[Network Embedding]

Skip-Gram

Project : Skip-Gram Implementation with Python

Seunghan Lee (CSE-URP)

20.01.17(Fri)

Goal

"Implement **Skip-Gram** Model using **Random Walk**"

INPUT : (One-Hot Encoded) Vertice

↓

Latent Representation of input vector (Embedded Vector)

OUTPUT : Probability Distribution of Vertices

Contents

1

Introduction

Brief overview of Skip-Gram & Random Walk

2

Implementation

1) Import Dataset & Libraries
2) Define Functions

   (Random Walk, Softmax, Feed Forward, Back Propagation)
3) Skip Gram

3

Result

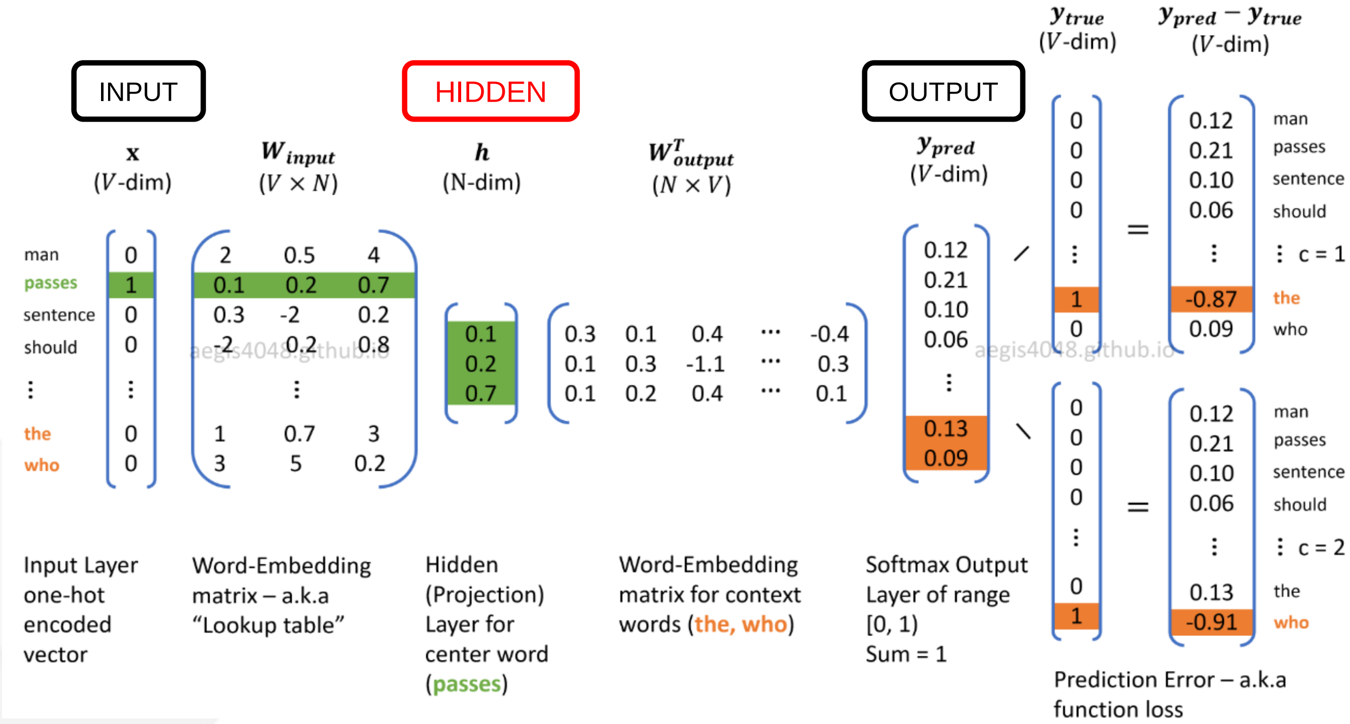
Visualization of Network

1. Introduction

Brief overview of **Skip-Gram** & **Random Walk**

1. Introduction

1. Skip-Gram



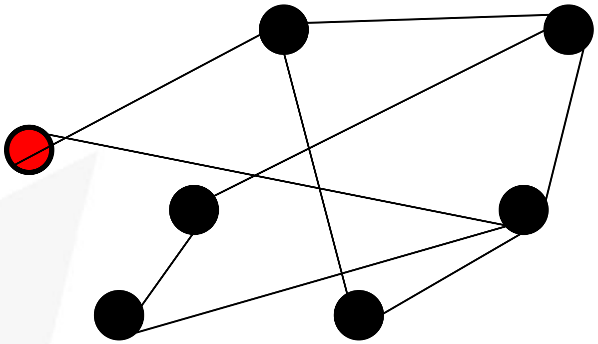
Predict Context Words given One Word

https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

1. Introduction

2. Random Walk

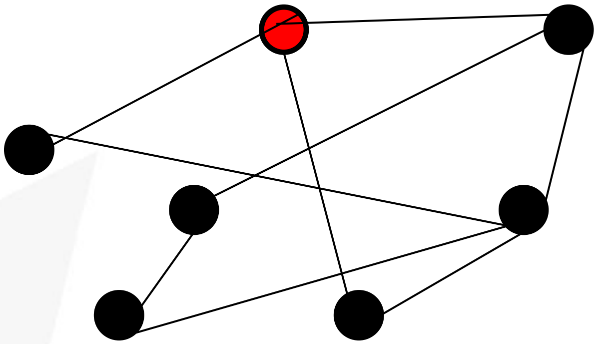
path that consists of a succession of **random** steps (wikipedia)



