Parsing with graph convolutional networks and clustering

Alexander Sak[*]

Moscow State University of Civil Engineering, Yaroslavskoe shosse, 26, Moscow, 129337, Russia

Abstract. When designing machine translation systems, an important task is to represent data using graphs, where words act as vertices, and relations between words in a sentence act as edges. One of these tasks at the first stage of the analysis is the classification of words as parts of speech, and at the next stage of the analysis is to determine the belonging of words to the sentence members' classes. A robust approach to carry out such a classification is to determine words embeddings by using Graph Convolutional Networks at the beginning of the analysis and then to apply k-means clustering which is an algorithm that splits objects (words) into groups. To determine weights an ordinary network is applied to obtained hidden layers in order to use these weights in subsequent analysis.

1 Introduction

In linguistics, graph theory solves many problems related to the representation of formal relations between the components of a sentence. The correct description of a sentence with the help of a graph allows to a certain extent to use, in addition to syntactic, semantic connections between tokens.

When translating from English into Russian, at the first stage of the sentence analysis, special attention is paid to the problem of polysemy / homonymy of lexical units in a sentence. Each lexeme receives from the online dictionary [1] a set of parts of speech that this word can correspond to in a sentence. Every possible sentence, i.e. a combination of parts of speech can be considered as a path in a graph, which vertices are a complete set of possible parts of speech corresponding to a given word [2]. Each possible part of speech corresponding to the i-th word is connected by an edge with each possible part of speech of the (i + 1) -th word. Using the statistical data of the linguistic environment it's possible to assign a certain weight to each part of speech corresponding to a word. The cheapest way on this graph will be the best interpretation of the sentence in terms of determining whether its words belong to certain parts of speech.

Determining what particular part of speech the words of a sentence belong to is the first step to interpret it and to find a subordination tree with a further transition to relations in the sentence shifting to the members of the sentence.

[*] Corresponding author: sakan@mgsu.ru

2 Methods

Parsing is narrowly connected to the classification problem.

Our system has a Matrix class consisting of the string fields named "partofspeech", "word", as well as a node field belonging to the "Wordfeatures" class, which includes each word of the sentence and all its uses, taken from an online dictionary. An English lexeme can refer to different parts of speech, which is taken into account in the online dictionary:

In the sentence "***The people gave my friend a book***" we have the following paradigm for the distribution of words by parts of speech:

Table 1. The belonging of the sentence words to different parts of speech.

The	article		
people		noun	verb
give		noun	verb
my	pronoun		
friend	adjective	noun	verb
a	article		
book	adjective	noun	verb

Analysis methods based on the synthesis of linear algebra and graph theory are improving and this has led to the emergence of a new direction in artificial intelligence, known as the Graph Convolutional Network [4]. Graph convolutional networks combine an analysis approach based on graph theory and convolutional neural networks.

First, we need to analyze the data at our disposal, because using data, for example, in a matrix of features, will allow to obtain a certain result even without training the network, which would significantly reduce the data processing time when implementing a machine translation system. Table 1 indicates data from an online dictionary about the belonging of sentence words to different parts of speech. It's possible to determine appropriate feature values and include those values in the X feature matrix. There are several approaches to fill in the X matrix. One-**Hot Encoding** is the most common, correct way to deal with non-ordinal categorical data. It consists of creating an additional feature for each group of the categorical feature and mark each observation belonging (Value=1) or not (Value=0) to that group. There is another approach to determine features. A lesser known, but very effective way of handling categorical variables, is **Target Encoding**. It consists of substituting each group in a categorical feature with the average response in the target variable [16].

