Temporal Relational Reasoning with Neural Networks

Samuel Gomes Fadel

Abstract

The study of relationships between objects and their properties is a common pattern in many areas of science. However, manually extracting and reasoning about this information quickly become a daunting undertaking once the number of connections and entities most of these phenomena involve is taken into account. Artificial neural networks (NNs) have recently been a central piece of many state-of-the-art machine learning methods, aimed exactly at dealing with large volumes of data. In particular, there has been an increasing interest in applying them to graph domains. Still, one important fact mostly unaccounted for is that almost all large real-world graphs evolve over time. In this research, we are interested in addressing the problem of learning from dynamic graphs, that is, graphs that change over time. Our hypothesis is that if a phenomenon can be represented by a dynamic graph, a model that learns from it will have better predictive performance on that domain when compared to a model that learns from a regular graph representation of that phenomenon. With this in mind, we aim to design neural network architectures exploiting relational inductive biases so that they learn from dynamic graphs, with particular focus on recommender systems and sports analytics. We propose the following steps to attain this objective. First, we intend to investigate how the many NN components for graphs and for sequences can be combined to handle dynamic graph data. This investigation will be useful as a first step for tackling for both tasks, as it aims to highlight the most promising approaches by identifying their limitations and advantages. Next, we approach the problem of recommendation using dynamic graphs as a temporal link prediction task. We will also explore how dynamic graphs and NNs can be employed to solve problems in sports analytics, specifically in the case of soccer, by modeling players as graph nodes and their interactions as edges. As a complex system of interactions, we anticipate that this leads to challenging problems not explored before, where the approaches proposed so far can be useful. Finally, we present some preliminary experiments as evidence supporting our motivation for this research. We show how relational reasoning can be used in a problem that, at first glance, does not seem to require relational reasoning. The competitive performance attained highlights how relational reasoning can be used to approach problems with a new perspective, even when relational information is not explicit. One can still leverage it to solve problems where relationships between objects have a role. Then, we show how one can use temporal information associated with a user-item ratings graph to attain competitive performance when compared to a state-of-the-art method that uses a similar framework for recommendation tasks. This experiment highlights a promising research direction for models that can understand how users and items are related over time, so that better recommendations can be provided. In addition, results suggest that dynamic graphs indeed have the potential to provide more detailed representations of the underlying phenomenon.

Contents

Chapter 1

Introduction

The study of relationships between objects and their properties is a common pattern in many areas of science [48, 40]. Graphs[1] are a powerful and expressive mathematical tool used to represent relationships between objects. Studies on developmental disorders [63], gene expression analysis [31], and rumor propagation in social networks [62] are some of the many examples where the analysis of graph data was fundamental to comprehending the phenomena in question. This fact strongly suggests that their structure contains significant information about the systems and entities they represent. However, manually extracting and reasoning about this information quickly becomes a daunting undertaking once the number of connections and entities most of these phenomena involve is taken into account.

Machine learning is concerned with methods and algorithms able to use data as a source of experience for improving their performance on some task related to that data [45]. These are useful for building machines that can predict future outcomes of phenomena and automate decision making, as they infer rules "learned" from data. The unprecedented rates of data generation and collection we experience today have stimulated a large amount of research on machine learning methods, specifically for taking advantage of this fact. In particular, artificial neural networks (NNs) have recently been a central piece of many state-of-the-art machine learning methods [37, 56]. Researchers using NNs have achieved remarkable performance in a diverse range of tasks, such as image classification [60, 21], speech generation [49], and automated control [46].

This recent success and widespread adoption are usually attributed to some factors [56]. First, an increased availability of large datasets of labeled data. Second, the use of commodity hardware, such as GPUs, to accelerate training and inference on those large datasets. This renewed interest fostered a large number of advances in NNs, with new architectures, learning (optimization) insights, and domains of application. Third, the hierarchical structure of NNs enables the model to extract features directly from raw input signals [16]. Specifically, NNs are mostly end-to-end, that is, they learn from data without substantial data preprocessing and feature engineering, encompassing the entire pipeline from input to output. Because of this, models trained on some task learn intermediate representations that can be effective employed in another task, making them useful not only for the original task they were trained on, but also other tasks requiring

[1] Due to the research terminology in complex networks literature, and the fact that artificial neural networks (NNs) are also networks, using the word 'network' explicitly may lead to confusion. Henceforth, we refrain from using 'network' to refer to either of them. We will, whenever possible, use 'graph' when referring to the data in our domain of interest and 'NN' (or whichever acronym is appropriate for the context) when referring to artificial neural networks from the machine learning literature.

rich representations of similar data [64].