1. Introduction

2. Random Walk

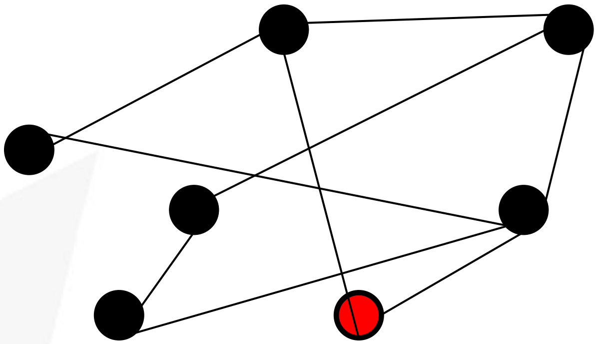
path that consists of a succession of **random** steps (wikipedia)



1. Introduction

2. Random Walk

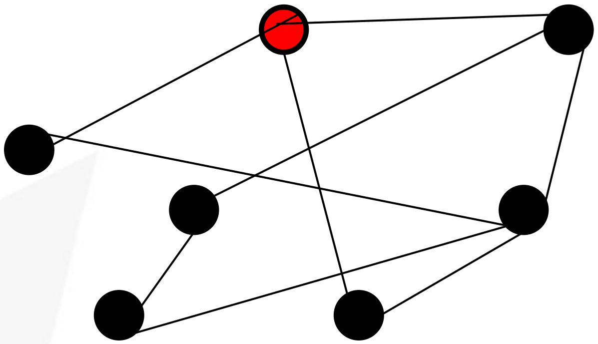
path that consists of a succession of **random** steps (wikipedia)



1. Introduction

2. Random Walk

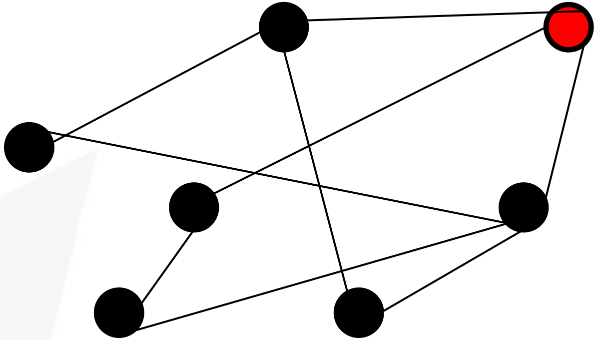
path that consists of a succession of **random** steps (wikipedia)



1. Introduction

2. Random Walk

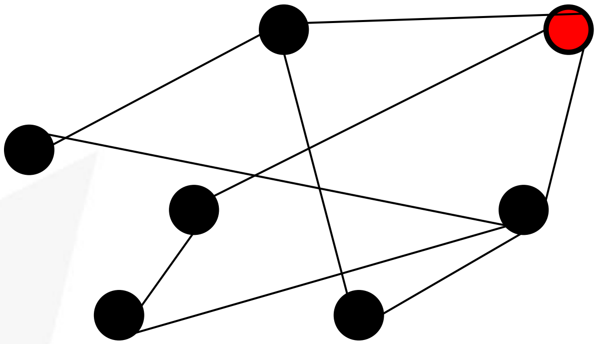
path that consists of a succession of **random** steps (wikipedia)



1. Introduction

2. Random Walk

path that consists of a succession of **random** steps (wikipedia)



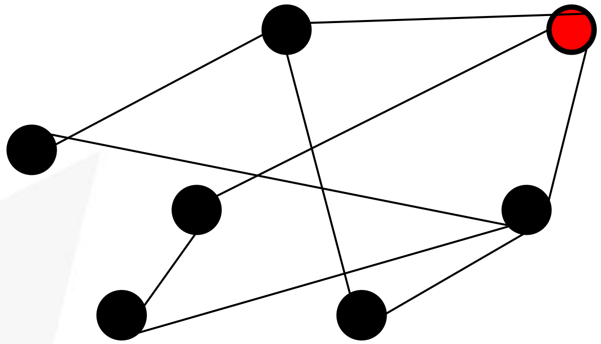
1. Local exploration is easy to parallelize!

2. No need for global recomputation
   (enable online learning)

1. Introduction

2. Random Walk

path that consists of a succession of **random** steps (wikipedia)



1. Local exploration is easy to parallelize!

2. No need for global recomputation
   (enable online learning)

1. Introduction

2. Random Walk

Original

34 Vertices

Ex) walk length = 9

Random Walk

10 Vertices

https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

1. Introduction

2. Random Walk

Original

34 Vertices

Ex) walk length = 9

Random Walk

10 Vertices

⬇

Implement Skip Gram from these 10 vertices!

https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

1. Introduction

2. Random Walk

Original

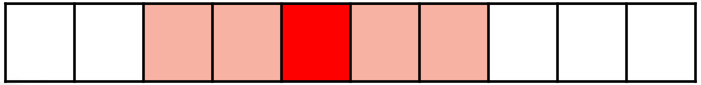
34 Vertices

Ex) walk length = 9

Random Walk

10 Vertices

Implement Skip Gram from these 10 vertices!



