Universitat Politècnica de Catalunya

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Graph Convolutional Networks for context-aware recommender systems

Paula Gómez Duran

Advisors: Alexandros Karatzoglou, Ioannis Arapakis, Xavier Giró-i-Nieto

A thesis submitted in fulfillment of the requirements for the Master in Advanced Telecommunication Technologies

Barcelona, February 2020

Abstract

With more multimedia content being available than ever, the need of recommender systems is becoming crucial. Recent studies show that including context (such as daytime, last clicked items, weather, etc.) helps in order to give better recommendations. Factorization machines (FM) models [17] are an effective solution for context-aware recommendation tasks. However, FM models are based on the second-order interactions between features, which cause one of its major drawbacks: they cannot capture complex high-order signal interactions. Some work has already been proposed ([8], [9] and [24]) in order to address this problem. Yet, these methods use deep neural networks (DNN) to learn high-order interactions between features, which makes them either too expressive ([8] and [9]) or too computational expensive ([24]).

In this work, we propose to capture high-order interactions from the embedding level. We want the feature embedding to encode not only the information from the feature itself but also to aggregate the correlated knowledge from other features. Our approach to do so is by using Graph Convolutional Networks (GCN) instead of the common embedding layer that conforms recommendation models (such as FM). In that way, we will be able to keep the structure of FM models while having the high-order signals automatically contained in the interaction between feature embeddings.

Our solution is implemented in Pytorch [11] and it will result in a module, which could be included in any recommender system thus leading to an end-to-end model. We build the work following the next steps:

1. Build Graph Convolutional Network (GCN) model based on the work of [13], and adapt it so that it can output embeddings that contain information not just from the node itself but also form the correlated nodes as well.
2. Extend the work of [3] to context, by considering matrix completion for recommender systems from the point of view of link prediction on graphs.
3. Incorporate the GCN module for recommender systems (unifying the first and the second point) to Factorization Machines (FM) [17] in order to build the desired end-to-end model. Then, FM could be substituted by any other RS model such as [9], [8] or [5], which is also tested in our work.

Experiments indicate that context helps models to give better recommendations. Besides, it is shown that the incorporation of GCN allows to outperform the original FM and variations in most of the cases.

As future work, we propose some ideas in order to focus on the optimization for specific raking tasks as well as trying different variations of GCN to see whether it can outperform the baseline models in all the cases.

Resum

Amb l'àmplia i creixent quantitat de contingut multimèdia d'avui en dia, la necessitat d'utilitzar sistemes de recomanació s'ha tornat quelcom essencial. Alguns estudis recents mostren que la inclusió de context (hora del dia, últims clics, clima, etc.) millora la qualitat de les recomanacions.

Les Màquines de Factorització (FM) [17] són una solució efectiva per a les tasques de recomanació tenint en compte el context. Tanmateix, les FM es basen en les interaccions de segon ordre entre les incrustacions (embeddings) de característiques, fet que origina un dels seus principals problemes: no poder capturar senyals complexes d'alt ordre entre interaccions. Actualment existeixen treballs ([8], [9] i [24]) que adrecen aquest problema. Però, aquests mètodes utilitzen xarxes neuronals profundes (DNN) per capturar les interaccions d'alt ordre entre característiques, fet que els fa ser o bé massa complexes ([8] i [9]), o bé massa costosos computacionalment ([24]).

Proposem capturar les interaccions d'alt ordre des del nivell d'incrustacions. Proposem que les incrustacions de característiques no codifiquin només la informació de la pròpia característica sinó que també agreguin coneixement d'altres característiques corralades. Per fer-ho, proposem les Xarxes Convolucionals de Grafs (GCN) en lloc d'utilitzar la capa d'incrustacions que hi trobem als models de recomanació (com les FM). D'aquesta manera, podem mantenir l'estructura simple dels sistemes de recomanació mentre, a la vegada, tenim els senyals d'alt ordre continguts automàticament en la interacció entre incrustacions de característiques.

La nostra solució s'implementa en Pytorch [11] i esdevè un mòdul que podrà ser inclòs en qualsevol sistema de recomanació, conformant així un model end-to-end. Per dur a terme el treball seguim els següents passos:

- Elaborar un model de Xarxa Convolucional de Grafs (GCN) basat en el treball de [13], i adaptar-lo de tal forma que pugui retornar incrustacions que continguin informació tant del mateix node com també d'aquells nodes correlats.
- Ampliar el treball de [3] amb la incorporació del context, mitjançant l'extensió de l'algoritme de completació matricial (matrix completion) per sistemes de recomanació des del punt de vista de la predicció d'enllaços en els grafs.
- Incorporar el mòdul GCN per a sistemes de recomanació (unificant els dos primers punts) a les Màquines de Factorització (FM) [17] per tal d'elaborar el model end-to-end desitjat. En aquest punt, les FM podrien ser substituïdes per qualsevol altre model de sistemes de recomanació com [9], [8] o [5], que també incorporem en el nostre treball.

Els experiments demostren que el fet d'incorporar les dades del context ajuda als models a millorar les recomanacions. A més a més, es demostra que la incorporació de GCN als sistemes de recomanació millora el rendiment dels models de referència (baseline) en la majoria dels casos, tot i que no en tots encara.

Com a proposta de treball futur, plantegem algunes idees per potenciar l'optimització de les tasques enfocades al rànquing d'ítems. També proposem provar diferents variants de GCN per veure si es poden millorar els resultats de tots els models de recomanació consistentment.

Resumen

Con la amplia y creciente cantidad de contenido multimedia que hay hoy en día, la necesidad de utilizar sistemas de recomendación se ha vuelto algo esencial. Algunos estudios recientes muestran que la inclusión de contexto (hora del día, últimos clics, clima, etc.) mejora la calidad de las recomendaciones.

Las Máquinas de Factorización (FM) [17] son una solución efectiva para las tareas de recomendación teniendo en cuenta el contexto. Sin embargo, las FM se basan en las interacciones de segundo orden entre las incrustaciones (embeddings) de características, lo que origina uno de sus principales problemas: no poder capturar señales complejas de alto orden entre interacciones. Actualmente existen trabajos ([8], [9] y [24]) que tratan de resolverlo. Sin embargo, su propuesta es usar redes neuronales profundas (DNN) para capturar las interacciones de alto orden entre características, hecho que les hace ser o demasiado complejos ([8] y [9]) o demasiado costosos computacionalmente ([24]).

Proponemos capturar las señales de interacciones de alto orden desde el nivel de incrustaciones, codificando en estas no solo la información de la propia característica sino también la de otras correlacionadas. Para ello, proponemos las Redes convolucionales de Grafos (GCN) para capturar las señales de interacciones de alto orden en lugar de utilizar la capa de incrustaciones que encontramos en los modelos de recomendación (FM). De este modo, conseguiremos mantener la estructura simple de los sistemas de recomendación mientras que a la vez tendremos las señales de alto orden contenidas automáticamente en las incrustaciones de características.

Nuestra solución se implementa en Pytorch [11] y llega a ser un módulo que podrá ser incluido en cualquier sistema de recomendación, conformando así un modelo end-to-end de entrenamiento. Para llevar a cabo el trabajo seguimos los siguientes pasos:

- Elaborar un modelo de Red convolucional de Grafos (GCN) basado en el trabajo de [13] y adaptarlo de tal forma que pueda devolver incrustaciones que contengan información no sólo del mismo nodo sino también de aquellos nodos correlacionados.
- Ampliar el trabajo de [3] con la incorporación del contexto, mediante la extensión del algoritmo de completación matricial (matrix completion) para sistemas de recomendación desde el punto de vista de la predicción de enlaces en los grafos.
- Incorporar el módulo GCN (unificación de los dos pasos anteriores) a las Máquinas de Factorización (FM) [17] para elaborar un modelo end-to-end. En este punto, las FM podrían ser sustituidas por cualquier otro modelo de sistemas de recomendación como [9], [8] o [5], que también incorporamos en nuestro trabajo.

Los experimentos demuestran que el hecho de incorporar el contexto a los datos ayuda a los modelos a mejorar las recomendaciones. Además, se demuestra que la incorporación de GCN a los sistemas de recomendación supera en rendimiento a los modelos de referencia (baseline) en la mayoría de los casos, aunque no en todos ellos aún.

Como propuesta de trabajo futuro, planteamos algunas ideas para potenciar la optimización de las tareas enfocadas al ranking de ítems, así como probar diferentes variantes de GCN con el fin de mejorar los resultados de forma consistente.