Due to their many favorable characteristics, NNs have been employed on many domains, such as images, video, and text. This leads to the question of whether they can also be applied to other, non-Euclidean domains, such as graphs. As already mentioned, graphs enable rich representation of phenomena, but they are very different in nature to data on which NNs have been mostly employed. Despite this, recent results [11, 58, 2, 13, 32, 10, 55, 53] support this research direction with increasingly more successful NNs on graph domains. Their successes are mostly attributed to the type of relational inductive bias present in the proposed architectures [1]. A relational inductive bias, loosely speaking, is an assumption used when designing such architectures, imposing constraints on how entities relate to each other in a learning process. Specifically for relational domains, the most important relational inductive bias is where entities and their relations are explicitly represented, as is the case of a graph.

Still, one important fact most of those do not take into account is that almost all large real-world graphs evolve over time [38]. While understanding how entities relate to each other is the basis of relational reasoning, we argue that reasoning about how entities are changed by those relations and even how relations themselves might change over time, is an important aspect to be incorporated into existing methods. This is, evidently, applicable to domains where time naturally arises as an influence. Examples of those include systems of physically interacting bodies [2, 53], recommender systems [3, 47], and sports analytics involving direct analysis of player interaction [50, 8]. As a consequence, we believe there are many opportunities for tackling problems represented as graphs that change over time.

1.1 Objectives

In this research, we are interested in addressing the problem of learning from dynamic graphs using NNs. Specifically, we consider *dynamic graphs* as graphs where either signals (values) associated with nodes (entities) or edges (relationships) can change over time, nodes and edges can be added or removed over time, or both. Our main hypothesis for addressing this problem is:

> *If a phenomenon can be represented by a dynamic graph, a model that learns from it will have better predictive performance on that domain when compared to a model that learns from a regular graph representation of that phenomenon.*

With this hypothesis, we state our objectives as:

> *To design neural network architectures exploiting relational inductive biases so that they learn from dynamic graphs, with particular focus on recommender systems and sports analytics.*

1.2 Challenges

Challenges faced by this project essentially stem from two factors: one is the graph domain, and another is the temporal domain. In the graph domain, we have scalability issues with some methods for large graphs [32]. Essentially, the training procedure of many of them require that the entire graph is processed every single time the NN has

its parameters updated. Furthermore, many of them also have a number of parameters directly proportional to either the number of edges or number of nodes in the graph. More recent advances, however, have shown that one can scale to graphs with billions of nodes and edges by simplifying some operations [66].

In the temporal domain, since there are very few solutions proposed so far, we expect this to be the most challenging aspect. One of the main reasons for that is related to changes over time regarding the addition and removal of nodes and edges in the graph. In these scenarios, assumptions of existing methods for graph domains do not hold, further limiting the options of extending them to dynamic graphs. Additionally, training NNs specifically tailored for sequential data, or data that changes over time, is inherently more difficult than training regular NNs, which are currently used for non-dynamic graphs. Thus, training the devised models can be more difficult than anticipated.

1.3 Structure of the document

The rest of the document is organized as follows:

- Chapter 2 introduces the technical background and discusses previous work related to the topic of this research;
- Chapter 3 presents our methodology for validating our hypothesis and attaining our objectives;
- Chapter 4 describes some preliminary results: (1) experiments using a NN designed to process word relations in order to make use of tags associated with multimedia items, followed by (2) experiments illustrating a promising approach to recommendation systems which use the order in which users rate items in order to produce recommendations;
- Chapter 5 details our schedule of activities for this research.