(window size = 2)

https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

2. Implementation

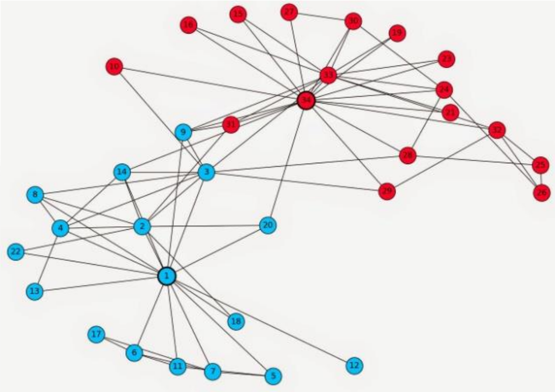
1) Import Dataset & Libraries
2) Define Functions

   (Random Walk, Softmax, Feed Forward, Back Propagation)
3) Skip Gram

Implementation

1. Import Dataset & Libraries

[Data Overview]



Karate Graph

Network Graph with

34 vertices (labeled 0 or 1)

karate_club.adjlist
YongminShin January 13th at 4:43 PM

```
#
# GMT Mon Jan 13 07:41:03 2020
# Zachary's Karate Club
0 1 2 3 4 5 6 7 8 10 11 12 13 17 19 21 31
1 2 3 7 13 17 19 21 30
2 3 7 8 9 13 27 28 32
3 7 12 13
4 6 10
5 6 10 16
6 16
7
8 30 32 33
9 33
10
11
12
13 33
14 32 33
15 32 33
16
17
18 32 33
19 33
20 32 33
21
22 32 33
23 25 27 29 32 33
24 25 27 31
25 31
26 29 33
27 33
28 31 33
29 32 33
30 32 33
31 32 33
32 33
```

[1. adjacency list]

karate_club.edgelist
YongminShin January 13th at 4:43 PM

```
0 1 {}
0 2 {}
0 3 {}
0 4 {}
0 5 {}
0 6 {}
0 7 {}
0 8 {}
0 10 {}
0 11 {}
0 12 {}
0 13 {}
0 17 {}
0 19 {}
0 21 {}
0 31 {}
1 2 {}
1 3 {}
1 7 {}
1 13 {}
1 17 {}
1 19 {}
1 21 {}
1 30 {}
2 3 {}
2 7 {}
2 8 {}
2 9 {}
2 13 {}
2 27 {}
2 28 {}
2 32 {}
3 7 {}
3 12 {}
3 13 {}
4 6 {}
```

[2. edge list]

https://bookdown.org/omarlizardo/_main/3-1-social-network-analysis-from-relationships-to-graphs.html

Implementation

1. Import Dataset & Libraries

1. Import Dataset

In [1]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import random
import pandas as pd
from random import shuffle
from copy import copy

%matplotlib inline
```

In [2]:
```
edge = pd.read_csv('karate_club.edgelist', sep=' ', names=['x','y','w'])
```

In [3]:
```
edge.head()
```

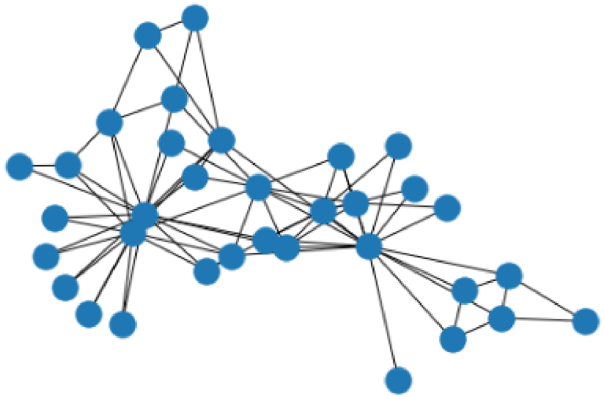
Out[3]:

	x	y	w
0	0	1	{}
1	0	2	{}
2	0	3	{}
3	0	4	{}
4	0	5	{}

In [3]:
```
graph = nx.Graph()
for i in range(edge.shape[0]):
    graph.add_node(node_for_adding = edge['x'][i])
    graph.add_node(node_for_adding = edge['y'][i])
    graph.add_edge(edge['x'][i], edge['y'][i])
```

In [4]:
```
nx.draw(graph,with_label=True)
```