Acknowledgements

First of all, I want to thank my advisors Alexandros Karatzoglou and Ioannis Arapakis for giving me the opportunity to work with them on a topic as interesting and innovative as Graph Convolutional Networks for Recommender systems. Also thanks to Xin Xin for being always available to help me with the code as well as with the understanding of some concepts.

Secondly, I would also like to thank Xavier Giró-i-Nieto for introducing me to research and making me realize how much I love it, as well as for guiding me through all this years in the process of becoming an engineer.

Finally, I would like to thank my family and my partner for supporting me since the beginning of my studies until now.

Thanks to all of you.

Revision history and approval record

Revision	Date	Purpose
0	10/09/2019	Document creation
1	01/01/2020	Document revision
2	23/01/2020	Document revision
3	30/01/2020	Document approbation

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Paula Gómez Duran	paula.maria.gomez@alu-etsetb.upc.edu
Xavier Giró i Nieto	xavier.giro@upc.edu
Ioannis Arapakis	ioannis.arapakis@telefonica.com
Alexandros karatzoglou	alexandros.karatzoglou@gmail.com

Written by:		Reviewed and approved by:	
Date	22/01/2020	**Date**	31/01/2020
Name	Paula Gómez Duran	**Name**	Alexandros Karatzoglou
Position	Project Author	**Position**	Project Supervisor

Contents

List of Figures

List of Abbreviations

AI	**A**rtifitial **I**ntelligence
GCN	**G**raph **C**onvolutional **N**etwork
DNN	**D**eep **N**eural **N**etworks
NN	**N**eural **N**etwork
FM	**F**actorization **M**achines
MC	**M**atrix **C**ompletion
RS	**R**ecommender **S**ystem
GPU	**G**raphics **P**rocessing **U**nit

Chapter 1

Introduction

1.1 Motivation

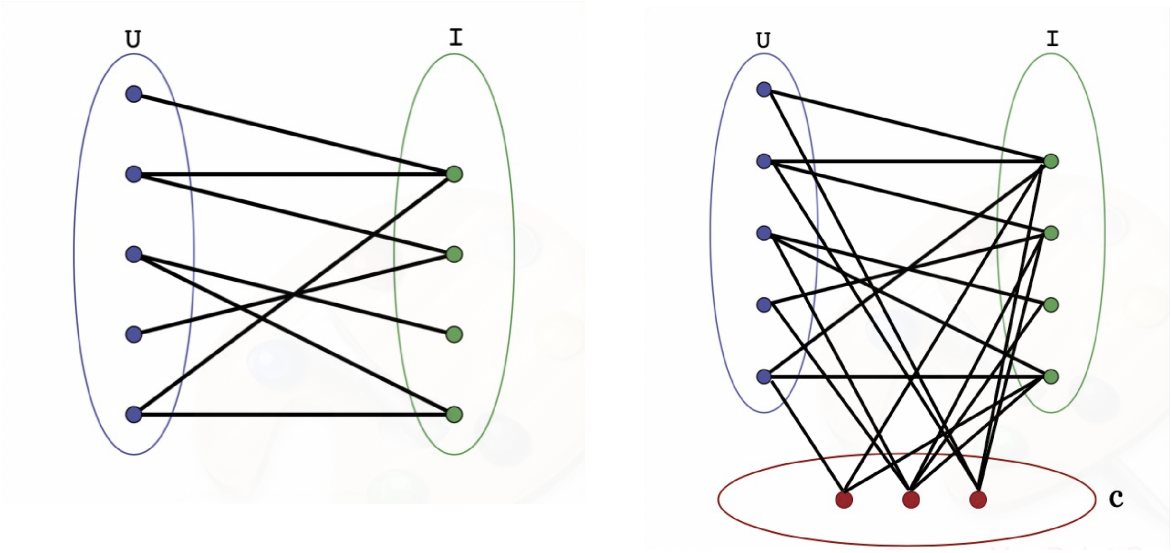
With more multimedia content being available than ever, the need of recommender systems is becoming bigger every day. Nowadays, even more, it is becoming harder and harder to truly find out which are our interests unless a filter is applied when displaying the content. For that reason, **recommender systems** [19] have been recently incorporated into commercial applications thus seeking to predict the *rating* that a user would give to an item, as well as the *preferences*.

Recommendation systems help users find and select items (e.g., books, movies, restaurants) from the huge number available on the web or in other electronic information sources [4]. So given a large set of items and a description of the user's needs, recommender systems present to the user a small set of ranked items that are well suited user's preferences. However, in recent years recommender systems have evolved further to account for the user context when trying to predict those relationships [1]. It is shown that taking context into account (such as daytime, weather, etc.) adds an additional dimension to the user-item data model which can be used in different ways during the whole recommendation process [23] in order to improve it.

In recent years, Graph Convolutional Networks[13], a type of neural network which can naturally integrate node information and topological structure, started to become very popular. This was due to its capability of working directly on graphs and leverage their structural information. These networks, provide great potential to advance social recommendation ([25] [3] [20] [6]) since data in social recommender systems can be represented as user-item social graph that allows to learn latent factors (embedding) of those users and items, which is the key. Simultaneously, Factorization Machines (FM) [17] models have also became very popular due to their capability of being context-aware. But, as they are based on second-order interactions between features embedding, they are not able to capture high-order interactions between features, which is a drawback that we will try to address by capturing them from the embedding level.

In this work, we aim to extend the previous concept of **user-item** bi-parted graph to a multi-parted graph version **user-item-context** as in Figure 1.1, where not only user-item connections are made but also connections between user and item with context. In addition, we aim to demonstrate that by applying Graph Convolutional Networks (GCNs) to the data it is possible to encode in every feature embedding, not just the information of the node itself but also the correlated knowledge from other features. In that way we would be capturing the high-order interaction signals from the embedding level thus making possible to keep the simple structure of FM models (or other predictions methods).

This thesis is structured as follows. A brief introduction of deep learning, related concepts and an overview of GCN, FM and Matrix completion is given in chapter 2. The data processing method and the implemented model are described in chapter 3. The corresponding results are shown in chapter 4 and in chapter 5 we give an estimation of the costs associated to the development of the project. Finally, discussion and future research directions are provided in Chapter 6.



(a) Bi-parted graph of user and item connections.

(b) Multi-parted graph between user, item and context connections.

Figure 1.1: Comparison between bi-parted graph and the multi-parted graph extension. The number of user nodes is just an example as well as the number of items and context nodes respectively.

1.2 Hardware and Software Resources

This project was developed using the *NVIDIA TITAN V* GPU of Telefonica I+D.

The algorithm was implemented in Pytorch[1], using CUDA and cuDNN for fast GPU primitives. This framework was chosen because it is a Python-based deep learning library which is open source. Moreover, it builds applications on top of dynamic graphs, which allows the user to make changes to the network architecture during run-time thus also making the debugging process way easier as the source of error is easily traceable.

The training and inference experiments were performed in *Telefonica research* servers. To use them, ssh command[2] was employed, which is a network protocol that gives users a secure way to access a computer over an unsecured network. In order to let the experiments run for different hours $tmux$ was employed, which is a terminal multiplexer that allows the user to detach to shell sessions (for example, while a model is training) and reattach to them without interrupting the running process.

In order to perform the graphics of the losses and the metrics while training or evaluating, we used Tensorboard: a visualization library originally developed by TensorFlow[3]. It is really useful in order to understand training/evaluation runs, tensors, and graphs.

Code is publicly available at GitHub[4].

[1] `https://www.pytorch.org/`
[2] Also known as Secure Shell or Secure Socket Shell
[3] Framework similar to Pytorch user to develop deep learning applications.
[4] `https://github.com/paulagd/pytorch-GEM`

1.3 Work Plan

This project has followed the attached work plan.