Chapter 2

Background and related work

Here we present and discuss concepts and results from previous work that are relevant to the research topic and its objectives. We start with the fundamental concepts of machine learning and how to learn from data. Following that, we introduce artificial neural networks (NNs), a family of machine learning methods which recently attained impressive results in a range of increasingly complex learning scenarios. Specifically, we present the simplest NN architecture and developments upon it that are used today in state-of-the-art models for a variety of learning tasks, particularly those pertaining this research. Finally, we summarize recent work at the intersection of both topics, highlighting important insights, results, and research opportunities.

2.1 Learning from data

Following the terminology adopted by previous authors [41], *learning* refers to inferring rules by observing examples. The central question to this process is how a machine, namely a computer, can "learn" to perform some task by following learning algorithms. The machine is shown some examples related to this task and then its goal is to infer a general rule which can either explain the examples it has seen, generalize to new (unseen) examples, or both. The rule or rules inferred by a learning algorithm from a certain set of examples is said to be a *model* and this set of examples is called a *learning set*, *training data*, or simply *dataset*.

For a more precise formulation, suppose our training data is a set of N objects, each represented as an m-dimensional vector $\mathbf{x} = (x_1, x_2, \ldots, x_m)$, where $x_j \in \mathcal{X}_j$ corresponds to the value of a random variable X_j $(j = 1, 2, \ldots, m)$. The set $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_m$ denotes our *input space*. This set contains all possible inputs to the learning algorithm. Some tasks also require that an output is produced by the machine. We define the *output space* in a similar way, where $y \in \mathcal{Y}$ denotes the value of the output random variable Y. The output space can also have more than one dimension and expressing this is analogous to the input space. Variables X_j are often referred to as *features* or *dimensions*, $\mathbf{x}$ as *sample* or *instance*, and Y as *label* or *target*.

Most machine learning tasks can be categorized as supervised, unsupervised, and reinforcement learning [7]. The main differences are at how the set of examples is presented to the algorithm, which in turn determines the way they are evaluated and how training is performed. In *supervised learning*, examples provided contain both the input and expected output. In other words, an example is a pair $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$. The algorithm then uses the expected output to adjust its inferred rule accordingly, as if being "supervised" by a

human. The two main tasks in supervised learning are *classification*, where $\mathcal{Y}$ is a finite set whose elements have no natural order, and *regression*, where we usually have $\mathcal{Y} \subseteq \mathbb{R}$.

Unsupervised learning comprises tasks where the data do not contain labels, thus the goal is to use the learning algorithm to discover some structure or property in the data. Common tasks include *clustering*, where we have some objects $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and the aim is to assign them to groups in such that elements belonging to the same group share some similarities.

In *reinforcement learning*, the machine is viewed as an agent immersed in a virtual environment. Given the current state of this environment, its task is to perform actions such that a numerical reward is maximized [59]. Note that, in contrast to the previous settings, here the outputs (actions) influence future inputs, which consequently leads to one of the fundamental challenges of reinforcement learning: the trade-off between exploration and exploitation. This trade-off refers to balancing the use of known actions that maximize some reward (exploitation) while still experimenting with new actions that might potentially lead to even better rewards (exploration).

2.2 Artificial neural networks

An artificial neural network (NN) is a machine learning model structured as a network (or graph) of units called *neurons*, with weighted, directed connections [56]. Let l denote the length of the path between a certain neuron j and the input of the NN. The neuron of a NN is mathematically expressed as a function $y(\mathbf{x}; \boldsymbol{\theta}_j^{(l)})$ of its inputs $\mathbf{x} = (x_1, x_2, \ldots, x_m) \in \mathbb{R}^m$ and parameters $\boldsymbol{\theta}_j^{(l)} = (\theta_{j0}^{(l)}, \theta_{j1}^{(l)}, \ldots, \theta_{jm}^{(l)}) \in \mathbb{R}^{m+1}$. In order to avoid clutter, we write $y(\mathbf{x}; \boldsymbol{\theta}_j^{(l)})$ as $y_j^{(l)}(\mathbf{x})$. A neuron is defined as