```
C:#Users#samsung#Anaconda3#lib#site-packages#networkx#drawi
The iterable function was deprecated in Matplotlib 3.1 and
  if not cb.iterable(width):
```



Implementation

1. Import Dataset & Libraries

1) Adjacency Matrix

In [5]:
```
A = nx.to_numpy_matrix(graph, nodelist=sorted(graph.nodes()))
```

In [6]:
```
A
```

Out[6]:
```
matrix([[0., 1., 1., ..., 1., 0., 0.],
        [1., 0., 1., ..., 0., 0., 0.],
        [1., 1., 0., ..., 0., 1., 0.],
        ...,
        [1., 0., 0., ..., 0., 1., 1.],
        [0., 0., 1., ..., 1., 0., 1.],
        [0., 0., 0., ..., 1., 1., 0.]])
```

2). Input Word Vector (One-Hot encoded)

In [7]:
```
OH = np.identity(34)
```

In [8]:
```
OH
```

Out[8]:
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

Implementation

1. Import Dataset & Libraries

1) Adjacency Matrix

In [5]:
```
A = nx.to_numpy_matrix(graph, nodelist=sorted(graph.nodes()))
```

In [6]:
```
A
```

Out[6]:
```
matrix([[0., 1., 1., ..., 1., 0., 0.],
        [1., 0., 1., ..., 0., 0., 0.],
        [1., 1., 0., ..., 0., 1., 0.],
        ...,
        [1., 0., 0., ..., 0., 1., 1.],
        [0., 0., 1., ..., 1., 0., 1.],
        [0., 0., 0., ..., 1., 1., 0.]])
```

1 in adjacent vertices,

0 otherwise

2). Input Word Vector (One-Hot encoded)

In [7]:
```
OH = np.identity(34)
```

In [8]:
```
OH
```

Out[8]:
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

Shape : 34 x 34

Implementation

1. Import Dataset & Libraries

1) Adjacency Matrix

In [5]:
```
A = nx.to_numpy_matrix(graph, nodelist=sorted(graph.nodes()))
```

In [6]:
```
A
```

Out[6]:
```
matrix([[0., 1., 1., ..., 1., 0., 0.],
        [1., 0., 1., ..., 0., 0., 0.],
        [1., 1., 0., ..., 0., 1., 0.],
        ...,
        [1., 0., 0., ..., 0., 1., 1.],
        [0., 0., 1., ..., 1., 0., 1.],
        [0., 0., 0., ..., 1., 1., 0.]])
```

1 in adjacent vertices,

0 otherwise

for Random Walk!

- each row : one vertex
- By finding the index of **NON-ZERO** values

2). Input Word Vector (One-Hot encoded)

In [7]:
```
OH = np.identity(34)
```

In [8]:
```
OH
```

Out[8]:
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

Shape : 34 x 34

for Input Vector of every vertex

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

1). Random Walk

In [9]:
```python
def random_step(i,w):
    walk_list = []
    walk_list.append(i)
    for k in range(w):
        ad = np.nonzero(A[i])[1] # i와 인접한 vertex들의 list
        rand = random.choice(ad) # 그 list중 랜덤하게 하나 고르기
        walk_list.append(rand)
        i = rand
    return walk_list
```

In [78]:
```python
random_step(3,10)
```

Out [78]:
```
[3, 2, 1, 21, 0, 21, 1, 0, 21, 1, 19]
```

2) softmax

In [93]:
```python
def softmax(x):
    c = np.max(x)
    b = x-c
    exp_x = np.exp(b)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y
```

https://medium.com/@jonathan_hui/machine-learning-summary-algorithm-d75c64963800

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

1). Random Walk

In [9]:

```python
def random_step(i,w):
    walk_list = []
    walk_list.append(i)
    for k in range(w):
        ad = np.nonzero(A[i])[1] # i와 인접한 vertex들의 list
        rand = random.choice(ad) # 그 list중 랜덤하게 하나 고르기
        walk_list.append(rand)
        i = rand
    return walk_list
```

In [78]:

```python
random_step(3,10)
```

Out [78]: [3, 2, 1, 21, 0, 21, 1, 0, 21, 1, 19]

2) softmax

In [93]:

```python
def softmax(x):
    c = np.max(x)
    b = x-c
    exp_x = np.exp(b)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y
```

Row 0	0	1	1	0	0	…	1	1
Row 1	1	0	0	1	0	…	1	0
…	…						1	…
Row 32	1	1	0	0	1	1	0	0
Row 33	1	1	1	0	0	…	0	0

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

1). Random Walk

In [9]:
```
def random_step(i,w):
    walk_list = []
    walk_list.append(i)
    for k in range(w):
        ad = np.nonzero(A[i])[1] # i와 인접한 vertex들의 list
        rand = random.cho ce(ad) # 그 list중 랜덤하게 하나 고르기
        walk_list.append(rand)
        i = rand
    return walk_list
```

In [78]:
```
random_step(3,10)
```

Out [78]: [3, 2, 1, 21, 0, 21, 1, 0, 21, 1, 19]

2) softmax

In [93]:
```
def softmax(x):
    c = np.max(x)
    b = x-c
    exp_x = np.exp(b)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y
```

Row 0	0	1	1	0	0	…	1	1
Row 1	1	0	0	1	0	…	**1**	0
…	…						1	…
Row 32	1	1	0	0	1	1	0	0
Row 33	1	1	1	0	0	…	0	0

(input) 1 - 32

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

1). Random Walk

In [9]:
```python
def random_step(i,w):
    walk_list = []
    walk_list.append(i)
    for k in range(w):
        ad = np.nonzero(A[i])[1] # i와 인접한 vertex들의 list
        rand = random.choice(ad) # 그 list중 랜덤하게 하나 고르기
        walk_list.append(rand)
        i = rand
    return walk_list
```

In [78]:
```python
random_step(3,10)
```

Out [78]: [3, 2, 1, 21, 0, 21, 1, 0, 21, 1, 19]

2) softmax

In [93]:
```python
def softmax(x):
    c = np.max(x)
    b = x-c
    exp_x = np.exp(b)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y
```

Row 0	0	1	1	0	0	…	1	1
Row 1	1	0	0	1	0	…	1	0
…	…						1	…
Row 32	1	1	0	0	1	1	0	0
Row 33	1	1	1	0	0	…	0	0

Row 0	0	1	1	0	0	…	1	1
Row 1	1	0	0	1	0	…	1	0
…	…						1	…
Row 32	**1**	1	0	0	1	1	0	0
Row 33	1	1	1	0	0	…	0	0

(input) 1 - 32 - 0

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

1). Random Walk

In [9]:

```
def random_step(i,w):
    walk_list = []
    walk_list.append(i)
    for k in range(w):
        ad = np.nonzero(A[i])[1] # i와 인접한 vertex들의 list
        rand = random.choice(ad) # 그 list중 랜덤하게 하나 고르기
        walk_list.append(rand)
        i = rand
    return walk_list
```

In [78]:

```
random_step(3,10)
```

Out [78]: [3, 2, 1, 21, 0, 21, 1, 0, 21, 1, 19]

2) softmax

In [93]:

```
def softmax(x):
    c = np.max(x)
    b = x-c
    exp_x = np.exp(b)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y
```

Row 0	0	1	1	0	0	…	1	1
Row 1	1	0	0	1	0	…	1	0
…	…						1	…
Row 32	1	1	0	0	1	1	0	0
Row 33	1	1	1	0	0	…	0	0

Row 0	0	1	1	0	0	…	1	1
Row 1	1	0	0	1	0	…	1	0
…	…						1	…
Row 32	1	1	0	0	1	1	0	0
Row 33	1	1	1	0	0	…	0	0

Row 0	0	1	**1**	0	0	…	1	1
Row 1	1	0	0	1	0	…	1	0
…	…						1	…
Row 32	1	1	0	0	1	1	0	0
Row 33	1	1	1	0	0	…	0	0

(input) 1 - 32 - 0 - 2

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

1). Random Walk

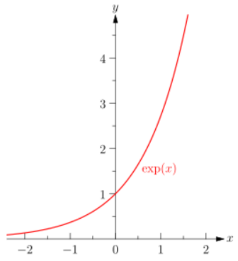
In [9]:
```python
def random_step(i,w):
    walk_list = []
    walk_list.append(i)
    for k in range(w):
        ad = np.nonzero(A[i])[1] # i와 인접한 vertex들의 list
        rand = random.choice(ad) # 그 list중 랜덤하게 하나 고르기
        walk_list.append(rand)
        i = rand
    return walk_list
```

In [78]:
```python
random_step(3,10)
```

Out [78]: [3, 2, 1, 21, 0, 21, 1, 0, 21, 1, 19]

2) softmax

In [93]:
```python
def softmax(x):
    c = np.max(x)
    b = x-c
    exp_x = np.exp(b)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y
```



Exponential Function

The problem arise when $x(i)$ is too small or too large. Suppose each $x(i)$ is very small negative number, $\exp(x(i))$ will be close to 0, since all the $x(i)$ are very small the denominator of softmax function will be close to 0 and result will be not defined. This is called **underflow**. If $x(i)$ is very large $\exp(x(i))$ will be very large number, may exceed the computational limit. This is called **overflow.**

$$softmax(x)_i = \frac{\overbrace{e^{x_i}}^{\text{score}}}{\sum_{j=1}^{n} e^{x_j}} = \frac{e^{x_i'}}{\sum e^{x_j'}}$$

Avoid overflow or underflow

$$m = \max(x)$$

$$x_j \rightarrow x_j - m = x_j'$$

https://medium.com/@jonathan_hui/machine-learning-summary-algorithm-d75c64963800
https://medium.com/@ravish1729/analysis-of-softmax-function-ad058d6a564d

Implementation

2. Define Functions
(Random Walk, Softmax, Feed Forward, Back Propagation)

1). Random Walk

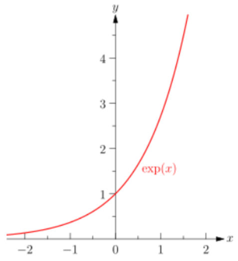
In [9]:
```python
def random_step(i,w):
    walk_list = []
    walk_list.append(i)
    for k in range(w):
        ad = np.nonzero(A[i])[1] # i와 인접한 vertex들의 list
        rand = random.choice(ad) # 그 list중 랜덤하게 하나 고르기
        walk_list.append(rand)
        i = rand
    return walk_list
```

In [78]:
```python
random_step(3,10)
```

Out [78]: `[3, 2, 1, 21, 0, 21, 1, 0, 21, 1, 19]`

2) softmax

In [93]:
```python
def softmax(x):
    c = np.max(x)
    b = x-c
    exp_x = np.exp(b)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y
```



Exponential Function

The problem arise when $x(i)$ is too small or too large. Suppose each $x(i)$ is very small negative number, $\exp(x(i))$ will be close to 0, since all the $x(i)$ are very small the denominator of softmax function will be close to 0 and result will be not defined. This is called **underflow**. If $x(i)$ is very large $\exp(x(i))$ will be very large number, may exceed the computational limit. This is called **overflow.**

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} = \frac{e^{x_i'}}{\sum e^{x_j'}}$$

score

Avoid overflow or underflow
$m = \max(x)$
$x_j \rightarrow x_j - m = x_j'$

https://medium.com/@jonathan_hui/machine-learning-summary-algorithm-d75c64963800
https://medium.com/@ravish1729/analysis-of-softmax-function-ad058d6a564d

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

3) Feed Forward

In [94]:

```
def feedforward(input_word,index,w1,w2):
    h=np.matmul(w1.T,input_word[index])
    u=np.matmul(w2.T,h)
    y = softmax(u)
    return h,u,y
```

4) Back Propagation

In [95]:

```
def backprop(input_word,w1,w2,lr,h,y_pred,index,window_size):
    front = input_word[index-window_size : index]
    back = input_word[index+1 : index+window_size+1]
    window_OH = np.concatenate([front,back])

    # output -> hidden
    for j in range(w2.shape[1]):
        adjust = (y_pred-window_OH)[:,j].sum()*h
        w2[:,j] -= -lr*adjust

    # hidden -> input
    adjust2 = ((y_pred-window_OH).sum(axis=0)*w2).T
    w1-= lr*adjust2
    return w1,w2
```

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

3) Feed Forward

In [94]:

```python
def feedforward(input_word,index,w1,w2):
    h=np.matmul(w1.T,input_word[index])
    u=np.matmul(w2.T,h)
    y = softmax(u)
    return h,u,y
```

4) Back Propagation

In [95]:

```python
def backprop(input_word,w1,w2,lr,h,y_pred,index,window_size):
    front = input_word[index-window_size : index]
    back = input_word[index+1 : index+window_size+1]
    window_OH = np.concatenate([front,back])

    # output -> hidden
    for j in range(w2.shape[1]):
        adjust = (y_pred-window_OH)[:,j].sum()*h
        w2[:,j] -= -lr*adjust

    # hidden -> input
    adjust2 = ((y_pred-window_OH).sum(axis=0)*w2).T
    w1-= lr*adjust2
    return w1,w2
```

$$\mathbf{h} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_I}$$

$$u_{c,j} = u_j = \mathbf{v}'_{w_j}{}^T \cdot \mathbf{h}$$

$$y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

3) Feed Forward

In [94]:

```
def feedforward(input_word,index,w1,w2):
    h=np.matmul(w1.T,input_word[index])
    u=np.matmul(w2.T,h)
    y = softmax(u)
    return h,u,y
```

4) Back Propagation

In [95]:

```
def backprop(input_word,w1,w2,lr,h,y_pred,index,window_size):
    front = input_word[index-window_size : index]
    back = input_word[index+1 : index+window_size+1]
    window_OH = np.concatenate([front,back])

    # output -> hidden
    for j in range(w2.shape[1]):
        adjust = (y_pred-window_OH)[:,j].sum()*h
        w2[:,j] -= -lr*adjust

    # hidden -> input
    adjust2 = ((y_pred-window_OH).sum(axis=0)*w2).T
    w1-= lr*adjust2
    return w1,w2
```

$$\mathbf{h} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_I}$$

Calculate the hidden layer

(W : input -> hidden weight)

(k : index of input word)

$$u_{c,j} = u_j = \mathbf{v'}_{w_j}{}^T \cdot \mathbf{h}$$

$$y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

3) Feed Forward

In [94]:
```
def feedforward(input_word,index,w1,w2):
    h=np.matmul(w1.T,input_word[index])
    u=np.matmul(w2.T,h)
    y = softmax(u)
    return h,u,y
```

4) Back Propagation

In [95]:
```
def backprop(input_word,w1,w2,lr,h,y_pred,index,window_size):
    front = input_word[index-window_size : index]
    back = input_word[index+1 : index+window_size+1]
    window_OH = np.concatenate([front,back])

    # output -> hidden
    for j in range(w2.shape[1]):
        adjust = (y_pred-window_OH)[:,j].sum()*h
        w2[:,j] -= -lr*adjust

    # hidden -> input
    adjust2 = ((y_pred-window_OH).sum(axis=0)*w2).T
    w1-= lr*adjust2
    return w1,w2
```

$$\mathbf{h} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_I}$$

$$u_{c,j} = u_j = \mathbf{v}'_{w_j}{}^T \cdot \mathbf{h}$$

Input of j-th unit on the c-th panel of the output layer (c : # of Multinomial Distributions)

$$y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

3) Feed Forward

In [94]:

```
def feedforward(input_word,index,w1,w2):
    h=np.matmul(w1.T,input_word[index])
    u=np.matmul(w2.T,h)
    y = softmax(u)
    return h,u,y
```

$$\mathbf{h} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_I}$$

$$u_{c,j} = u_j = \mathbf{v}'_{w_j}{}^T \cdot \mathbf{h}$$

$$y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u_{j'})}$$

output of the j-th unit on the c-th panel

4) Back Propagation

In [95]:

```
def backprop(input_word,w1,w2,lr,h,y_pred,index,window_size):
    front = input_word[index-window_size : index]
    back = input_word[index+1 : index+window_size+1]
    window_OH = np.concatenate([front,back])

    # output -> hidden
    for j in range(w2.shape[1]):
        adjust = (y_pred-window_OH)[:,j].sum()*h
        w2[:,j] -= -lr*adjust

    # hidden -> input
    adjust2 = ((y_pred-window_OH).sum(axis=0)*w2).T
    w1-= lr*adjust2
    return w1,w2
```

Implementation

2. Define Functions
(Random Walk, Softmax, Feed Forward, Back Propagation)

3) Feed Forward

In [94]:
```
def feedforward(input_word,index,w1,w2):
    h=np.matmul(w1.T,input_word[index])
    u=np.matmul(w2.T,h)
    y = softmax(u)
    return h,u,y
```

4) Back Propagation

In [95]:
```
def backprop(input_word,w1,w2,lr,h,y_pred,index,window_size):
    front = input_word[index-window_size : index]
    back = input_word[index+1 : index+window_size+1]
    window_OH = np.concatenate([front,back])