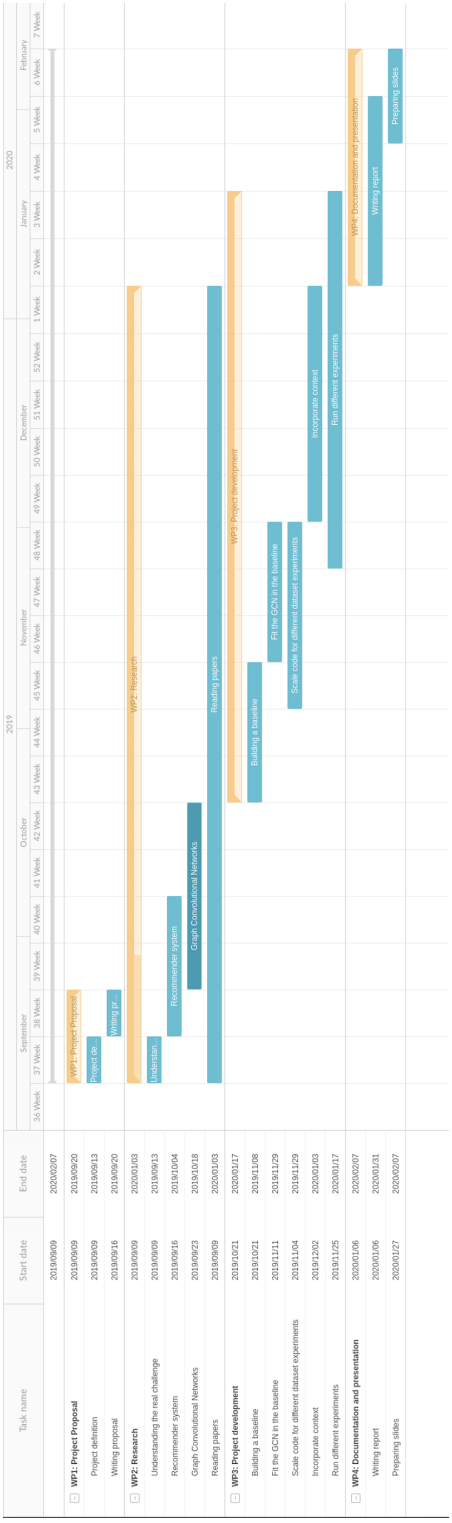


Figure 1.2: Gantt Diagram

Chapter 2

State of the art

2.1 Introduction to Deep Learning

Research in Artificial Intelligence (AI) is motivated by the objective of creating intelligent machines that could help when solving human tasks. Its goal is to be able to do anything that human beings can do, such as understanding the semantics in a news article, recognizing an image, speaking, or even driving a car. Thus, intelligent systems can help us in our daily life and improve it.

Recommendation systems (RS) are quickly becoming the primary way for users to be exposed to the whole digital world through the lens of their experiences, behaviors, preferences, and interests. Having RS could have positive effects on the user experience, thus translating it to higher customer satisfaction and retention. Therefore, taking a film browser as an example, RS could avoid the user flicking through thousands of box sets and movie titles by just presenting a much narrower selection of items that are interesting for the user. In that way, a lot of time would be saved for the user accordingly delivering better user experience.

However, even it is known that machines are faster and more precise than human beings when solving mathematical problems or optimizing tasks, such as finding the best route with GPS navigation, when it comes to the kind of tasks which hold subjective interpretation such as judging how good a person is or choosing which film suits today's mood, it becomes more complicated for machines since it involves human intuition. *Deep Learning* presents a method to solve some of these more intuitive problems by allowing machines to learn from experience and not just from patterns. It does it by simulating human behavior with a structure of neurons similar to biological neural networks, which are called artificial neural networks.

Landing these ideas from a mathematical point of view, the goal of a neural network is to approximate some function $y = f(x;\theta)$ by learning the value of the parameters θ that result in the best function approximation. A neural network can be composed of different layers, and the more layers it has, the more complex the function (model) becomes. Thus, the capacity of the model will increase as well while increasing the complexity of the function[7]. However, the algorithm is not the only important part of deep learning: data plays a crucial role.

An enormous amount of data is necessary for Deep Learning models to learn. In fact, as the amount of training data increases, the easier it becomes for the model to achieve a good performance on unseen data (this is called generalization). Besides, the data needs to be provided with labels, which will be the ones guiding the model in the learning process. So, the first time that the model learns it will initialize randomly the weights thus producing a random output. In that stage, the outputs will be compared with the corresponding labels through an error or *loss* function and the gradients of the error will be back-propagated through the network. Then, the *Stochastic Gradient Descent* algorithm is used tune the required parameters and update the learned weights in order to optimize the loss function. Finally, the whole process is repeated by the model until convergence, where the network will be ready to accomplish the task it has been asked to solve.

There are different neural network architectures depending on the type of data and the task to be solved. We will review some of the literature involved in recommender systems and in graph neural networks in the next sections.

2.2 Recommender systems techniques

Mass customization is becoming more popular than ever and so current recommendation systems such as content-based filtering and collaborative filtering use different information sources to make recommendations[2].

- **Content-based Filtering:** Creates a profile for each user or product to characterize its nature. It makes recommendations based on user preferences for product features.
- **Collaborative Filtering:** Analyzes relationships between users and inter-dependencies among products to identify new user-item associations. It predicts the user's preferences as a linear, weighted combination of other user preferences.

Both methods have limitations but we will use collaborative filtering in this work, we will not be adding *side information*[1] of neither users or items, so we would not be able to create a profile for them. However, collaborative filtering suffers from the *cold start problem*, which is observed when recommending items to a new user who would not have any inter-dependencies among others.

Factorization Machines (FM) models are at the cutting edge of Machine Learning techniques for RS which have different applications to solve the cold-start problem. Besides, they have proven to be an extremely powerful tool with enough expressive capacity to generalize methods such as Matrix Factorization, which is a class of collaborative filtering algorithms explained in the next section.

2.3 Matrix Factorization

Matrix Factorization [14] is a class of collaborative filtering algorithms used in recommender systems (RS). The way it works is by decomposing the user-item interaction matrix into two lower dimensionality rectangular matrices whose dot product will result in the same interaction matrix again.

When a user gives feedback to a certain movie, this collection of feedback can be represented in a matrix form called either **rating matrix** or **adjacency matrix**, in which each row represents each user and each column represents a different movie. This matrix will be sparse, since in a real scenario not everyone will watch every movie and thus a lot of missing values will be present.

The intuition behind using matrix factorization to solve this problem is that there should be some latent features that determine how a user rates an item. The assumption that the number of features would be smaller than the number of users and the number of items is done so

[1]Side information refers to the user/item entity attributes in this case.

that it makes sense to give recommendations. Not assuming that would mean that users are not interested in the items rated by other users, under what it would make no sense to give recommendations.

So, if we define a set U of users, and a set I of items, we know that $\mathbf{A}$ (adjacency matrix) of size $|U| \times |I|$, would be the matrix that contains all the ratings that the users have assigned to the items. Also, we call K to the amount of latent features to discover. Our task, then, is to find two matrices $\mathbf{P}$ ($|U| \times |K|$matrix) and $\mathbf{Q}$ ($|I| \times |K|$ matrix) such that their product approximates $\mathbf{A} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{A}}$

In this way, each row of $\mathbf{P}$ would represent the features of a user and each row of $\mathbf{Q}$ would represent features of an item. To get the prediction of a rating of an item item$_j$ by user$_i$, we can calculate the dot product of the two corresponding feature vectors: $\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$.

So, by finding P and Q matrices which approximate R instead of replicating it, will avoid the predictions of all the unseen ratings to be zero. This will be done by just trying to minimize the errors of the observed user-item pairs instead of having a zero error completely.

	D1	D2	D3	D4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	5	4

(a) Original adjacency matrix A

	D1	D2	D3	D4
U1	4.97	2.98	2.18	0.98
U2	3.97	2.40	1.97	0.99
U3	1.02	0.93	5.32	4.93
U4	1.00	0.85	4.59	3.93
U5	1.36	1.07	4.89	4.12

(b) Reconstructed adjacency matrix $\hat{A}$

Figure 2.1: Comparison between A matrix (left) and its reconstruction $\hat{A}$ (right).

2.4 Graph Convolutional Networks

A Graph Neural Network is also known as a Graph Convolutional Networks (GCN) for having filter parameters which are typically shared over all locations in the graph or over a subset of the graph. GCN performs a convolution on a graph instead of a typical CNN does on images composed by pixels and it relies on the assumption that connected nodes in the graph are likely to share the same label.

Its goal is to learn a function of signals or features on a graph $\mathcal{G}$ and, as defined in [13], GCN is denoted by $\mathcal{G} = (\mathcal{V}; \mathcal{E})$, which takes as input:

- a feature description x_i for every node i summarized in a $N \times D$ feature matrix X, with N being the number of nodes and D the number of input features.
- a representative description of the graph structure in matrix form, which is called adjacency matrix $A \in \mathbb{R}^{NxN}$ and which will have both in rows and in columns the total number of nodes.

and produces a node-level output Z, which is an $N \times k$ feature matrix with k being the number of output features per node.