$$z_j^{(l)}(\mathbf{x}) = \theta_{j0}^{(l)} + \sum_{i=1}^{m} \theta_{ji}^{(l)} x_i, \tag{2.1}$$

$$y_j^{(l)}(\mathbf{x}) = \xi\left(z_j^{(l)}(\mathbf{x})\right), \tag{2.2}$$

where $\theta_{ji}^{(l)}$ represents the connection weight between neuron j and its i-th input connection, $\theta_{j0}^{(l)}$ is a bias term, $\xi : \mathbb{R} \to \mathbb{R}$ denotes a differentiable, nonlinear function called the *activation function*. Figure 2.1 illustrates a neuron schematically. We refer to all neurons which are l connections away from the input variables as the l-th *layer* of a NN. We denote the number of neurons in layer l as k_l and $\boldsymbol{\theta}^{(l)} = (\boldsymbol{\theta}_1^{(l)}, \boldsymbol{\theta}_2^{(l)}, \ldots, \boldsymbol{\theta}_{k_l}^{(l)})$. The output of the l-th layer is

$$\mathbf{y}(\mathbf{x}; \boldsymbol{\theta}^{(l)}) = \mathbf{y}^{(l)}(\mathbf{x}) = (y_1^{(l)}(\mathbf{x}), y_2^{(l)}(\mathbf{x}), \ldots, y_{k_l}^{(l)}(\mathbf{x})). \tag{2.3}$$

Layers such as this one are often referred to as linear layers. When each neuron of a layer l is connected to each neuron of layer $l-1$ it is a dense or fully-connected layer. For a NN with L layers, the 0^{th} layer is called the *input* layer, with $\mathbf{y}^{(0)}(\mathbf{x}) = \mathbf{x}$, while the L-th layer is called the *output* layer.

By "stacking" L fully-connected layers, such that the output of the l-th layer is the input of the $(l+1)$-th layer, we construct a *feedforward neural network* [56]. This model, written as a function f with parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(L)})$ becomes

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{y}^{(L)} \circ \ldots \circ \mathbf{y}^{(2)} \circ \mathbf{y}^{(1)}(\mathbf{x}). \tag{2.4}$$

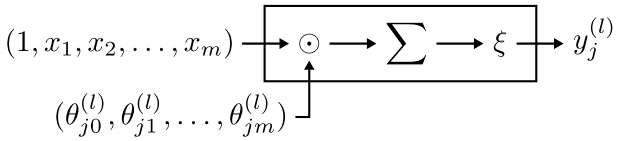


Figure 2.1: A neuron used in linear (or fully-connected) layers (2.2). The $\odot$ symbol denotes the Hadamard (i.e., point-wise) product.

