node2vec: Embedding of Networks

David McDonald

21st August 2017

Some Context: Why embed networks?

- Visualisation
- feature extraction
- Similar nodes should be embedded close together
- Bijection

Introduction

For our model we are taking inspiration from Natural Language Processing (NLP):

- How can we learn vector representations of words?
- 'similar' words should be mapped close together
- how do can a machine learn these semantics?

Skipgram Model I

- We are going to train a neural network with a single layer to perform a task
- But we are not going to use the neural network for the task it was trained on!
- We will use the weights of the hidden layer – which will be the word representations that we are looking for

Skipgram Model II

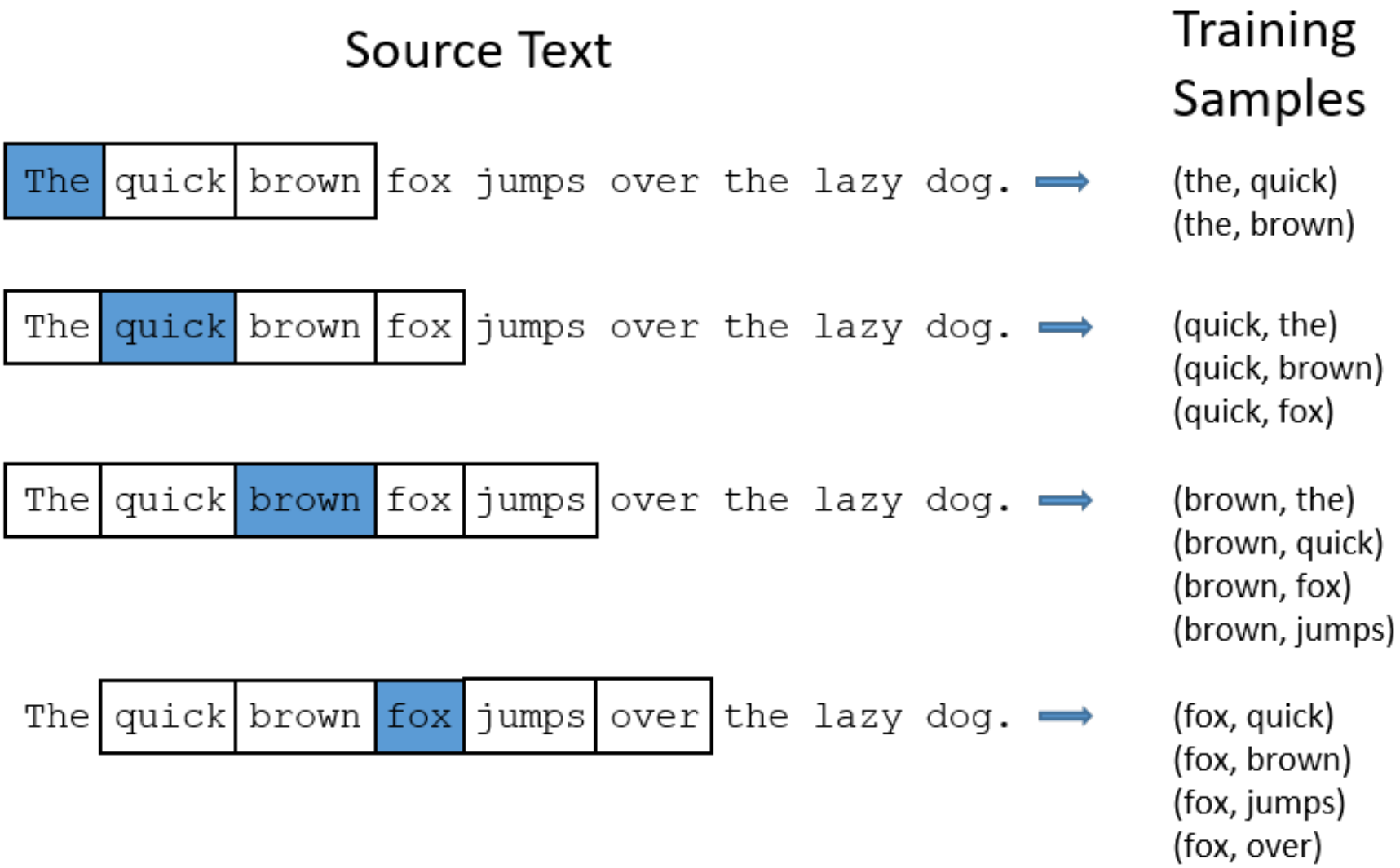
We have seen this technique before
In an autoencoder:

Skipgram Model III

What is the task that we are training on?
Given a sentence an input word, our network will tell us the how likely it is that we see the words in the sentence close to our input word

- Window size

Skipgram Model IV



Training

Our model learns probabilities of the form

$$p(y|x)$$

for example, given (the, quick) as a training sample, we aim to maximise

$$p(\text{quick}|\text{the})$$

and minimise

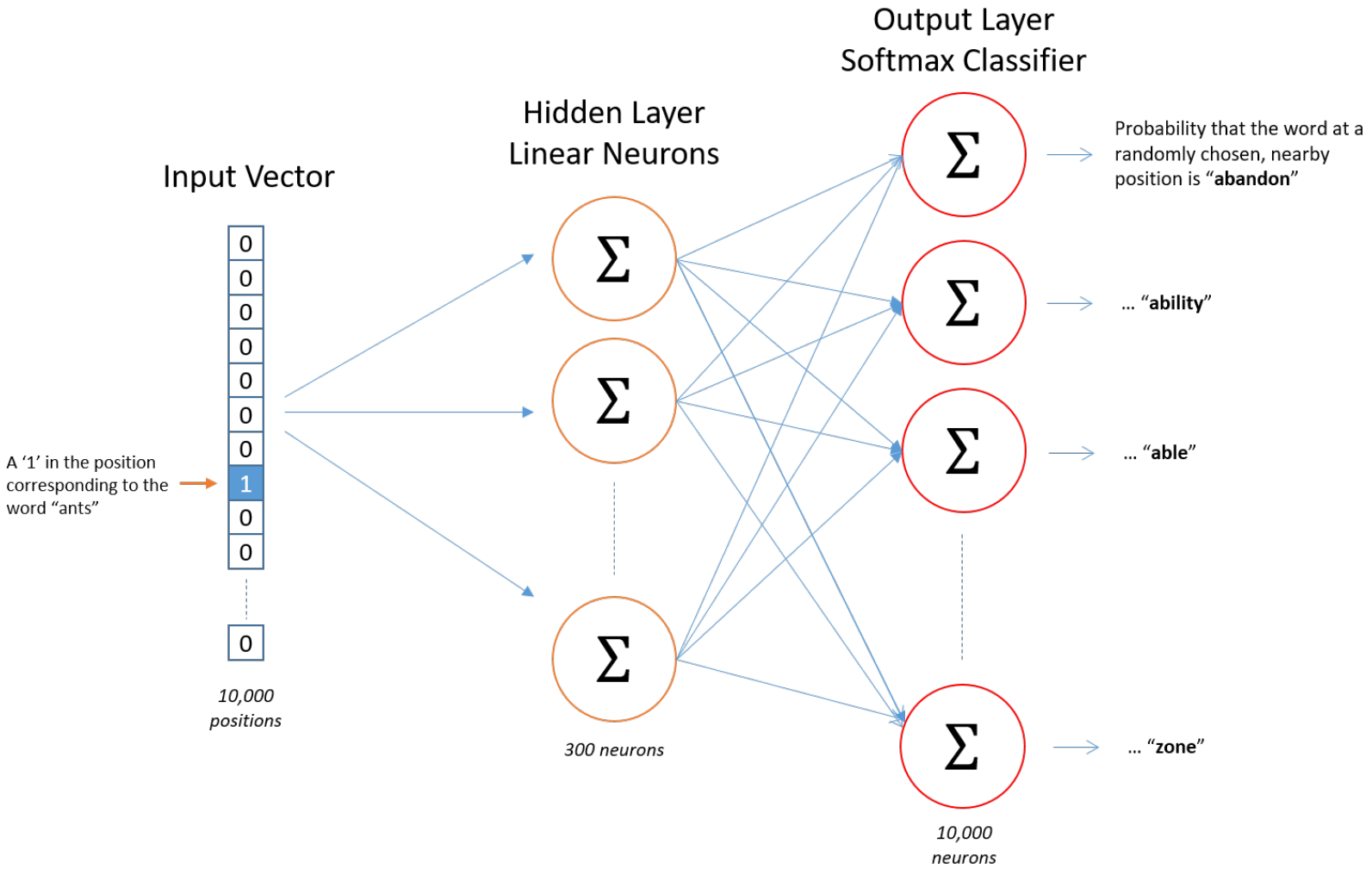
$$p(\text{*anything else*}|\text{the})$$