A multi-layer Graph Convolutional Network (GCN) follow the next layer-wise propagation rule:

$$H^{(l+1)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W(l)) \tag{2.1}$$

where D is the diagonal degree matrix $D_{ii} = \sum_j A_{ij}$ that allows to normalize A by taking the average of neighboring node features; $\hat{A} = A + I_N$ is the adjacency matrix of the unidirected graph $\mathcal{G}$ with added self-connections in order to include the node itself that was not reflected in A interactions; I_N is the identity matrix that we use to add the self-connections; σ is an activation function such as $ReLu(\Delta) = max(0, \Delta)$, and $W^{(l)}$ is the layer-specific trainable weight matrix[2].

2.5 Matrix-completion for recommender systems

In work [3], they propose Graph Convolutional Matrix Completion (GC-MC): a graph-based auto-encoder framework for matrix completion, which builds on recent progress in deep learning on graph-structured data [13].

They consider matrix completion for recommender systems from the point of view of link prediction on graphs. Interaction data such as movie ratings can be represented by a bipartite user-item graph with labeled edges denoting observed ratings. So, the task consists on predicting the value of an unobserved entry in A, being A defined in section 2.4. They define A_{ij} as an observed rating from $user_i$ for $item_j$, while $A_{ij} = 0$ would reflect an unobserved rating.

They propose a graph auto-encoder framework based on differentiable message passing on the bipartite interaction graph but, in this work, we will just leverage the encoder model part, which produces latent features of user and item nodes through a form of message passing[3]. An scheme of GC-MC work with is shown in section 2.5 with the leveraged part for this work highlighted in a red square.

The graph convolutional layer performs local operations that only take the direct neighbors of a node into account, whereby the same transformation is applied across all locations in the graph. This type of local graph convolution can be seen as a form of message passing. In this work, they assign a specific transformation for each rating level, resulting in edge-type specific messages $\mu_{j \to i,r}$ from items j to users i. However, we will unify the rating levels in one, thus denoting any rating a positive interaction or **positive sample** and therefore making binary the adjacency matrix A. The equation is then defined in the following form:

$$\mu_{j \to i} = \frac{1}{c_{ij}} W x_j \tag{2.2}$$

where c_{ij} is a normalization constant, which can either be $|\mathcal{N}(u_i)|$ (left normalization) or $\sqrt{|\mathcal{N}(u_i)||\mathcal{N}(v_j)|}$ (symmetric normalization), with $\mathcal{N}(u_i)$ denoting the set of neighbors of user node i; W is the learned parameter matrix, and x_j is the (initial) feature vector of $item_j$ node.

[2]Note then that the initial layer of H would be $H^{(0)} = X$, being X the features of the user and items and, the last layer of H would be the node-level output $H^{(l)} = Z$

[3]These latent user and item representations are the ones used to reconstruct the rating links through a bilinear decoder but this is the part we are not addressing in this work

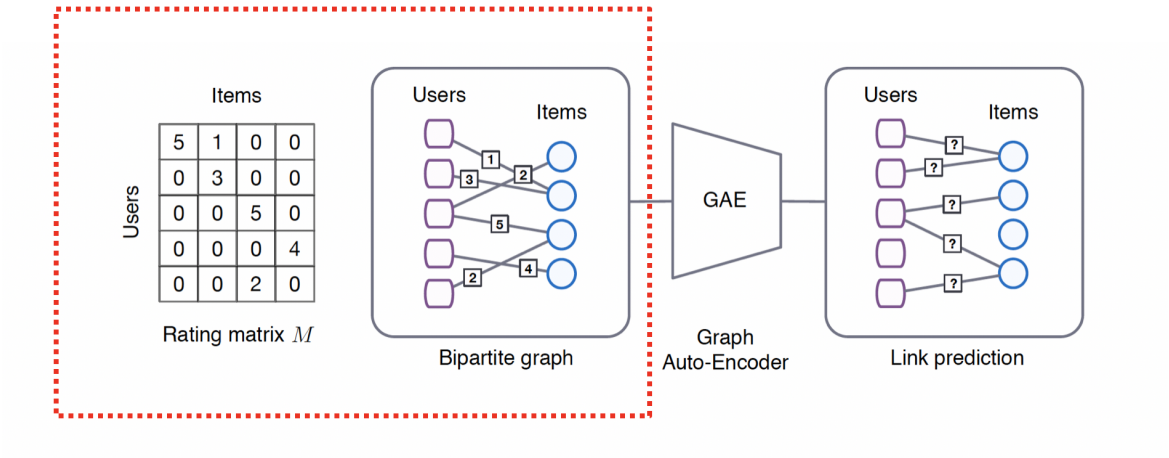


Figure 2.2: Figure taken from [3]. The red square is the part we are leveraging from the whole system. **Left:**Rating matrix M is the same with our A and its entries correspond to user-item interactions (ratings between 1-5) or missing observations (0). **Right:** User-item interaction graph with bipartite structure. Edges correspond to interaction events, numbers on edges denote the rating a user has given to a particular item.

2.6 Factorization Machines model

In [17], they introduce Factorization Machines(FM) as a new model class that extends Matrix Factorization 2.3 to multiple features (such as context). It is also able to estimate interactions even in problems with huge sparsity.

The equation of FM is the following:

$$\hat{y}(x) = w_0 + \sum_{i=i}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_i x_j \langle \boldsymbol{v_i}, \boldsymbol{v_j} \rangle \tag{2.3}$$

where w_0 is the global bias, w_i models the strength of the i-th variable and $\langle \boldsymbol{v_i}, \boldsymbol{v_j} \rangle$ is the dot product of two vectors of size k, which models the interaction between the i-th and j-th variable.

We should note that if the only features involved are two categorical variables (e.g. users and items) then FM is equivalent to the matrix factorization model 2.3.

Chapter 3

Methodology

3.1 Problem definition

The aim of this work is to build a recommender system (RS) which is based not just on user-item interactions but also in context when predicting a recommendation. Therefore, given a past record of items watched (or read, listened, etc.) by a user, we aim to build a RS which helps the user discover items (movies, books, songs, etc.) of their interest. Specifically, given <**userID, itemID, contextID**> occurrence sets, we need to generate a ranked list of preferred items for each user by taking context into account. Note that our work is easily scalable to multiple context features. We model the problem as a binary classification problem, where we learn a function to predict whether a particular user will like a particular item or not. In the end, we will have for every user a set of items which will be ranked in preference order so that the most likely items to be liked by the user would be in the first positions of the ranking.

3.2 Datasets

We needed to look for datasets which would have several ratings R from N_u users on N_i items, being N_u the total number of users and N_i the total number of items. It is constituted a strong requirement the fact that we need an item to be rated by several users in order to allow us building relationships between users and thus predict accurate recommendations.

Moreover, another requirement is to find datasets that provide context, either being the context explicitly written such daytime, weather or location could be; or implicit in the data, such as for example the timestamp of the rating, from where we can extract the last clicked item from a user by sorting them out. We found two datasets that satisfy those requirements which are explained now: Frappè dataset and MovieLens(1M) dataset.

3.2.1 Frappè Dataset

The frappe dataset[1] contains a context-aware app usage log. It consist of 96203 entries by 957 users for 4082 apps used in various contexts (daytime, weekday, isweekend, homework, cost, weather, country, city). For this work, we just make use of 3 context sets; daytime (with 7 different options), country (with 80 different encoded options) and city (with 233 different options). The reason why we opt for three context sets only is because our initial experiments indicated that the performance of the models degrades when we incorporate too much context infromation. This is due to the fact that some context are meaningful for being just irrelevant when the user chooses an item. Thus, adding too much context can act as noise when training the neural network and downgrade the performance.

[1] https://github.com/hexiangnan/neural_factorization_machine/tree/master/data/frappe

3.2.2 MovieLens-1M Dataset

We use the MovieLens 1M dataset[2], which has 1 million ratings from 6000 users on 4000 movies. The ratings are given to us in form of <**userID, itemID, rating, timestamp**> tuples. Each user has a minimum of 20 ratings. For this dataset, we used as context the last clicked item of each user. This way, the context is the previous movie that a user chose before the actual item that we are evaluating.