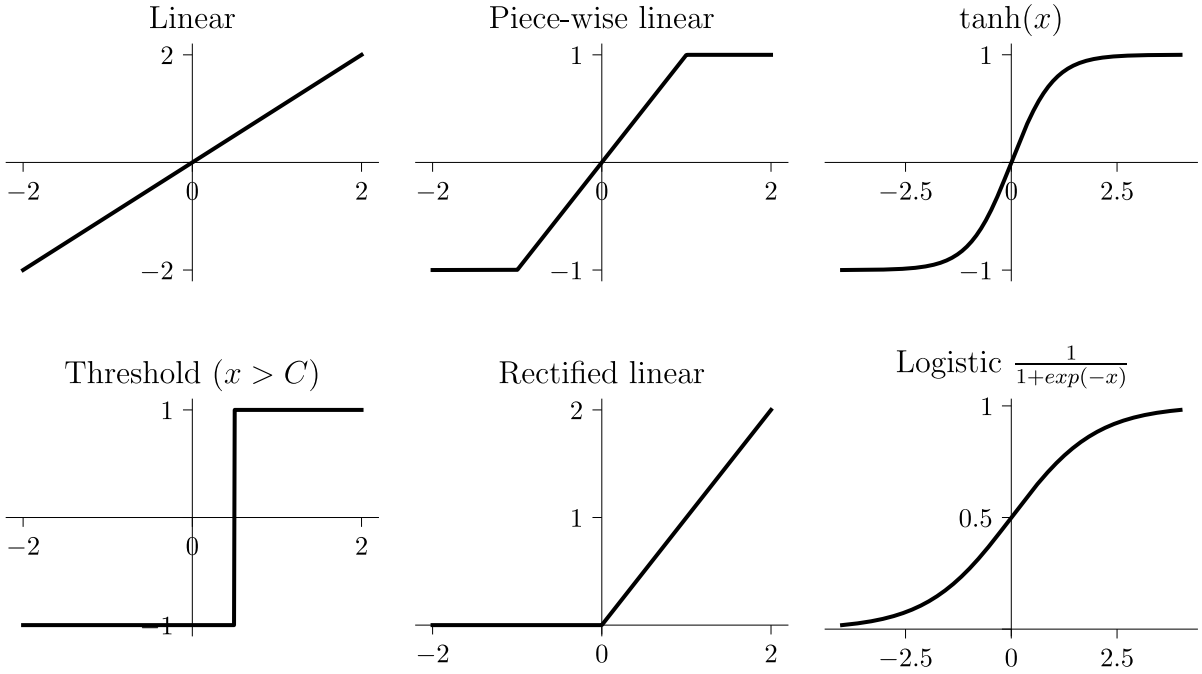


Figure 2.2: Plots of commonly used activation functions. Most resemble a sigmoid curve. Horizontal axis refers to $z_j^{(l)}$ values and the vertical axis refers to $y_j^{(l)}$ values.

Informally, signals $\mathbf{x}$ propagate throughout a neural network layer by layer, transforming their inputs and propagating the results to the following layers, until the output layer is reached. Its name refers to the way data flows through the NN, with the input layer as the starting point, then flowing towards the output layer, being transformed by each layer in a cascade fashion. Connections weights are used to strengthen or weaken the signals propagated, making the input of a neuron the weighted sum of values coming from all its connections.

Aside from connections and their weights, activation functions (ξ) also determine the behavior of NNs. As we see in (2.2), the activation function is the last operation performed on a neuron before its output is propagated further. This led researchers to investigate and propose many types of activation functions, each with their own properties. Figure 2.2 shows plots of commonly used activation functions.

This class of NNs is important not only due to their historical development, but also because they are, with the appropriate number of neurons, theoretically capable of approximating any function [12, 24]. In fact, one attains the same conclusion even if we restrict models to have one input layer, one intermediate layer and one output layer. Consequently, feedforward NNs are models expressive enough to solve a really large number of problems. Despite this remarkable result, given any function one desires to approximate, it is not trivial to design a NN that performs a satisfactory approximation of it, requiring substantial expertise and trial-and-error.

In order to be useful for a particular task, NNs are trained using data related to that task. The most used algorithm for training NNs is *backpropagation* [52]. For a given input $\mathbf{x}$, we compute the corresponding outputs $\mathbf{y}$ by propagating $\mathbf{x}$ through the NN. Now suppose $\mathbf{y}$ is not the desired output for that input. We express this fact by defining a *loss function*, also called a *cost function*, that represents the difference between the desired and the attained outputs. Usually, for higher loss values, we are further from the desired behavior. Thus, we update the connection weights in an attempt to optimize $\mathcal{L}$ with respect to some inputs by "backpropagating" errors through the NN. The most widely used method for performing those updates is stochastic gradient descent (SGD), an approximation of regular gradient descent, which consists of computing the gradient of $\mathcal{L}$ with respect to the weights and changing them so that we move, in the parameter (weight) space, the current weights in the opposite direction of the gradient. Usually, this update is a small proportion γ of the original gradient, called the *learning rate*.

In most scenarios, the training process is only responsible for determining the connection weights. The NN *architecture*, that is, number of neurons, layers (L), and their connections is set before training even begins. The architecture plays a significant role on the performance of a NN. As a result, many researches have come up with different architectures for different problem domains and tasks. In the following sections, we detail NN architectures that build upon feedforward NNs. Those architectures are designed to exploit properties of the data they learn from and are significantly different from the feedforward NN in the way they process their inputs and parametrize their connections. We will present those most relevant to our research in the following sections.