    # output -> hidden
    for j in range(w2.shape[1]):
        adjust = (y_pred-window_OH)[:,j].sum()*h
        w2[:,j] -= -lr*adjust

    # hidden -> input
    adjust2 = ((y_pred-window_OH).sum(axis=0)*w2).T
    w1-= lr*adjust2
    return w1,w2
```

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

3) Feed Forward

In [94]:

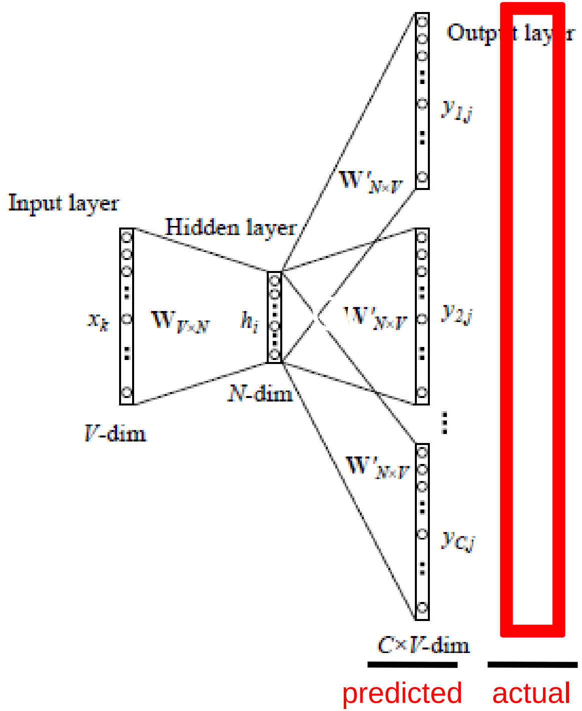
```
def feedforward(input_word,index,w1,w2):
    h=np.matmul(w1.T,input_word[index])
    u=np.matmul(w2.T,h)
    y = softmax(u)
    return h,u,y
```

4) Back Propagation

In [95]:

```
def backprop(input_word,w1,w2,lr,h,y_pred,index,window_size):
    front = input_word[index-window_size : index]
    back = input_word[index+1 : index+window_size+1]
    window_OH = np.concatenate([front,back])

    # output -> hidden
    for j in range(w2.shape[1]):
        adjust = (y_pred-window_OH)[:,j].sum()*h
        w2[:,j] -= -lr*adjust

    # hidden -> input
    adjust2 = ((y_pred-window_OH).sum(axis=0)*w2).T
    w1-= lr*adjust2
    return w1,w2
```

Actual Value of Output Words



Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

3) Feed Forward

In [94]:

```
def feedforward(input_word,index,w1,w2):
    h=np.matmul(w1.T,input_word[index])
    u=np.matmul(w2.T,h)
    y = softmax(u)
    return h,u,y
```

4) Back Propagation

In [95]:

```
def backprop(input_word,w1,w2,lr,h,y_pred,index,window_size):
    front = input_word[index-window_size : index]
    back = input_word[index+1 : index+window_size+1]
    window_OH = np.concatenate([front,back])

    # output -> hidden
    for j in range(w2.shape[1]):
        adjust = (y_pred-window_OH)[:,j].sum()*h
        w2[:,j] -= -lr*adjust

    # hidden -> input
    adjust2 = ((y_pred-window_OH).sum(axis=0)*w2).T
    w1-= lr*adjust2
    return w1,w2
```

$$\boxed{\mathbf{v}'_{w_j}{}^{(\text{new})} = \mathbf{v}'_{w_j}{}^{(\text{old})} - \eta \cdot \mathrm{EI}_j \cdot \mathbf{h}}$$

$$\mathrm{EI}_j = \sum_{c=1}^{C} e_{c,j} \qquad \text{for } j = 1, 2, \cdots, V.$$

$$\mathbf{v}_{w_I}^{(\text{new})} = \mathbf{v}_{w_I}^{(\text{old})} - \eta \cdot \mathrm{EH}^T$$

$$\mathrm{EH}_i = \sum_{j=1}^{V} \mathrm{EI}_j \cdot w'_{ij}.$$

Implementation

2. Define Functions

(Random Walk, Softmax, Feed Forward, Back Propagation)

3) Feed Forward

In [94]:

```
def feedforward(input_word,index,w1,w2):
    h=np.matmul(w1.T,input_word[index])
    u=np.matmul(w2.T,h)
    y = softmax(u)
    return h,u,y
```

4) Back Propagation

In [95]:

```
def backprop(input_word,w1,w2,lr,h,y_pred,index,window_size):
    front = input_word[index-window_size : index]
    back = input_word[index+1 : index+window_size+1]
    window_OH = np.concatenate([front,back])

    # output -> hidden
    for j in range(w2.shape[1]):
        adjust = (y_pred-window_OH)[:,j].sum()*h
        w2[:,j] -= -lr*adjust