3.3 Preprocessing, negative sampling and data split

To use the datasets in our model and obtain a good performance in RS task we first must process the data so that it can be fed to the model. We follow the next steps:

First of all, we download the data and split it into training, validation and test data. As explained in section 2.5, for all datasets we will drop the exact value of rating (1,2,3,4,5) and convert it to an implicit scenario where any positive interaction is given the value of 1.

We should note that for problems where the final solution is to perform a ranking of the data, the evaluation task is complex and it needs to be done in a specific way. Therefore, the data split needs to be done according to that and so it **cannot** be done as in classical machine learning problems where you take 70% of the data for training and keep 30% of unseen data for test. For this task, we use the *leave-one-out* method as represented in Figure 3.1 to test the performance of the models. This strategy has been widely used in literature [[10], [26], [9]] and it consists in holding out one transaction of each user for validation (and one for testing) and treat the remaining data as the training set. Since we are training a classifier, we will need both positive and negative samples. The records present in the dataset are counted as positive samples so, we will need to generate negative samples our-self. To do so, we will assume that all entries in the user-item-context interaction matrix (also called adjacency matrix) which have no interactions (represented by 0) are negative samples (a strong, easy to implement assumption).

We define **item$_i$** as one of the positive interacted items (which are 1 in the adjacency matrix). Similarly, we define **item$_j$** as one of the negative interacted items (0 in the adjacency matrix). When doing negative sampling, we fix user and context for a given interaction and just change the $item_i$ by sampling different $item_j$ from the items which the user did not interact with. We will sample as many $item_j$ as the number of negative samples we want per user.

In order to evaluate a user, we need to have at least 3 interactions with different items to be able to put one for training, one for validation and one for testing (due to *leave-one-out* evaluation method that we are following). If we have more than three interactions per user, they will be taken for the training set.

SO, in order to build the training set, we will take the training samples of a given user (all the samples unless the one separated for validation and the one separated for test) and, for each of these positive interactions, we will have <**userID,** $item_i$**, contextID**> plus four[3] interactions like <**userID,** $item_j$**, contextID**>, where $item_j$ will be forced to be different in each of the four samples. As we sample four negative items for every positive interaction in the training

[2] `https://grouplens.org/datasets/movielens`
[3] In our case we decided to sample four negative interactions for a given positive one.

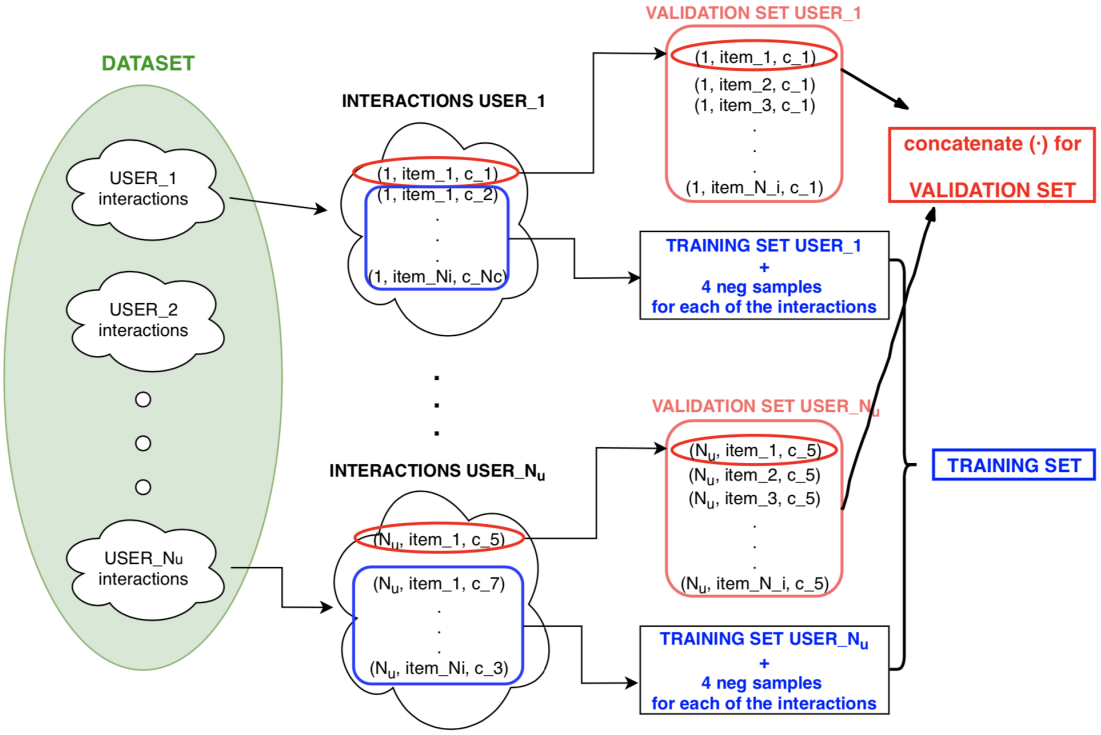


Figure 3.1: Figure showing the *leave-one-out* split strategy. Same procedure is followed for test split. **Green:** Entire dataset containing interactions of all the users. **Red:** Validation set which contains one positive interaction of each user together with negative samples of all the items for each of the user (one per batch). **Blue:** Training set containing all the interactions of every user unless the one saved for validation (and the one for test). Besides, it also contains the 4 corresponding negative samples for each of the users - which are different in each epoch.

data, the final size of the dataset will be multiplied by four. We will be doing batches of 256 and shuffling the data in order to speed up and help the training.

For the validation case, we will just keep the positive interaction (previously separated) and append it with $N_i - 1$ <**userID,** $\boldsymbol{item_j}$**, contextID**> interactions, where the $item_j$ will correspond to every different item of the data. Thus, we will have the positive saved interaction of a user plus the same interaction (letting the user and context fixed) $N_i - 1$ times, in each of the ones we will be changing the $item_i$ for each of the other items of the dataset. This way, we will have a validation set which will have, for a fixed user-context, N_i interactions (one with every item) from which just one will be the positive interaction (ground-truth). This will allow us to evaluate the performance of our model by checking if the positive interaction (the one with $item_i$) is among the top K samples of the ranking ($top@K$) - we would expect that. Furthermore, in order to evaluate correctly we should remove for each item the interactions which have items that were seen in training so that they do not occupy any of the $top@K$ positions when performing the ranking.

To sum up:

On the one hand we will have a training dataset which is composed of positive interactions plus four negative sampled interactions for each of this positive ones.

On the other hand we will have a validation (and the same for test) set which will contain N_i interactions for a fixed user and context, with just one of them being positive - interaction with $item_i$ - and the others being negative - sampled with all different $item_j$ of the dataset, always subtracting the interactions which were already seen in training.

Once we have all data ready to be used, we proceed with the model explanation.

3.4 Building Graph Factorization Machine (GFM)

In this section we will define our model by explaining the modifications that we made in order to extend GCN to context and, finally, generalize it so that it is scalable to fit in any FM structure model such as DFM, NFM or WD. We chose the name of GraphFM in order to point that it is a graph version of the original FM model, which we take as a baseline. However, it is scalable to other RS models such a s Neural FM, Deep FM or Wide Deep models (see chapter 4 for explanation of those models), which they all are different variations of FM model.

So, we started by taking the equation of the FM model 2.3 and modifying it such that high order interactions are captured. Our approach to do so is to make the embedding itself contain those high order signals. We do it by performing the embeddings with a Graph Convolutional Network instead of performing them with a classical embedding layer[4]. In that way, our system will be scalable to any variation of FM model by just changing the way of obtaining the embeddings.

The equation would be written in the following way:

$$\hat{y}(x) = w_0 + \sum_{i=i}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_i x_j \langle \boldsymbol{g}(x_i), \boldsymbol{g}(x_j) \rangle \tag{3.1}$$

where $\boldsymbol{g}(x_i)$ would be the GCN embedding function for feature x_i. The other parameters stay the same as defined in FM section 2.6.

From this perspective, the original FM would be a special case of GFM when $\boldsymbol{g}(x_i)$ is just an embedding table lookup operation (i.e. $\boldsymbol{g}(x_i) = v_i$).

What we expect in this work is that $\boldsymbol{g}(x_i)$ encodes not only the information of x_i but also knowledge from other correlated features. In Figure 3.2 below we illustrate the modifications done in this work. So, as it can be seen, we have just changed the way of performing the embeddings thus leading to more expressive representations for each node. In fact, the structure shown is very important because it allows our GCN module to be scalable to any RS model which used embeddings as a first layer. However, a lot of the work relies on how we build GCN embeddings with $\boldsymbol{g}(x_i)$ and how we extend it to context dimensions That is explained in the next section.