2.2.1 Convolutional neural networks

Originally proposed around three decades ago [36], one of the major driving forces of recent successes in neural networks are *convolutional neural networks* (CNNs). The main reason for this is their ability to take advantage of statistical properties of data such as natural images, video, and speech [10]. The fundamental ingredient of CNNs are convolutional layers whose purpose is to convolve a bank of filters over their input, producing the convolution results as output.

We consider an input space of square-integrable functions on a compact d-dimensional Euclidean domain, denoted $L^2(\Omega)$, with $\Omega = [0,1]^d$. A convolutional layer has the form $\mathbf{g} = \mathbf{y}(\mathbf{f}; \boldsymbol{\theta})$ and consists of a filter bank $\boldsymbol{\theta} = \Gamma = (\gamma_{ji})$, with $i = 1, 2, \ldots, p$ and $j = 1, 2, \ldots, q$ applied on a p-dimensional input $\mathbf{f}(x) = (f_1(x), f_2(x), \ldots, f_p(x))$, followed by a point-wise activation function ξ. Convolutional layers produce a q-dimensional output $\mathbf{g}(x) = (g_1(x), g_2(x), \ldots, g_q(x))$, expressed as

$$g_j(x) = \xi\left(\sum_{i=1}^{p} (f_i * \gamma_{ji})(x)\right), \tag{2.5}$$

where

$$(f * \gamma)(x) = \int_{\Omega} f(x - \omega)\gamma(\omega)\,\mathrm{d}\omega \tag{2.6}$$

denotes a convolution. Input dimensions are called *channels*, while output dimensions specifically are called either channels or *feature maps*. In image applications, for instance, an image is interpreted as function over a regular grid of m points (number of pixels) of the unit square, thus $\Omega = [0,1]^2$, with $p = 3$ input channels in the first convolutional layer, one for each color component in RGB-encoded images, while the γ_{ji} filters are small

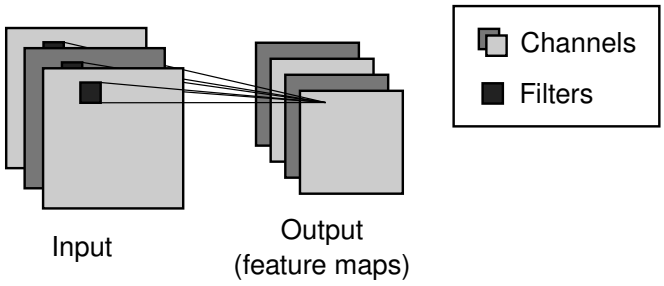


Figure 2.3: A convolutional layer. In this example, the layer has 3 input channels and 4 output channels. Here we show only the filters for the first output channel, however each output channel has a filter for each input channel.

matrices, usually 3×3, 5×5, etc. Figure 2.3 illustrates a convolutional layer with 3 input channels and 4 output channels in an image domain.

In addition to convolutional layers, CNNs also commonly incorporate *downsampling* (or *pooling*) layers, defined as

$$g_i(x) = P(\{f_i(x') \,:\, x' \in \mathcal{N}(x)\}) \tag{2.7}$$

for $i = 1, 2, \ldots, p$, where $\mathcal{N}(x) \subset \Omega$ denotes a neighborhood of x and P is a permutation-invariant function such as the frequently used L_p-norm. The usual choices of $P = L_1$, $P = L_2$, or $P = L_\infty$ are denominated *average-*, *energy-*, or *max-pooling*, respectively. Note that, in contrast to layers presented so far, downsampling layers have no learnable parameters.

Statistical properties such as stationarity, locality, and compositionality are owed to translation-invariance, local connectivity, and multi-resolution structure (respectively) of CNNs. Those properties are achieved by using "stacked" layers of convolutions and downsampling [10], analogous to what is done in feedforward NNs. Since the parameters of a convolutional layer are its filters, they are updated during training to gradually extract better, useful features from its inputs. For that reason, they are able to extract local features shared across the entire image domain, greatly reducing the amount of parameters when compared to generic neural network architectures. Such reduction in parameters entails no sacrifice of their expressive power, while reducing the risk of overfitting [10].

