ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

.dit/*
Informatics & Telematics

Εργασία στο μάθημα Τεχνολογίες Γραφημάτων

Ιανουάριος 2025

Κωνσταντίνος Χάφης *it22120*
Άρβι Χότζα *it22121*

1 Μεθοδολογία **1**
1.1 Δημιουργία του Dataset 1
1.2 Υλοποίηση του CBOW Μοντέλου 2
1.3 Διαδικασία Εκπαίδευσης 3
1.4 Υπολογισμός Ενσωματώσεων & Οπτικοποίηση 3
2 Αποτελέσματα **4**
3 Χρήσεις της Μεθόδου DeepWalk **5**
Αποθετήριο Κώδικα **5**

1 Μεθοδολογία

Σε αυτή την ενότητα περιγράφεται η υλοποίηση της μεθόδου **DeepWalk** χρησιμοποιώντας το **CBOW** μοντέλο. Η υλοποίηση έχει τρία βασικά στάδια:

1. Δημιουργία του dataset μέσω **τυχαίων περιπάτων**.
2. Εκπαίδευση του μοντέλου **CBOW**.
3. Υπολογισμός ενσωματώσεων και οπτικοποίηση των αποτελεσμάτων.

1.1 Δημιουργία του Dataset

Για την εξαγωγή ενσωματώσεων (embeddings) των κόμβων, αρχικά δημιουργούμε τυχαία μονοπάτια στο γράφημα. Για το σκοπό αυτό χρησιμοποιούμε τη βιβλιοθήκη **NetworkX** και εφαρμόζουμε τη διαδικασία των **random walks**.

Η κλάση `GraphDataset` διαχειρίζεται τη δημιουργία, αποθήκευση και ανάκτηση του dataset με δυναμικό τρόπο:

```
class GraphDataset(Dataset):
   def __init__(self, graph:nx.Graph, gamma: int, walk_length: int, window_size:
int):
       self.window_size = window_size
       self.graph = graph
       self.gamma = gamma
       self.walk_length = walk_length
```

```
        self.dataset = self.generate_dataset()
```

Ο αλγόριθμος εκτελεί **γ τυχαίους περιπάτους** από κάθε κόμβο με σταθερό μήκος `t`. Στη συνέχεια, τα δεδομένα αποθηκεύονται σε μορφή context-target pairs:

```
    def generate_dataset(self):
        if self.check_if_exists():
            print("Reading from stored dataset")
            dataset = pd.read_csv(self._get_dataset_name())
            return [(torch.tensor(list(map(int, x[0][1:-1].split(",")))),
torch.tensor(x[1])) for x in dataset.values]
        walks = self.generate_walks()
        dataset = []

        # half window size
        hw_size = (self.window_size - 1) // 2
        for walk in walks:
            for i in range(hw_size, self.walk_length - hw_size):
                target = walk[i]
                context = walk[i - hw_size:i] + walk[i + 1:i + hw_size + 1]
                dataset.append((context, target))

        pd.DataFrame(columns=["context", "target"], data=dataset)\
            .to_csv(self._get_dataset_name(), index=False)

        dataset = [(torch.tensor(context), torch.tensor(target)) for context, target
in dataset]

        return dataset
```

Το αποτέλεσμα είναι ένα **σύνολο δεδομένων** που αποτελείται από συμφραζόμενα κόμβων και τους αντίστοιχους target nodes, το οποίο θα χρησιμοποιηθεί στην εκπαίδευση του CBOW μοντέλου.

1.2 Υλοποίηση του CBOW Μοντέλου

Το CBOW μοντέλο υλοποιείται χρησιμοποιώντας **PyTorch**. Η αρχιτεκτονική του περιλαμβάνει:

- **Embedding Layer**: Μαθαίνει αναπαραστάσεις κόμβων.
- **Mean Pooling**: Υπολογίζει το μέσο όρο των συμφραζόμενων κόμβων.
- **Linear Layer**: Χαρτογραφεί την κρυφή αναπαράσταση σε πιθανότητες κόμβων.

- **Softmax Activation**: Μετατρέπει τα logits σε πιθανότητες κατηγοριών.

Η βασική υλοποίηση του CBOW φαίνεται παρακάτω:

```
class CBOW(nn.Module):
   def __init__(self, vocab_size: int, embedding_dim: int):
       super(CBOW, self).__init__()
       self.embeddings = nn.Embedding(vocab_size, embedding_dim)
       self.linear = nn.Linear(embedding_dim, vocab_size)
       self.sf = nn.LogSoftmax(dim=1)

   def forward(self, w):
       w = self.embeddings(w)
       w = w.mean(dim=1)
       w = self.linear(w)
       w = self.sf(w)
       return w
```

1.3 Διαδικασία Εκπαίδευσης

Το CBOW εκπαιδεύεται με τη χρήση της **CrossEntropyLoss** και του **Adam Optimizer**. Για την αποφυγή υπερ-εκπαίδευσης (overfitting), εφαρμόζεται **Early Stopping**.

Η εκπαίδευση εκτελείται για **500 εποχές** (by default), με τα δεδομένα να χωρίζονται σε **training (80%)** και **test set (20%)**.