The Model I

Lets say out vocabulary is 10000 words and we want to learn a 300 dimensional embedding:

- Input: one hot vector of length 10000
- Hidden: 300 linear neurons
- Output: also a vector of length 10000

The Model II


Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"
A '1' in the position corresponding to the word "ants"
0
0
0
0
0
0
0
1
0
0
0
10,000 positions
300 neurons
10,000 neurons

The Hidden Layer I

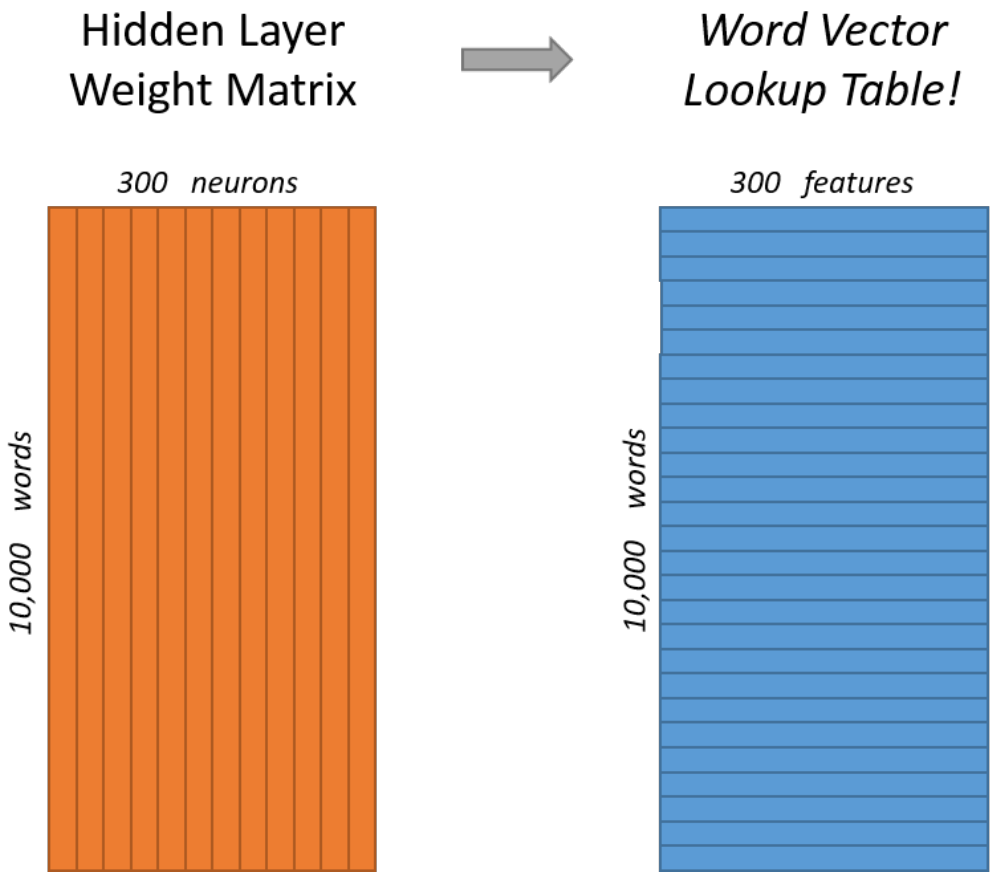
Toy example: vocabulary size of 5, embedding dimension 30

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

The rows of the hidden weight matrix become our word representations!
We just toss the output layer after we have finished training.