Neither approach suits us. With target encoding, the average value of the variable for different parts of speech will be the same, which will not give us the expected result. We set the following input features**: article - 0.1; preposition -0.2; adjective-0.3: adverb - 0.4; pronoun - 0.5; noun-0.6; verb-0.7.** When filling in the matrix of features, if a word can belong to several parts of speech at once, these values are summed up. Let's consider the above example of the sentence ***The people give my friend a book***. We have the following matrix:

X=[0.1 -0.1;1.3 -1.3;1.3 -1.3;0.5 -0.5; 1.6 -1.6;0.1 -0.1;1.6 -1.6];

We also need an adjacency matrix. At this stage, we do not know the structure of the sentence, so we will set the connections of each vertex exclusively with its neighbors. The first and last words in a sentence have a formal relationship only with the previous and subsequent words, respectively, but we assign them a relationship with the two previous or subsequent words which don't exist to align their values.

In this case, we find new features with the formula:

Z=ReLu(D_hat*A_hat*ReLu(D_hat*A_hat*X*W^0)*W^1) (1) [4]

X=[0.1 -0.1;1.3 -1.3;1.3 -1.3;0.5 -0.5; 1.6 -1.6;0.1 -0.1;1.6 -1.6];

```
D=[3 0 0 0 0 0 0;0 3 0 0 0 0 0;0 0 3 0 0 0 0;0 0 0 3 0 0 0;0 0 0 0 3 0 0;0 0 0 0 0 3 0;0 0 0 0 0 0 3 ];
D_hat=inv(D);
W=[1 -1;-1 1 ];
disp (X)
disp(D_hat )
disp(A_hat)
Z=D_hat*A_hat*X*W;
Z1=D_hat*A_hat*Z*W;
disp(Z1)
```

Multiplying A * X in this form denotes the sum of the features of their neighbors. This means that the convolutional layer of the graph represents each layer as a resultant of its environment. It turns out that the representation of the vertex does not include its own features. The representation is the result of interaction of features of neighboring vertices, i.e. only vertices with a loop include their own features in this set. Vertices with a large value of the degrees will have a greater value in the representation of their features, and vice versa, the lower the degree at the vertex, the lower the value of its features. This can lead to fading or explosive growth of gradients. It can also create problems when using stochastic descent algorithms, which are used to train networks and are sensitive to the range of values of each input feature. Thus, it is necessary to add the identity matrix ***I*** to the adjacency matrix.

```
for i=1:5
   A(i,i)=1;
  End
A_hat=[1 1 1 0 0 0 0;1 1 1 0 0 0 0;0 1 1 1 0 0 0;0 0 1 1 1 0 0;0 0 0 1 1 1 0;0 0 0 0 1 1 1;0 0 0 0 1 1 1];
```

The feature representation can be normalized by the vertex degree by multiplying the adjacency matrix by the inverse vertex degree matrix ***D***. First, we find the vertex degree matrix for the adjacency matrix. Thus, the simplified propagation rule will be: f(X,A)=D^{-1}AX. Here, in each row, the values of each row of the adjacency matrix are divided by the degree of the vertex corresponding to the row. We apply the propagation rule to the transformed adjacency matrix. First of all, it is necessary to apply the weights for the vertices: W = [1, -1; -1,1];. At this stage, weights are applied arbitrarily

```
D_hat=inv(D);
```

And we apply the ReLu activation function to each product:

```
Z=D_hat*A_hat*X*W;
Z1=D_hat*A_hat*Z*W;
```

We have the result:

Table 2. Two hidden layers ReLu activation function

	3.7738
	3.7738
	4.0889
Z=	3.8667
	3.9556
	3.9111
	3.9111

On just two layers, without calculating the gradient of the weights W by using the loss function *loss*=$\sum_{i=1}^{p}(\mathrm{ti}-\mathrm{oi})$, we get an interesting result: the largest value is assigned to the third element, i.e. the verb. Elements of the noun groups *the people* and *a book* comprised of an article and a noun receive the same feature values, respectively, despite the completely different values of their input values of the adjacency matrix. The 4 and 5 elements also did not diverge much in meanings, although the possessive pronoun has a value of 3.8667, and the crucial word has 3.9556. It is obvious that the sentence is divided into groups already at the stage of preliminary preparation of the two hidden layers.

