- Conditional likelihood of every node pair written as a softmax function

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}.$$

word2vec vs Node2vec

node2vec:

With these two assumptions, original function simplifies to:

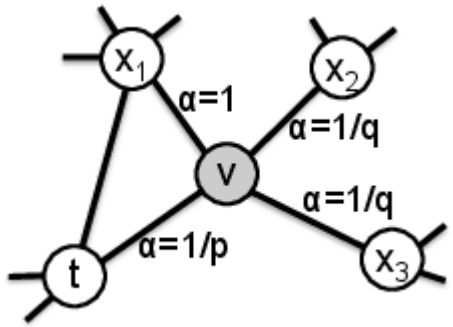
$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u)).$$

$$\max_f \sum_{u \in V} \left[-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right].$$

word2vec vs Node2vec

node2vec:

If we know the representation for node v (the gray node), can we predict its neighborhood (x1, x2, x3)?

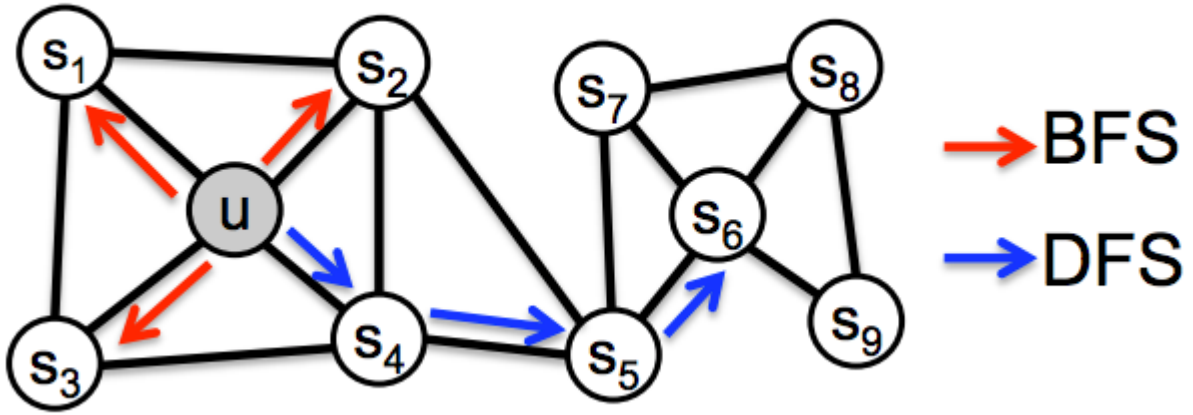


wework

node2vec Algorithm

Problem:

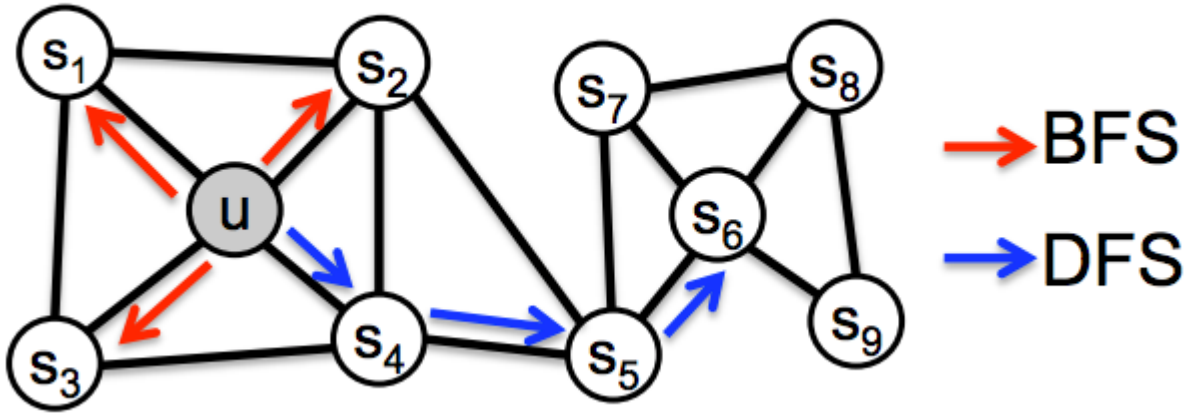
There’s no obvious way to identify separate neighborhoods in a graph like you can identify sentences in a document