The Hidden Layer II

Back to 10000 words and 300 dimensions:



The Output Layer I

Softmax regression:

$$\text{Softmax}(hW^{out}_{,i}) = \frac{\exp(hW^{out}_{,i}))}{\sum_j \exp(hW^{out}_{,j}))}$$

- Each neuron (one for each word in the vocabulary) will output a number between 0 and 1
- All outputs will sum to 1

The columns in W^{out} become 'neighbour' representation of the words. When the hidden representation of our input word (h) is similar to $W_{,i}$ then $hW_{,i}$ is large

Recap

- We train by feeding the model pairs of input/target words determined by 'context window'
- We use one hot representations as input/targets
- We train using Softmax to maximise probability of observing target word, given the input word

Some Problems

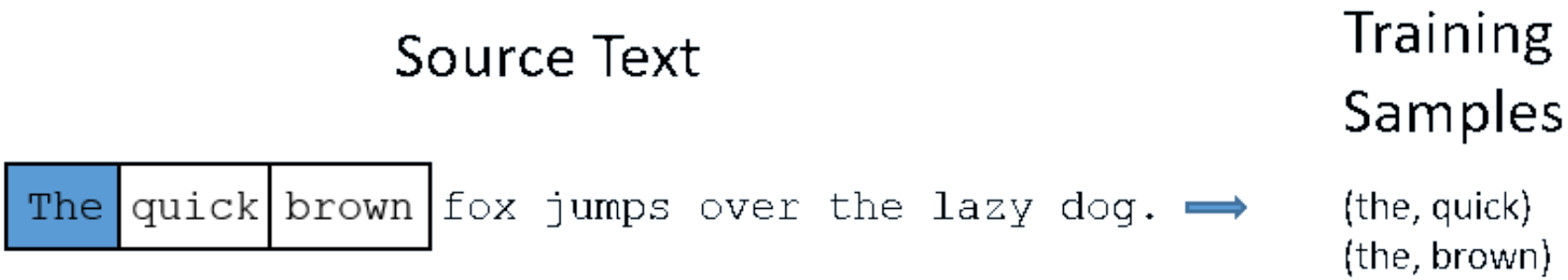
- Our model has a lot of parameters to estimate: $\approx$10000 x 300 + 300 x 10000
- Back propagation with Softmax requires knowledge of all the outputs
- Millions of training examples

All of this leads to very slow training!
3 adaptions to speed up training (and obtain better results!)

Phrases

- scan through text and look for words that frequently appear together
- for example, ‘New’ and ‘York’
- and add these ’phrases’ to the vocabulary