We can also change the formula of the hidden layers and see how the features change.

Z=softmax(A_hat*ReLu(A_hat*X*W^0)W$^{(1)}$) (2) [5]

In this formula, the first layer is activated using the activation function ReLu, which activates a value greater than zero and otherwise returns zero, f(z)=max(0,z).

The second layer is supposed to use the modified weight matrix W after the loss function has been applied. We multiply the first layer by the original matrix to see the result without training. The second layer is activated by the ***softmax*** activation function:

$$\sigma(z)_i = e^{Zi} \Big/ \sum_{k=1}^{K} e^{Zk} \qquad (3)$$

We obtain the result as follows:

Table 3. The output layer softmax activation function

	0.0310
	0.0310
	0.5097
Z=	0.06
	0.15
	0.1029
	0.1029

As we can see, the predicate in the 3rd position gets the greatest value again. Each element of the noun phrase in the position of the subject and the direct object receives the same values. The possessive pronoun of the indirect object gets the least meaning, and the noun of the indirect object has a meaning within the meaning of the object [0,1; 0.5].

We have another rule of multilayer distribution:

H$^{(l+1)}$=σ(D_hat$^{-1/2}$*A_hat*D_hat$^{-1/2}$*H$^{(l)}$W$^{(l)}$) (4) [6]

where σ stands for the activation function, ReLu in this case, $H^{(l)} \in R^{N \times D}$ is the activation matrix of the l-th layer

$H^{(0)}$=X.

We have the following result:

Table 4. D_hat$^{-1/2}$.
Because we normalize twice, we change "-1" to "-1/2"

	3.06
	3.06
	3.2120
Z=	3.1320
	3.2040
	3.1683
	3.1683

A situation similar to the previous cases is observed: the predicate gets the greatest value again. But this time the difference from the indirect complement value is negligible. The same values are also observed in the respective noun groups of the subject and the direct object. It is obvious that formula (4) suits better for multilayer propagation, since identifies the predicate with a significant difference from the values of other elements, clearly tells the difference between the possessive pronoun and the noun-object, and also indicates the difference in the values of the subject and object.

3 Results

Putting it all together in Python:

```
import numpy as np
from networkx import karate_club_graph, to_numpy_matrix
from sklearn.preprocessing import OneHotEncoder
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib inline
from math import sqrt
color_map=[]
color_map.append('green')
zkc=karate_club_graph()
from scipy.linalg import fractional_matrix_power
from sklearn.cluster import KMeans
A_hat=np.matrix([                      X=np.matrix([           D=np.matrix([
[1, 1, 1, 0, 0, 0, 0],                     [0.1, -0.1],             [3, 0, 0, 0, 0, 0, 0],
[1, 1, 1, 0, 0, 0, 0],                     [1.3, -1.3],              [0, 3, 0, 0, 0, 0, 0],
[0, 1, 1, 1, 0, 0, 0],                     [1.3, -1.3],              [0, 0, 3, 0, 0, 0, 0],
[0, 0, 1, 1, 1, 0, 0],                      [0.5, -0.5],             [0, 0, 0, 3, 0, 0, 0],
[0, 0, 0, 1, 1, 1, 0],                      [1.6, -1.6],             [0, 0, 0, 0, 3, 0, 0],
[0, 0, 0, 0, 1, 1, 1],                       [0.1, -0.1],            [0, 0, 0, 0, 0, 3, 0],
[0, 0, 0, 0, 1, 1, 1]],                       [1.6, -1.6]],            [0, 0, 0, 0, 0, 0, 3]],
   dtype=float  )                                dtype=float)            dtype=float)
```

The weights are set randomly for both hidden layers.

```
W_1 = np.random.normal(
    loc=0, scale=1, size=(2, 2))
W_2 = np.random.normal(
    loc=0, size=(2, 2))
D_hat=D**-1
def gcn_layer(A_hat, D_hat, X, W):
    return relu(D_hat * A_hat * X * W)
def relu (H):
    H[H<0]=0
    return H
H_1 = gcn_layer(A_hat, D_hat, X, W_1)
H_2 = gcn_layer (A_hat, D_hat, H_1, W_2))
output=H_2
```