1.4 Υπολογισμός Ενσωματώσεων & Οπτικοποίηση

Αφού το μοντέλο εκπαιδευτεί, εξάγουμε τις ενσωματώσεις των κόμβων και τις οπτικοποιούμε χρησιμοποιώντας **t-SNE**:

```
reduced_embeds = TSNE(n_components=2, random_state=42).fit_transform(
          extract_embeddings(model))
acc = KMeans_Accuracy(reduced_embeds, G)
```

Επιπλέον, χρησιμοποιούμε **K-Means Clustering** για να δούμε και να αξιολογήσουμε καλύτερα αν τα embeddings διαχωρίζουν τις πραγματικές κοινότητες του Karate Club Graph:

```
 def visualize_embeddings(reduced_embeddings, labels):
   # Perform k-means clustering with 2 clusters
   kmeans = KMeans(n_clusters=2, random_state=0).fit(reduced_embeddings)
   clusters = kmeans.labels_

   plt.figure(figsize=(10, 10))
   for i, label in enumerate(labels):
```

```
        x, y = reduced_embeddings[i, :]
        color = 'red' if clusters[i] == 0 else 'blue'
        plt.scatter(x, y, c=color)
        plt.annotate(label, (x, y), textcoords="offset points", xytext=(0, 10),
ha='center')

    plt.show()
```

2 Αποτελέσματα

Από τις δοκιμές που κάναμε αρχικά και μετέπειτα με τον έλεγχο όλων τον πιθανών παραμέτρων (custom GridSearch), καταλήξαμε στο συμπέρασμα πως σε γενικές γραμμές κατά την εκπαίδευση όσο μικρότερο είναι το window_size σε συνδυασμό με τους περιπάτους να είναι όσο πιο κοντά στο μέγεθος του window_size (δλδ window_size = walk_length) θα δώσει τα καλύτερα αποτελέσματα κατά την εκπαίδευση.

```
py training.py --window_size 3 --batch_size 256 --num_walks 2000
Current running configuration:
{"window_size": 3, "walk_length": 3, "num_walks": 2000, "embedding_dim": 12,
"batch_size": 256, "epochs": 500}
Reading from stored dataset
Dataset size: 68000 walks
Final Test Accuracy: 66.3194%
Final Train Accuracy: 67.2517%
Early stopping - Last epoch: 347
Accuracy w/ KMeans: 94.12%
```

```
py training.py --window_size 3 --walk_length 2 --batch_size 256 --num_walks 2000
Current running configuration:
{"window_size": 3, "walk_length": 5, "num_walks": 2000, "embedding_dim": 12,
"batch_size": 256, "epochs": 500}
Reading from stored dataset
Dataset size: 204000 walks
Final Test Accuracy: 59.5459%
Final Train Accuracy: 59.5770%
Early stopping - Last epoch: 286
Accuracy w/ KMeans: 97.06%
```

```
py training.py --window_size 5 --batch_size 256 --num_walks 2000
Current running configuration:
{"window_size": 5, "walk_length": 5, "num_walks": 2000, "embedding_dim": 12,
"batch_size": 256, "epochs": 500}
Reading from stored dataset
Dataset size: 68000 walks
Final Test Accuracy: 26.7289%
Final Train Accuracy: 27.6005%
```

```
Early stopping - Last epoch: 335
Accuracy w/ KMeans: 94.12%
```

Βλέπουμε πως όσο πιο πολύ μεγαλώνουμε το μέγεθος του παραθύρου και το μήκος των μονοπατιών τόσο πιο μικρή είναι η ακρίβεια του μοντέλου.

Ωστόσο με την βοήθεια της αναλυτικής ικανότητας του KMeans μπορούμε να δούμε πως η κατηγοριοποίηση των κόμβων σε 2 ομάδες (που αυτό είναι και το ζητούμενο) σε γενικές γραμμές είναι καλή ανεξαρτήτως.

3 Χρήσεις της Μεθόδου DeepWalk

Εφόσον η μέθοδος DeepWalk ενισχύει τη καταγραφή και αποτύπωση σημασιολογικών ομοιοτήτων με τη βοήθεια ενός μοντέλου CBOW ή Skip-Gram, μπορούμε έτσι να επεκτείνουμε τις δυνατότητες του αρχικού μοντέλου πέραν από τη κατηγοριοποίηση κόμβων σε ένα γράφο. Μερικές εφαρμογές που μπορούσαν να κάνουν χρήση της μεθόδου αυτής είναι:

- **Συστήματα Συστάσεων**: Αν ένας χρήστης αλληλεπιδρά με συγκεκριμένα προϊόντα, η DeepWalk μπορεί να προβλέψει ποια άλλα προϊόντα θα μπορούσαν να τον ενδιαφέρουν.
- **Πρόβλεψη Λειτουργίας Γονιδίων**: Αναγνώριση γονιδίων με παρόμοιες λειτουργίες αντιμετωπίζοντας τις ακολουθίες ως λέξεις και τα γονίδια ως προτάσεις.
- **Ανάλυση Καταγραφών (Log Analysis) για Ανίχνευση Ανωμαλιών:** Εντοπισμός ανωμαλιών σε καταγραφές συστήματος αναλύοντας ακολουθίες γεγονότων.

Αποθετήριο Κώδικα

https://github.com/Kostas-Xafis/DeepWalk