Subsampling

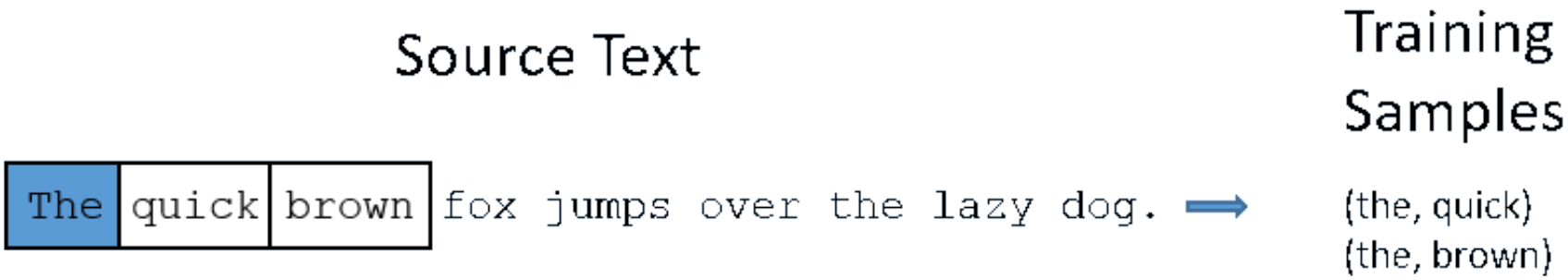


- We would like to cut frequent words from our text
- For example the word ‘the’ does not tell us very much about the word ‘brown’
- the probability of deletion is proportional to frequency
- if we cut a specific instance of the word ‘the’, then it will not appear in the contexts of any nearby words

Negative Sampling I

- Each training sample will adjust **all** the weights in the network a tiny bit
- with negative sampling, we only select a small number of negative samples (say, 5) to adjust the weights for
- negative samples are words that we want the network to output a 0 for
- we also want to update the weights of the positive example

Negative Sampling II



For the training pair (the, quick), we would adjust the weights for

- quick (our positive sample)
- 5 randomly chosen negative samples (that never appear in the context of the input word 'the')

Negative samples are also chosen according to frequency.

node2vec I

So how is this relevant at all to network embedding?

node2vec II

We can use random walks to generate the 'neighbourhood' of a node and use that as its 'context'.

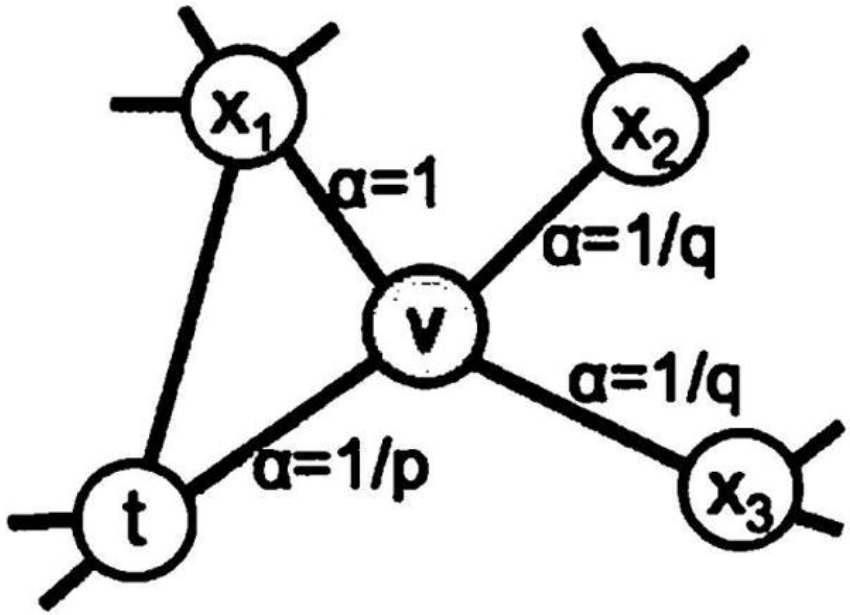
Generating the Context of a Node I

Given a source node u, we define a random walk of length l as a sequence $c_0, ..., c_l$ defined by the following probability distribution:

$$P(c_i = x | c_{i-1} = v, c_{i-2} = t) = \begin{cases} \frac{\alpha_{tx} w_{xv}}{Z} & \text{if } (x, v) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