After finding the second hidden layer H_2, we look for min and max:

```
min1=10  max1=-10
for idx in range(len(output)):
  if output[idx,0]<min1:
    min1=output[idx,0]
  if output[idx,1]<min1:
    min1=output[idx,1]
……………………………..
print(min1)
print(max1)
print (output)
out[[0.15011922 0.44917613]
    [0.15011922 0.44917613]
    [0.16248198 0.4861671 ]
    [0.15365144 0.45974498]
    [0.15718366 0.47031383]
    [0.15541755 0.4650294 ]
    [0.15541755 0.4650294 ]]
```

We draw a graph where the words are denoted by the resulting numerical values. You can see that the edges of the graph sequentially connect the vertices with each other, since this is the only graph representation we have of this sentence.

```
G1=nx.Graph()
G1.add_node(output[0,0])
for idx in range(len(output)-1):
  G1.add_node(output[idx,0])
  G1.add_edge(output[idx-1,0],output[idx,0])
nx.draw_circular(G1, with_labels=True)
```

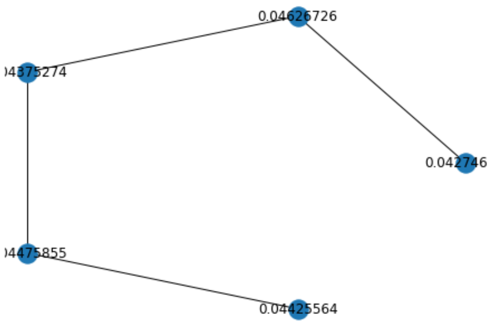

Fig. 1. Initial sentence representation by means of a graph

Depending on the weights set randomly, the values of the hidden layers of the neural network change, but the proportions of the relationships between the elements remain unchanged. We apply k-means clustering. This is the most popular clustering method. The idea of the algorithm is to minimize the total squared deviation of clusters' points from the centers of these clusters

$$V = \sum_{i=1}^{K} \sum_{x \in Si} (x - \mu i)^2$$

Python has a built-in method for finding K-means. We have 7 parts of speech, of which two – the article and the preposition are determined in advance, so we set 5 clusters.

```
kmeans=KMeans(n_clusters=5)
kmeans.fit(vec)
assignement=kmeans.predict(vec)
group1=np.where(assignement==0)[0]+1
……………………………………………
group5=np.where(assignement==4)[0]+1
print(group1)
……………
print(group5)
```

We get the result: [1 2] [6 7] [3] [5] [4]

There are five classes, because the first and second words, as well as the fifth and sixth, have the same numerical values of features (word embedding) and form separate groups (clusters). Obviously, K-means is great at grouping words into clusters based on data preprocessed with Graph Convolutional Networks. The network, consisting of two hidden layers, assigns the maximum meaning to the verb-subject, the elements of the noun phrase, consisting of the article and the noun, receive the same weights with very different input data for the features matrix X. The possessive pronoun and its defining noun do not hit the same cluster but this is a fixable problem. Perhaps, when setting the weights after training, the results will improve. Clustering a dataset into groups is a form of unsupervised learning. A dataset of points will turn into a set of labels, according to the patterns discovered by the algorithm.