node2vec Algorithm

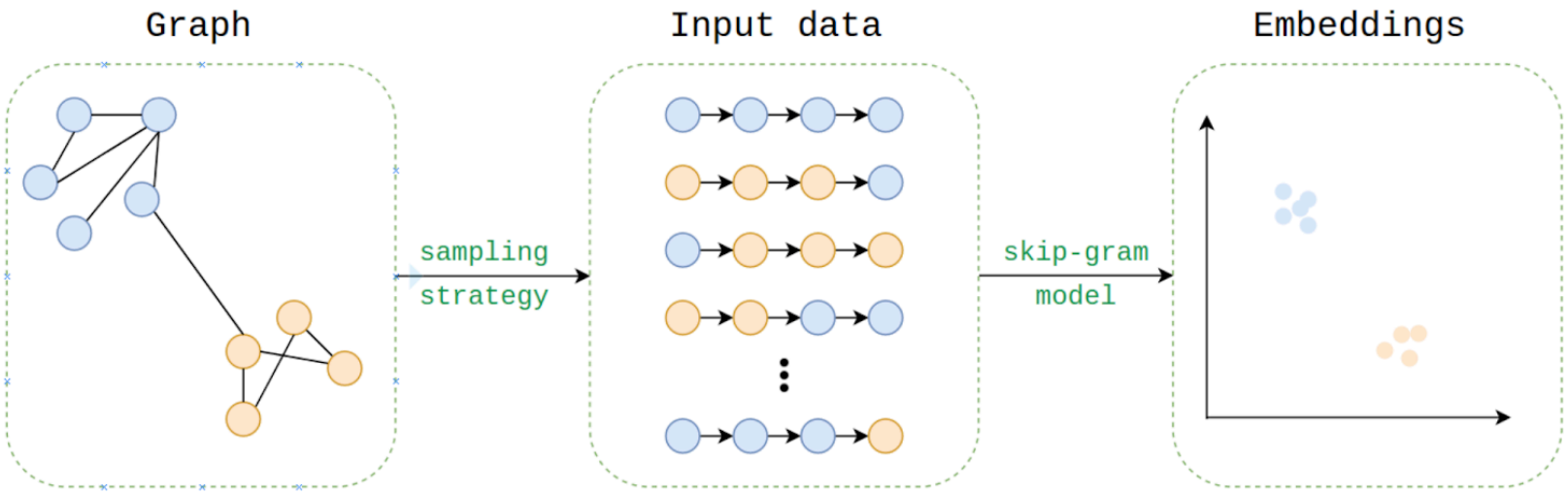
Solution:

Simulate many random walks around each node to define possible “neighborhoods” around the node



node2vec Algorithm

Each walk is a directed subgraph analogous to a “sentence” in a text corpus

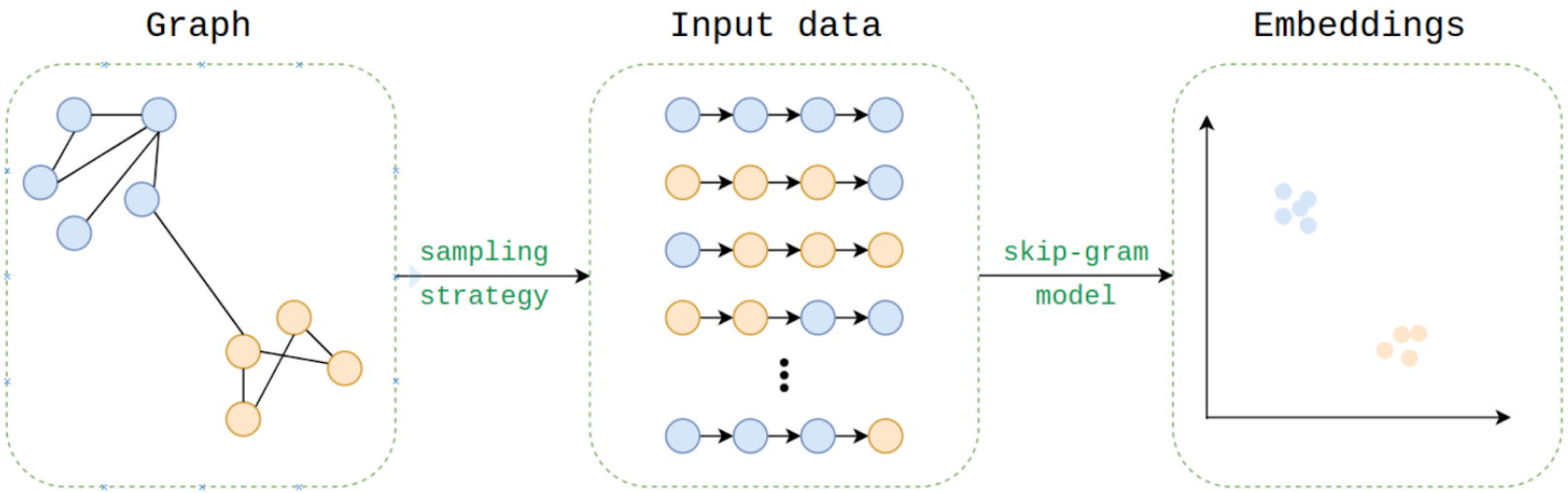


Node2vec embedding process

wework

node2vec Algorithm

From the graph on the left we can create multiple potential neighborhoods and use them as “training data”



Node2vec embedding process

wework

node2vec Algorithm

In order to efficiently explore many possible neighborhoods of node *u,* the random walk algorithm takes two parameters P and Q