    # hidden -> input
    adjust2 = ((y_pred-window_OH).sum(axis=0)*w2).T
    w1-= lr*adjust2
    return w1,w2
```

$$\mathbf{v}'_{w_j}{}^{(\text{new})} = \mathbf{v}'_{w_j}{}^{(\text{old})} - \eta \cdot \text{EI}_j \cdot \mathbf{h}$$

$$\text{EI}_j = \sum_{c=1}^{C} e_{c,j} \qquad \text{for } j = 1, 2, \cdots, V.$$

$$\boxed{\mathbf{v}_{w_I}^{(\text{new})} = \mathbf{v}_{w_I}^{(\text{old})} - \eta \cdot \text{EH}^T}$$

$$\text{EH}_i = \sum_{j=1}^{V} \text{EI}_j \cdot w'_{ij}.$$

Implementation

3. Skip Gram

3. Skip-Gram

In [96]:

```
def Skipgram(input_word, reduced_dim, lr, walk_size, window_size,epoch):
    W1 = np.random.random((input_word.shape[0],reduced_dim))
    W2 = np.random.random((reduced_dim, input_word.shape[0]))

    for _ in range(epoch):
        input_word = copy(input_word)
        shuffle(input_word)
        for index in range(input_word.shape[0]):
            RW = input_word[random_step(index,walk_size)]
            for i in range(len(RW)):
                h,u,y = feedforward(RW,i,W1,W2)
                W1,W2 = backprop(RW,W1,W2,lr,h,y,i,window_size)

    return W1,W2
```

[Input Variables]

1. input_word :
 Matrix of Input Words
(34 x 34 Identity Matrix)

2. reduced_dim :
Dimension of the embedded vector

3. lr :
Learning Rate

4. walk_size :
Walk length in random walk

5. window_size :
(one-sided) Size of the window from the index

6. epoch :
Walks per vertex

Implementation

3. Skip Gram

3. Skip-Gram

In [96]:

```python
def Skipgram(input_word, reduced_dim, lr, walk_size, window_size,epoch):
    W1 = np.random.random((input_word.shape[0],reduced_dim))
    W2 = np.random.random((reduced_dim, input_word.shape[0]))

    for _ in range(epoch):
        input_word = copy(input_word)
        shuffle(input_word)
        for index in range(input_word.shape[0]):
            RW = input_word[random_step(index,walk_size)]
            for i in range(len(RW)):
                h,u,y = feedforward(RW,i,W1,W2)
                W1,W2 = backprop(RW,W1,W2,lr,h,y,i,window_size)

    return W1,W2
```

[Process]

1) Initialize weight
(uniform distribution)
(W1 : input – hidden Weight)
(W2 : hidden – output Weight)

2) Shuffle the words

3) Implement a Random Walk

4) Feed Forward
(with the vertices selected by RW)

5) Back Propagation

6) Return Weights

3. Result

Visualization of Network

Result

4. Result

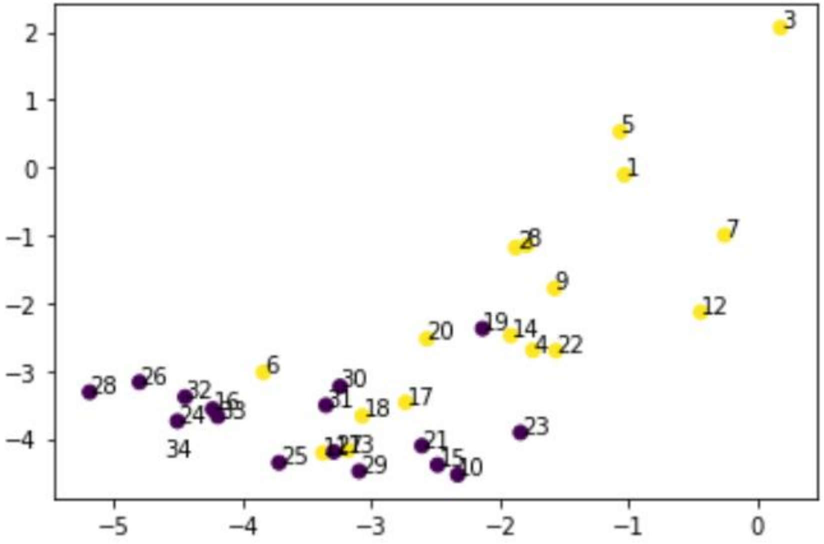
```python
w1,w2 = Skipgram(OH,reduced_dim=2, lr=0.02,
                 walk_size=10,window_size=3,epoch=7)
```

```python
Emb = np.matmul(OH,w1)
```

```python
Emb_df = pd.DataFrame({'X':Emb[:,0], 'Y':Emb[:,1],'Label':range(1,35)})

blue = [1,2,3,4,5,6,7,8,9,11,12,13,14,17,18,20,22]
red = list(set(range(0,34))-set(blue))

Emb_df.loc[Emb_df.Label.isin(blue),'Color']=1
Emb_df.loc[Emb_df.Label.isin(red),'Color']=0
```

Visualization

```python
plt.scatter(Emb_df['X'], Emb_df['Y'], c=Emb_df['Color'])

for i,txt in enumerate(Emb_df['Label']):
    plt.annotate(txt, (Emb_df['X'][i], Emb_df['Y'][i]))
```



Reference

[1] Bryan Perozzi, Rami Al-Rfou, Steven Skiena : *Deepwalk : Online Learning of Social Representations*

[2] Xin Rong : *word2vec Parameter Learning Explained*

Thank You!!