Taking into account the distribution of vertices across clusters, we can color each group with its own color.

```
for idx in range(len(output)):
  my_set=idx+1
  x=0
  for i in range(len(group1)):
    if my_set==i:
      x=1
  if x==1:
    color_map.append('red')
………………………………
for i in range(7-len(color_map)):
  color_map.append('purple')
plt.scatter([output[:,0]],[output[:,1]],c=color_map,marker='o',s=60)
plt.show()
```

We get the following coloring of the vertices:

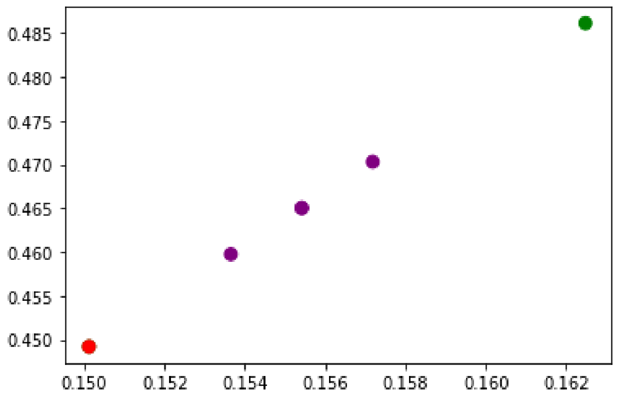

Fig. 2. Classification of words in a sentence by using clustering

The figure depicts 5 groups-clusters, not 7. This is due to the fact that the values of the articles' features coincide with the values of the nouns' features. Now we need to match a new graph to our sentence using clustering. At the first stage, we add the vertex with the highest weight that is verb (subject). If there is more than one element in the group, we establish a link between the last element of the group and the verb, and the rest of the group's elements are linked together. If there is one element in the cluster, and it is not the main one, a connection is established between these elements and between the main element of this new group and the verb.

```
G1.add_node(3)
if len(group1)>1:
  for i in range(len(group1)-1):
    G1.add_node(group1[i])
    G1.add_node(group1[i+1])
    G1.add_edge(group1[i],group1[i+1])
  G1.add_edge(3,group1[len(group1)-1] )
else:
  if group1[0] !=3:
    my_set2.append(group1[0])
............................................
for j in range(1,len(my_set2)):
  G1.add_node(my_set2[i])
  G1.add_edge(my_set2[i-1],my_set2[i])
G1.add_edge(my_set2[len(my_set2)-1],3)
nx.draw_circular(G1, with_labels=True)
```

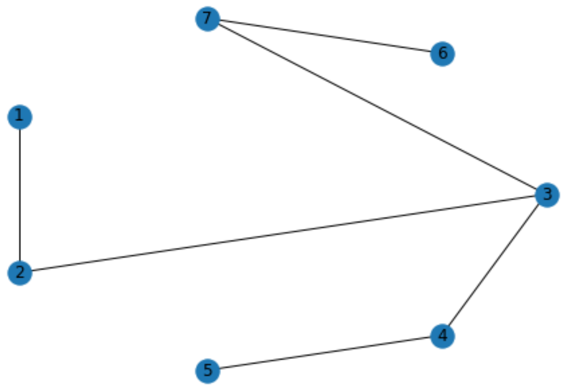

Fig. 3. Representation of hierarchical relationships in a sentence

To train weights in hidden layers, we launch the neural network [15]:

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
```

Here we use the second layer of GCN

```
x_train=np.array([[0.15011922, 0.44917613],
              [0.15011922, 0.44917613],
              [0.16248198, 0.4861671],
              [0.15365144, 0.45974498],
              [0.15718366, 0.47031383],
              [0.15541755, 0.4650294],
              [0.15541755, 0.4650294 ]])
```

Here we use the values we want to obtain by approximating the input ones to those of parts of speech which will clearly denote them:

```
y_train=np.array([[0.1], [0.6],  [0.7],  [0.5],  [0.6],   [0.1],  [0.6]])
model=Sequential()
model.add(Dense(num_neurons,input_dim=2))
model.add(Activation('tanh'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.summary()
sgd=SGD(lr=0.1)
model.compile(loss='binary_crossentropy',optimizer=sgd,metrics=['accuracy'])
model.fit(x_train,y_train,epochs=100)
model.predict(x_train)
model.predict_classes(x_train)
model.predict(x_train)
```

The result obtained after a hundred epochs:

```
array([[0.45717323],
       [0.45717323],
       [0.45823628],
       [0.45747808],
       [0.45778203],
       [0.45763016],
       [0.45763016]], dtype=float32)
```

As we can see, after training, the structure of the values of the output layer is saved, and the obtained weights are saved for initialization during subsequent use of the network without training using the command model.save_weights (“basic_weights.h5)
Thus, “supervised learning” is used to obtain the weights because the values in the test case are preset.

4 Discussion

In this case, a set of three adjacent words can be considered as a sliding window (convolution) of a convolutional neural network over the text, which, in addition to predetermined feature values, makes up for the lack of knowledge about the structure of a sentence when classifying its nodes. In subsequent works, we will consider the construction of a graph of syntactic relations in a sentence based on this graph networks using the weights obtained after training a convolutional neural network.

5 Conclusion

In this study, we consider a graph convolutional network, one of which goals is the classification of the graph vertices. Without knowing the syntactic structure of the sentence, and using only the connection with neighboring nodes (words), as well as data on the belonging of words to certain parts of speech, we can achieve certain success in parsing by using the K-means clustering algorithm, without even using the loss function and without training the weight matrix.

References

1. A.N. Sak, *Identificacia chlenov predlozhenia kak osnovnaya zadacha machinnogo /The identification of sentence members as the main purpose of machine translation/* in Naucnoye obozreniye, Ed. 14, Moscow, Russia (2015)

2. Skiena Stiven, *Algoritmy. Rucovodstvo po razrabotke/Algorithms. Guidelines for developing/ 2 edition Translated from English* in BVH-Petersburg, Saint-Petersburg, Russia (2011)

3. Graph Neural Network // https://neerc.ifmo.ru/wiki/

4. Tobias Skovgaard Jepsen, *How to do Deep Learning on Graphs with Graphs Convolutional Networks,* https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780

5. Chau Pham, *Graph Convolutional Networks (GCN),* https://medium.com/ai-in-plain-english/graph-convolutional-networks-gcn-baf337d5cb6b

6. Thomas N. Kipf, Max Welling, *Semi-Supervised Classification with Graph Convolutional Networks*, https://arxiv.org/abs/1609.02907

7. William L. Hamilton, Rex Ying, Jure Lescovec, *Representation Learning on Graphs: Methods and Applications*// Department of computer Science Stanford University, Stanford, CA, 94305 https://arxiv.org/abs/1709.05584

8. Jasmijn Bastings, Ivan Titov, Wilker Aziz, *Graph Convolutional Encoders for Syntax-aware Neural Machine translation* https://www.aclweb.org/anthology/D17-1209

9. Flawson Tong, *Everything you need to learn about graph theory*, https://towardsdatascience.com/graph-theory-and-deep-learning-know-hows-6556b0e9891b

10. Trist'n Joseph, *The mathematics Behind Deep Learning* https://towardsdatascience.com/the-mathematics-behind-deep-learning-f6c35a0fe077

11. Michael Bronstein, *Latent graph neural networks: Manifold learning 2.0?* https://towardsdatascience.com/manifold-learning-2-99a25eeb677d

12. Connor Shorten, *Embedding Graphs with Deep Learning* https://towardsdatascience.com/embedding-graphs-with-deep-learning-55e0c66d7752

13. Flawson Tong, *Graph Embedding for Deep Learning* https://towardsdatascience.com/overview-of-deep-learning-on-graph-embeddings-4305c10ad4a4

14. James McCaffrey, *Deep Neural Networks IO Using C#* https://jamesmccaffrey.wordpress.com/2017/08/02/deep-neural-network-io-using-c/

15. Hobson Lane, Cole Howard, Hannes Max Hapke, *Natural Language Processing in Action*, Manning, Shelter Island (2019)

16. Eugenio Zuccarelli, *Handling Categorical Data. The Right Way* https://towardsdatascience.com/handling-categorical-data-the-right-way-9d1279956fc6