Determines how ”locally” or “globally” the walk explores the neighborhood of node *u*

High P → less likely to revisit an already-visited node
High Q → more likely to visit close-by nodes

wework

node2vec Algorithm

Combinations of P, Q can emulate different sampling strategies:

Breadth First Search: Neighborhood is restricted to immediate neighbors of the node *u*

Depth First Search: Neighborhood consists of nodes sampled at increasing distances from *u*

node2vec

The node2vec algorithm can be decomposed into 3 steps:

a) Precompute transition probabilities for the random walk simulation
b) For every node, simulate *r* random walks of fixed length *l*
c) Feed random walks into a word2vec model and solve with SGD

node2vec

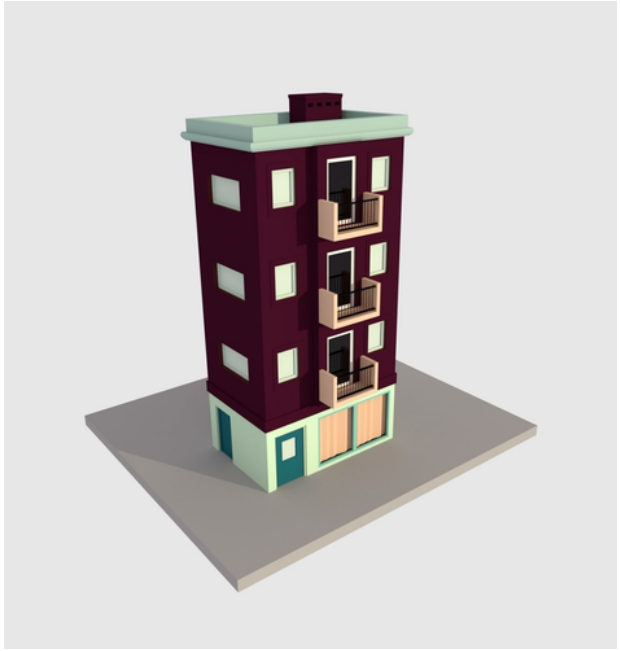
Once we train a model and find embeddings for each node we can do:

- Clustering using node embeddings
- Node classification
- Link prediction
- Community Detection

wework

Clustering off node2vec

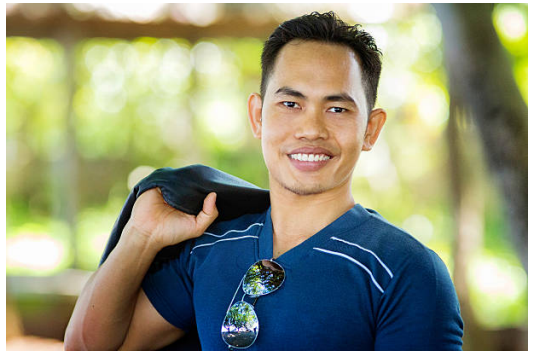
For each WeWork location we can run node2vec on its social graph…



wework

Clustering off node2vec

Using the data in the Member Knowledge Graph…



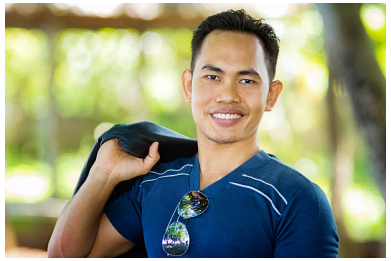
Laundry
The color blue

The color blue

wework

Clustering off node2vec

Map every member to a vector…



```
array([0.4538611 , 0.27743287, 0.25933201, 0.24732495, 0.70126288,
       0.30814868, 0.16309851, 0.979808  , 0.1905782 , 0.23823703])
```

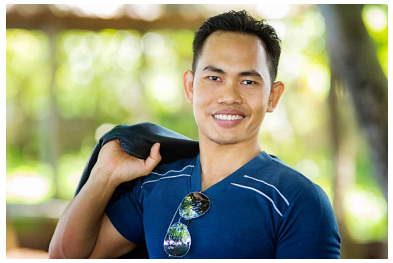
```
array([0.37425164, 0.3933045 , 0.55216442, 0.09399904, 0.00195709,
       0.18054741, 0.91324634, 0.96009897, 0.37971078, 0.98799726])
```

```
array([0.21394969, 0.96883666, 0.44219008, 0.38977323, 0.1161356 ,
       0.30654193, 0.34719876, 0.3183716 , 0.73832435, 0.12602231])
```

wework

Clustering off node2vec

Then retrieve the most similar members for each member…



wework

Clustering off node2vec

And use this to power different kinds of member recommendations

wework

Clustering off node2vec

And use this to power different kinds of member recommendations

-General member recommendations during onboarding

wework

Clustering off node2vec

And use this to power different kinds of member recommendations

-General member recommendations during onboarding

-Facilitated introductions between WeWork Labs members

wework

Thanks for Listening!

wework