[4]An embedding layer is a representation of an entity (node, word, letter, etc.) with k latent factors. It is more computationally efficient than other encoding ways such as one-hot encoding when using very big datasets. The embeddings get updated during the training process of a deep neural network, and explore entities that are similar to each other in a multi-dimensional space, which can be visualized by using dimensionality reduction techniques like t-SNE [15]

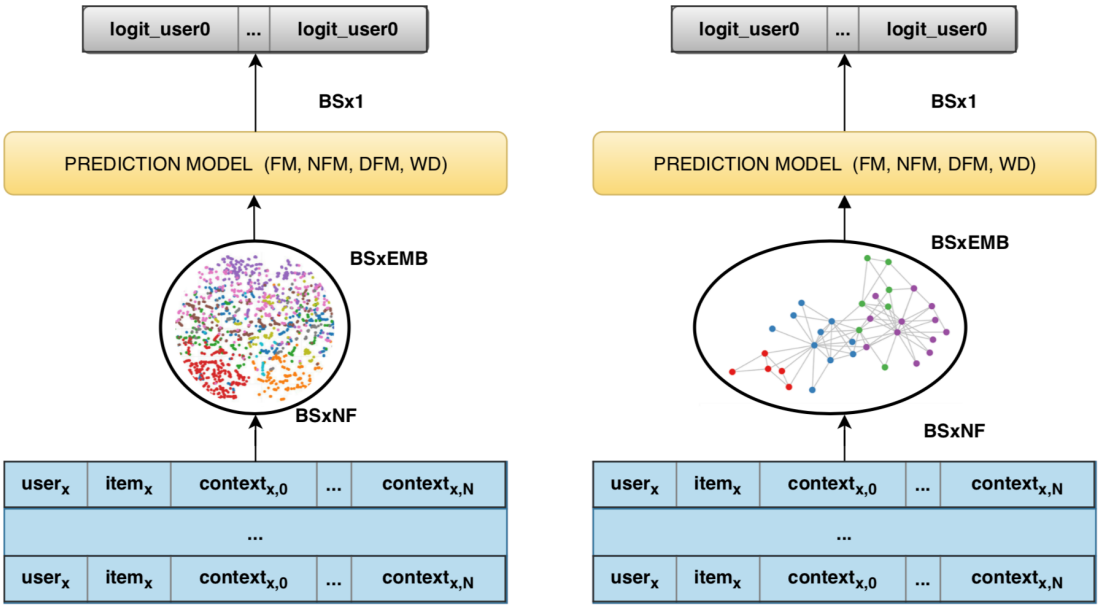


Figure 3.2: Scheme showing how will our model be incorporated in any other FM model by just changing the **embedding layer** (left) by a **GCN layer** (right).

3.4.1 Building GCN embeddings with context extension

Graph convolution networks (GCN)[13] are a natural solution to construct $\boldsymbol{g}(x_i)$ because it performs information propagation in the graph thus making the learnt embeddings contain high-order signals. The way GCN does it is by aggregating information from the local neighbourhoods by pooling some node properties from individual nodes or by computing graph analytics so that some extra data can bring in different information relationships from the network.

3.4.1.1 Graph construction

First of all, we need to construct a graph based on the data. For every transaction, we will have a multi-field categorical feature vector $x \in \mathbf{R}^m$ which will be containing the indexes of the $<$**userID, itemID,** $\boldsymbol{contextID_1, contextID_2, ..., contextID_N}>$ of the interaction. Note that in order to avoid index confusions, we will need to re-index all the nodes in order to provide them a unique identifier, thus being 0 the first user node and $N_u + N_i + N_{Nc}$ the last context node[5].

From the graph, we first build the adjacency matrix A, which always defines the graph itself. An illustration is done 3.3 so that it is very clear how the graph is built. Note that A is built just with the training set data.

[5]Just remember that N_u refers to the total number of users, N_i to the total number of items and N_{Nc} to the total number of context dimensions.

	User_0 .... User_N_u	Item_0 ... Item_N_i	Context_0 ... Context_N_c
User_0 ... User_N_u	0	Interactions user-item	Interactions user-context
Item_0 ... Item_N_i	Interactions user-item	0	Interactions item-context
Context_0 ... Context_N_c	Interactions user-context	Interactions item-context	0

Figure 3.3: Schema providing an intuition of how matrix A is built.

In this case, A would be a symmetric matrix which has both in rows and columns all the $\langle user, item, context_{1..N} \rangle$ node items. Besides, after re-indexing, $item_0$ from the figure above would be $item_{N_u}$ in A, $item_1$ would be $item_{N_u+1}$, and so on with all the nodes until $context_{Nc}$ (from figure above) would be $context_{N_u+N_i+N_{Nc}}$.

Taking as an example an interaction of user-item with two different context such as the weekday and the location, we can illustrate a sub-graph information as in Figure 3.4. We should note that user-items, user-context and item-context are connected with bidirectional links while context-context interactions are not linked.

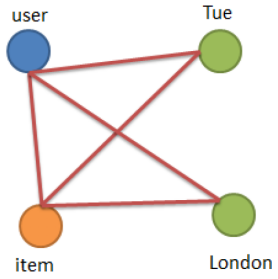


Figure 3.4: Illustration of a sub-graph from the interaction of a user which rated an item with two different context taken into account: weekday (Tuesday) and city (London). 3.4.

3.4.1.2 Context extension

In the original GCN for Matrix Completion paper [3] the local convolution is seen as a form of message passing from the representations of the $items_j$ to the representation of the $user_i$ and vice-versa. The message passing is done as in Equation 2.2 but, in this work, we modify their equation in order to extend it to context, which can have either one or several dimensions (from C_1 to C_N). A possible extension for the equation would be:

$$\mu_{j,C_1,...,C_N \to i} = \frac{1}{c_{ij}}((\boldsymbol{W} x_j)x_{C_1}...)x_{C_N} \tag{3.2}$$

where W is a tensor with as many dimensions as types of graph nodes we have i.e users, items

and context dimensions. However, to avoid dealing with multidimensional arrays, we propose another approach which makes W be a 'big' matrix instead of being tensor. The corresponding equation would be:

$$\mu_{j,C_1,...,C_N \to i} = \frac{1}{c_{ij}}(W_j x_j + W_{C_1} x_{C_1} + W_{C_N} x_{C_N}) \tag{3.3}$$

In this case, W would be a 'big' matrix of $N_{Nc} \times k$ dimensions which will contain in each row the embedding of a given entity such as user, item or context. Having W defined, note that the embedding of feature x_i would be the i-th row in W.

3.5 Training Procedure

We trained the FM baseline and our GFM model by using Binary Cross Entropy, which is a loss function suitable to binary classification tasks. More specifically, we used the Binary Cross Entropy loss with logits from PyTorch[6], which combines a Sigmoid layer and the Binary Cross Entropy Loss in one single operation. It takes advantage of the log-sum-exp trick for numerical stability and returns as an output a scalar.

$$L = \sum_{i=1}^{N} y_i \log(\sigma(p_i)) + (1 - y_i) \log(1 - \sigma(p_i)) \tag{3.4}$$

where y is the label (1 for interacted or 0 non interacted items) and p_i is the predicted probability of the point being interacted for all N points.

Both models were trained using Stochastic Gradient Descent with ADAM optimization [12] and a learning rate of 1e-4. We used Dropout regularization [21] which helps prevent overfitting by dropping some units of the network during training. The specific dropout values used for the experiments are specified in chapter 4.

An epoch is when the entire dataset is passed forward and backward through the network. Every epoch will have the whole data split in batches of 256 samples (in our case we use batch size = 256) and during training, the model will update the weights using Stochastic Gradient Descent with ADAM optimization to generate a better prediction for each step.

In order to find the optimal weights we would be looking for the point where the validation loss is minimum (assuming that the training loss is always decreasing). So, we should stop training the model when the validation loss stops decreasing and starts increasing (early stopping technique[16]). In this work, the model will be trained until the validation loss is going up for more than 10 epochs in order to find out the best performance. Results are shown in chapter 4.

[6] `https://pytorch.org/docs/stable/nn.html?highlight=bcewithlogitsloss#torch.nn.BCEWithLogitsLoss`

3.6 Evaluation

We used Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG) metrics on the validation set for evaluating the recommendation quality of the models.