with

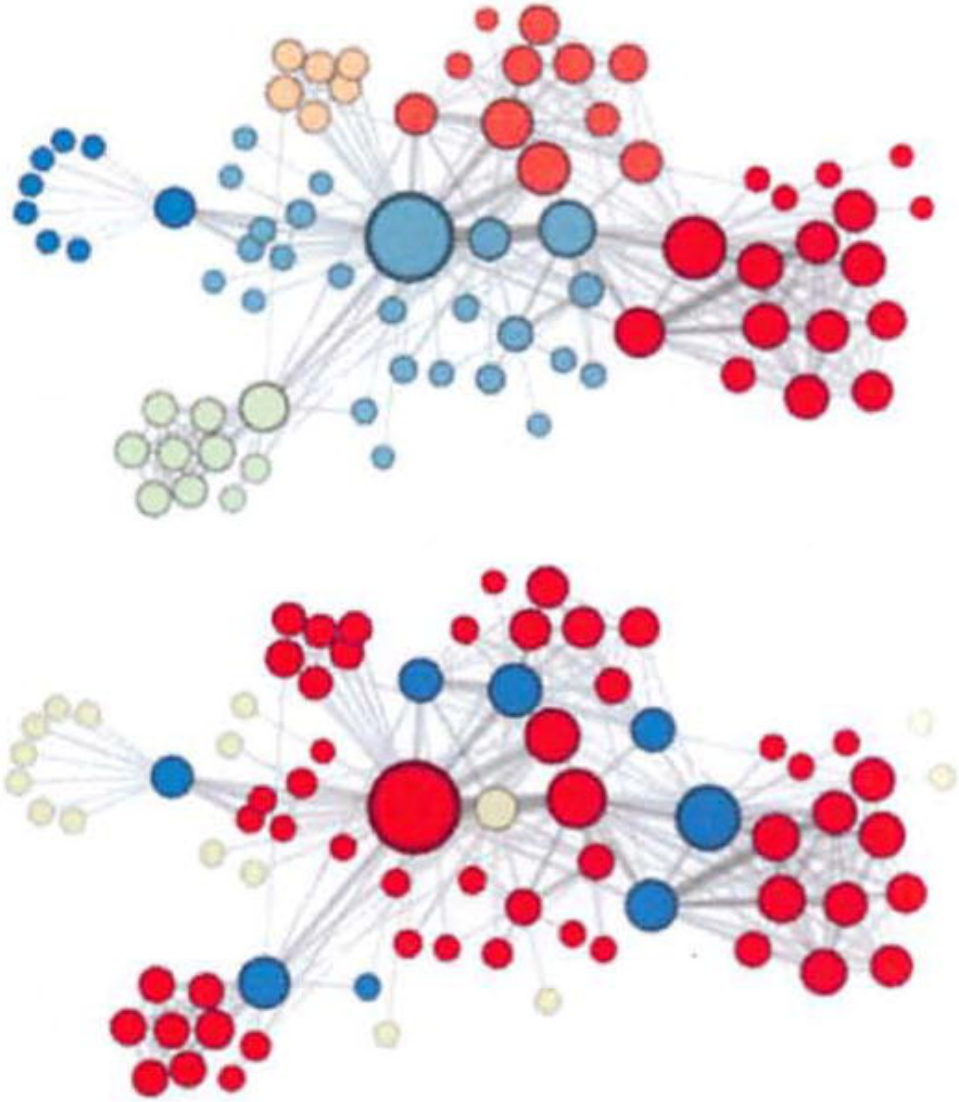
$$\alpha_{tx} = \begin{cases} \frac{1}{p} & \text{if } d(t, x) = 0, \\ 1 & \text{if } d(t, x) = 1, \\ \frac{1}{q} & \text{if } d(t, x) = 2. \end{cases}$$



Generating the Context of a Node II

- l is the context size
- p is the *return parameter*
  - setting it to a high value reduces the chance of backtracking
  - a low setting keeps the search close (local) (BFS)
- q is the *in-out parameter*
  - $q > 1$: biases towards nodes close to t
  - $q < 1$: biases towards nodes further away (DFS)
- Walks are second order Markovian

Example on Les Miserables Network



Extensions

- Multi-layer networks
- Node attributes
- Hyperbolic space

A recommended blog

http://mccormickml.com/2016/04/19/
word2vec-tutorial-the-skip-gram-model/