- **HR@K:** It is a recall-based metric, measuring whether the test item is in the $top@K$ positions of the recommendation list (1 for yes and 0 otherwise).
- **NDCG@K:** It is a measure of ranking quality which gives us information about where in the ranking is our GT sample, this assigning higher scores to the $top-ranked$ items. It is computed by the following equation:

$$1/\log_2(2+index) \tag{3.5}$$

  where index is the position of the item in the list.

As an output, our model will give a score (logit) for each item present in the validation/test set for a given user. The items are sorted in decreasing order of their score, and *top@K* items are given as recommendation. If the GT positive sample (which is only one for each user) is present in this *top@K* list, HR would be one for this user, else it will be zero. The final HR is reported after averaging for all users and an analogous calculation is done for NDCG but using the respective formula for the computation of the metric.

While training, we will be minimizing the cross-entropy loss with logits. However, the real strength of recommender systems lies in giving a ranked list of *top@K* items which a user is most likely to interact with. Thus, metrics like NDCG and HR help in capturing this phenomenon by indicating the quality of our ranked lists.

Chapter 4

Experimental Results

In this section, we describe the experiments we do. They are divided into two sections where we will show, firstly, how adding relevant context helps to RS when giving recommendations and, secondly, the difference in terms of performance between the baselines and their corresponding graph-versions. Finally, we will add a section where we will discuss the experimental results conclusions.

Before showing the results, we give an insight of the baselines models with the ones our model is compared to in order to briefly explain their differences.

Baselines

We implemented all models using Pytorch. We compare the performance of the following baseline vs their performance when incorporating our model (GFM) for capturing embeddings (graph version). The baselines are:

- **Factorization Machines (FM)** [17]: As described in 2.3, this is the original FM. We rewrote the official implementation[1] to the Pytorch version.
- **Neural Factorization Machines (NFM)** [9]: It is a strong baseline which feeds FM second-order interaction terms into a multi-layer perceptron (MLP) to learn nonlinear and high-order interaction signals.
- **Deep Factorization Machines (DFM)** [8]: It ensembles the original FM but adding a third term which are the embeddings fed into a multi-layer perceptron (MLP).
- **Wide&Deep (WD)** [5]: Its similar to DFM but removing the FM term of the equation. So, it is just a linear regression plus the MLP of the embedding features.

4.1 Non-context vs context in recommendation systems

In this section, we perform different experiments in order to state that adding context to RS helps the model to give better recommendations. Our approach is to train each of the baselines *with* and *without context* (green and red rows from 4.1 respectively) information and then evaluate them for the *top@K* items, where $K = \{10, 20\}$ mean that we should seek for our GT sample being in the 10 or 20 top items for each of the metrics **HR** and **NDCG**. In the next figures, [4.1 and 4.2], it can be observed that, in both baselines, adding context helps a lot to the model when predicting recommendations. It has to be note that for each of the datasets the results are different and this is due to the different number of users and items that they have, which will affect to the performance when averaging the metrics.

[1] http://www.libfm.org

		Frappè dataset			
MODEL	**CONTEXT**	**HR@10**	**NDCG@10**	**HR@20**	**NDCG@20**
FM	✗	0.5486	0.1898	0.6590	0.2060
	✓	**0.7125**	**0.5991**	**0.7447**	**0.6072**
NFM	✗	0.6197	0.3924	0.7382	0.4225
	✓	**0.7410**	**0.6292**	**0.7695**	**0.6364**
DFM	✗	0.6035	0.3765	0.7322	0.4092
	✓	**0.7361**	**0.6097**	**0.7670**	**0.6176**
WD	✗	0.4857	0.2768	0.5898	0.3038
	✓	**0.4931**	**0.2955**	**0.5998**	**0.3198**

Figure 4.1: Comparison between context and non-context data for the different baselines on the Frappe dataset.

		ML-1M dataset			
MODEL	**CONTEXT**	**HR@10**	**NDCG@10**	**HR@20**	**NDCG@20**
FM	✗	0.0998	0.0404	0.1268	0.0444
	✓	**0.1005**	**0.0541**	**0.1325**	**0.0599**
NFM	✗	0.1070	0.0247	0.1510	0.0417
	✓	**0.1187**	**0.0657**	**0.1634**	**0.0771**
DFM	✗	0.0702	0.0111	0.1076	0.0208
	✓	**0.0784**	**0.0402**	**0.1188**	**0.0454**
WD	✗	0.0705	0.0429	0.1159	0.0545
	✓	**0.1537**	**0.0740**	**0.2341**	**0.0986**

Figure 4.2: Comparison between context and non-context data for the different baselines on the ML-1M dataset.

4.2 Baselines vs their Graph-version performance

In this section we performed different experiments for each of the datasets (Frappè and ML-1M) in order to see whether the graph version of a baseline model is able to outperform it.

The hyperparameters we used are a learning rate of 0.001 with Adam optimizer, a batch size of 256 and an embedding dimension of 64 for all the models. We run all the experiments for 300 epochs until convergence of the metrics.The time elapsed and the dropout regularization probability is written in the table results for each of the training experiments (all with GPU including training and evaluation every 10 epochs).

On the one hand, for Frappè dataset we will train for each of the baselines the original model (i.e. FM, NFM, DFM, WD) and their corresponding graph version (i.e. GFM, GNFM, GDFM, GWD). However, the graph version of a model does not always outperform the baseline for both metrics, specially in cases where models are already too expressive (like NDM and DFM). For that reason we propose two more experiments which consist in pre-training the embeddings of each baseline and then use them as input features for the GCN layer (X) or as an initialization of its learned weights (W). Thus, instead of taking the identity matrix as feature matrix X which is fed as an $H(0)$ in 2.1, we will take the pre-trained embeddings of each baseline (i.e. the pre-trained embeddings of FM will be used as X when training GFM) or, instead of initializing randomly the W of the same equation, we will initialize it with the pre-trained embeddings (transfer-learning [22]).

Then, we will run four different experiments for each of the baselines: $\mathbf{1}$.Original model or baseline (i.e. FM), $\mathbf{2}$.Graph version (i.e. GFM), $\mathbf{3}$.Pre-trained embeddings as input features X (i.e. pre-GFM-I), $\mathbf{4}$.Pre-trained embeddings as initialization for W(i.e. pre-GFM-W). Results are shown in figure 4.3 in order to compare the metrics of the four different trained models for each of the baselines.

On the other hand, for ML-1M dataset, we have the results shown in figure 4.4. We should note that the results of the two different datasets are written in different tables because they can not be compared, as they each have different number of users, of items and of context as well as different number of total interactions. For that reason, a score of *HR@10=0.1* in ML-1M can mean a good performance while the same score metric in Frappe dataset would mean having disastrous performance of the model. Thus, as in ML-1M the graph-version of the models already outperform each of the baselines respectively, for this dataset we will not be doing the pre-trained embedding experiments.

FRAPPE dataset

In the figure 4.3, we can observe that a consistent outperforming of the baselines is just achieved for FM and WD models when capturing the high-order signal interactions from the embedding level (in FM we need to modify the GFM model by initializing W to the pre-trained FM embeddings while in WD its enough by just training GWD model from scratch). However, in more expressive models like NFM or DFM, we could just achieve the improvement of either the ranking (HR metric in pre-trained W experiments) or of its quality (NDCG metric in pre-trained I experiments) but not from both at the same time. We propose many research lines in the Future Work section of chapter 6 in order to solve that problem.

	FRAPPE (DAYTIME + COUNTRY + CITY context)					
	TOP-10		TOP-20		Dropout	Time elapsed
	HR	NDCG	HR	NDCG		
FM	0.7125	0.5991	0.7447	0.6072	0	03: 12: 17
GFM	**0.7311**	0.5809	**0.7559**	0.5873	0	05: 45: 40
PRE-GFM-I	0.7150	**0.6316**	0.7385	**0.6370**	0 - 0	04: 09: 42
PRE-GFM-W	**0.7373**	**0.6249**	**0.7546**	**0.6292**	0 - 0	03: 45: 09
NFM	0.7410	0.6292	0.7695	0.6364	0.2 - 0	03: 00: 13
GNFM	**0.7447**	0.6182	**0.7757**	0.6261	0.2 - 0	05: 00: 43
PRE-GNFM-I	0.7100	**0.6287**	0.7336	**0.6334**	0.2 - 0	04:00:45
PRE-GNFM-W	**0.7485**	0.5881	**0.7720**	0.5967	0.2 - 0	02: 56: 20
DFM	0.7361	0.6097	0.7670	0.6176	0.2 - 0	03: 16: 13
GDFM	0.7299	0.6030	0.7497	0.6103	0.2 - 0	04: 39: 36
PRE-GDFM-I	0.7100	**0.6287**	0.7336	**0.6334**	0.2 - 0	03: 49: 50
PRE-GDFM-W	**0.7410**	0.5571	**0.7757**	0.5659	0.2 - 0	02: 56: 20
WD	0.4931	0.2955	0.5998	0.3198	0.2	03: 07: 08
GWD	**0.5266**	**0.2955**	**0.6444**	**0.3257**	0.2 - 0	04: 14: 52
PRE-GWD-I	**0.5291**	**0.3059**	**0.6382**	**0.3337**	0.2 - 0	02: 43: 22
PRE-GWD-W	**0.5056**	**0.2885**	**0.6344**	**0.3211**	0.2 - 0	04: 05: 23

Figure 4.3: Results of **1.** original model, **2.** graph version, **3.**pre-trained embeddings as input features X, **4.**pre-trained embeddings as initialization of W. Results computed for each of the baselines (FM, NFM, DFM, WD) highlighted in red on the Frappe dataset.

ML-1M dataset

In Figure 4.4, we can observe that a consistent outperforming of the baselines is achieved for all the models when capturing the high-order signal interactions from the embedding level.

Besides, if we focus on the *Time elapsed* row, we can note that the time that the Graph version models take for training and evaluating the model is much bigger than the time consumed for training the baselines. However, we state that the graph version of the models take a lot less time to converge than the baselines, needing the baseline models over 250 epochs to converge while the graph-version models converge in epoch 75 approximately.

	ML-1M (Last-clicked-item as context)					
	TOP-10		TOP-20		dropout	Time elapsed
	HR	NDCG	HR	NDCG		
FM	0.1005	0.0541	0.1325	0.0599	0	04: 17: 02
GFM	**0.1411**	**0.0815**	**0.2010**	**0.0965**	0 - 0	09: 57: 25
NFM	0.1187	0.0657	0.1634	0.0771	0.2	04: 26: 33
GNFM	**0.1547**	**0.0860**	**0.2187**	**0.1022**	0.2 - 0	09: 51: 18
DFM	0.0784	0.0402	0.1188	0.0454	0.2	04: 55: 47
GDFM	**0.1552**	**0.0894**	**0.2129**	**0.1039**	0.2 - 0	09: 51: 18
WD	0.1537	0.0740	0.2341	0.0986	0.2	06: 17: 24
GWD	**0.1881**	**0.1116**	**0.2481**	**0.1267**	0.2 - 0	09: 57: 25

Figure 4.4: Results of **1.** original model, **2.** graph version. Results computed for each of the baselines (FM, NFM, DFM, WD) in the ML-1M dataset.

4.3 Discussion

FM models are the reference models for being an effective solution for context-aware recommender systems. However, they are insufficient to capture high-order and nonlinear interaction signals. Because of that, several recent efforts have enhanced FM with neural networks like in NFM and in DFM models, which we should assume as strong baselines. They are more expressive than FM, which can actually be seen as a special case of NFM but without hidden layers or DFM without the MLP term.

Therefore, we realise that it made no sense to compare Graph Factorization Machines (our model) with NFM or DFM models but instead compare each baseline with its corresponding graph version: comparing FM vs GFM; NFM with GNFM; DFM with GDFM; and WD with GWD. At that stage, we achieved to outperform the baselines for both datasets when evaluating HR and NDCG metrics. However, for Frappè dataset, we couldn't achieve to outperform NFM or DFM in both metrics yet. Therefore, we think that we should analyse as future work if its due to the difference in the amount of data (Frappè dataset is smaller) or due to the context relevance that we are using.

Chapter 5

Budget

The hardware resources needed for the project were a Macintosh laptop and a NVIDIA GPU. The GPU was used during 5 months for the development of the model, which adds to 3.360 hours of computation. We compute the computation cost based on Amazon Web Services (AWS) rates[1] for *p2.xlarge* instances with one NVIDIA K80. Regarding software, we used *Pycharm Professional* which licence cost 89 €.

The main costs of this projects comes from the salary of the researches and the time spent in it. The team for the development of this thesis is formed by two senior engineers as the advisors and myself as a junior engineer. The length of the project was 22 weeks, as presented in the Gantt diagram. Assuming a commitment of 30 weekly hours by the Junior engineer (myself) plus an average of 1h per week from each advisor on weekly meetings, the complete costs for the project are the following:

	Amount	**Cost/hour**	**Time**	**Total**
GPU *p2.xlarge*	1	0,90 €	3.364h	3.024 €
Pycharm Professional	1	-	-	89 €
Junior engineer	1	10,00 €	600h	6.000 €
Senior engineer	2	30,00 €	22h	1.320 €
Other equipment	-	-	-	3.000 €
			Total	13.433 €

Table 5.1: Cost of the project. *Other equipment* includes campus services and employed laptop.

[1] https://aws.amazon.com/ec2/instance-types/p2/?nc1=h_ls

Chapter 6

Conclusion and Future Work

The main goal of our work was to capture the high-order signal interactions from the embedding level by using a Graph Convolutional Networks (GCN) extended to context. To do so, we had three main contributions in this work: demonstrate that adding context data helps the models to achieve more accurate performance; modify GCN in order to extend it to context; and integrate GCN in FM models thus using them to capture those high-order signal interactions from the embedding level.

The first contribution is clearly achieved in section 4.1, where we present for each of the baselines the results when training the models *with* and *without* context. We can state that relevant context clearly helps recommender systems when giving recommendations. Regarding the second contribution, we modified GCN by taking as an approach the extension of matrix-completion algorithm[3] to multiple context. Finally, in order to achieve the third contribution, we integrated the GCN extended algorithm into FM in order to use them as the new layer for capturing the embeddings.

Thus, we presented an effective model for capturing high-order signal interactions in context-aware recommender systems, which we called Graph Factorization Machines (GFM) and which is based on FM structure in order to be scalable to any of the FM variation models such as NFM, DFM or WD. Therefore, at the same time we show that the novel idea of capturing high-order interactions from the embedding level through a graph-convolutional layer is scalable to any other model by just changing the way of capturing embeddings.

Regarding the results, we state that using graph convolutions helps to encode better information and thus give better recommendations. This is due to the fact that graph convolutions encode in each node of the graph (feature embeddings), not just the information of the node itself but also the information of correlated nodes thus leveraging the graph structure which is created from the data. Reaching this point though has not been a straight line. Many inconveniences aroused when building the model, specifically when tuning the hyperparameters and designing how to pre-train the GCN model and improve its performance. In fact, we do not have a consistent solution which works for any model and any dataset. However, we can say overall that we came up with an structure which allows to capture high-order signal interactions and which allows to outperform the baselines in most of the cases.

As a future work, we plan to change the BCE with logits loss for the Bayesian Personalized Ranking (BPR) loss [18], which is specific for ranking prediction problems. Besides, we plan to extend the work by adding more datasets which will allow us to extract data patterns and end up with a solution which is consistent for integrating it with any model and any dataset. Last but not least, we propose to find a way for evaluating the results not just in a quantitative way but in a qualitative way as well.

Code is publicly available at `https://github.com/paulagd/pytorch-GEM`.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[2] Asim Ansari, Skander Essegaier, and Rajeev Kohli. Internet recommendation systems, 2000.

[3] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[4] Robin Burke. The wasabi personal shopper: a case-based recommender system. In *AAAI/IAAI*, pages 844–849, 1999.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10. ACM, 2016.

[6] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pages 417–426. ACM, 2019.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[8] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[9] Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 355–364. ACM, 2017.

[10] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2354–2366, 2018.

[11] Nikhil Ketkar. Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer, 2017.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[14] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[15] Alexander Platzer. Visualization of snps with t-sne. *PloS one*, 8(2):e56883, 2013.

[16] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[17] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[18] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[19] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–59, 1997.

[20] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[22] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010.

[23] Wolfgang Woerndl and Johann Schlichter. Introducing context into recommender systems. 01 2007.

[24] Xin Xin, Bo Chen, Xiangnan He, Dong Wang, Yue Ding, and Joemon Jose. Cfm: convolutional factorization machines for context-aware recommendation. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3926–3932. AAAI Press, 2019.

[25] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983. ACM, 2018.

[26] Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 227–236, 2016.