2.2.2 Recurrent neural networks

Temporal or sequential data are a special type of data where the order of data items is part of what they represent. Examples of those include video recordings, where the sequence of images and sound signals and their order determine the content they depict, and text, where words and letters are arranged in a specific order to convey a certain meaning. NNs specifically designed for learning from sequential data are called *recurrent neural networks* (RNNs) [56]. In contrast to feedforward neural networks, RNNs allow connection cycles, the most common being self-loops. This makes inputs in RNNs include not only data from the current step, but also data that were input in previous steps. Because of this, carefully designed RNNs can learn to make use of a potentially arbitrary subsequence of inputs it has already seen [56].

Different models of RNN elements exist, the main difference between them being the way they store and handle information between the input sequence elements [14, 27, 23].

Some models, however, have issues with learning dependencies among elements with many other elements intermediate elements. Most of those problems stem from the fact that gradients for RNNs tend to become increasingly small as one goes backwards in the sequence. Being very small, the NN parameters change very little or not at all, preventing the learning process from continuing. Long short-term memory (LSTM) cells [23] are trainable neural network cells with specific gates that provide mechanisms for manipulating recurrent connections and retain internal information [56]. These gates control when the information stored in the cell is updated, retained, or used as output. However, instead of having fixed behavior, LSTMs learn to use their gates during training. They help them handle issues with vanishing gradients mentioned earlier, and are what makes them able to learn long time dependencies [23].

An LSTM cell is organized as follows. It contains a *cell state* $\mathbf{c}_t$, which represents the information stored in the cell, a regular linear layer, $\mathbf{g}_t$, applied to the input, and three other linear layers implementing the gating mechanisms. The gates are: the *input gate* $\mathbf{i}_t$, controlling how much of the input (transformed by $\mathbf{g}_t$) is added to the cell state; the *forget gate* $\mathbf{f}_t$ determines how much of the cell state is preserved at time step t; and the *output gate* $\mathbf{o}_t$, which controls how much of the new cell state is used in the output $\mathbf{h}_t$. Being a recurrent unit, it also includes its previous output, also called hidden state, $\mathbf{h}_{t-1}$ as part of the input. Given an input sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$, an LSTM cell produces an output $\mathbf{h}_t$ at each time step $t = 1, 2, \ldots, T$ as follows

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{z}^{(ii)}(\mathbf{x}_t) + \mathbf{z}^{(hi)}(\mathbf{h}_{t-1})\right), \\
\mathbf{f}_t &= \sigma\left(\mathbf{z}^{(if)}(\mathbf{x}_t) + \mathbf{z}^{(hf)}(\mathbf{h}_{t-1})\right), \\
\mathbf{g}_t &= \tanh\left(\mathbf{z}^{(ig)}(\mathbf{x}_t) + \mathbf{z}^{(hg)}(\mathbf{h}_{t-1})\right), \\
\mathbf{o}_t &= \sigma\left(\mathbf{z}^{(io)}(\mathbf{x}_t) + \mathbf{z}^{(ho)}(\mathbf{h}_{t-1})\right), \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned} \tag{2.8}$$

where $\mathbf{z}$ is defined as in (2.1), $\odot$ denotes the Hadamard (i.e., point-wise) product, $\sigma(\cdot)$ is the point-wise logistic (sigmoid) function, and $\tanh(\cdot)$ is the point-wise hyperbolic tangent (see Figure 2.2). Figure 2.4 illustrates an LSTM cell.

LSTM-based RNNs are currently the state of the art in most temporal or sequential tasks, including sequence labeling, speech generation, and image captioning [56].

2.2.3 Autoencoders

Not all NNs are designed for supervised learning tasks, however. One important class of NNs, called *autoencoders*, were designed specifically for unsupervised learning. Essentially, they attempt to learn the input data distribution by reconstructing its inputs $\mathbf{x}$ using an internal representation $\mathbf{z}$. Specifically, an autoencoder is comprised of two components: an encoder and a decoder. As their names suggest, the encoder transforms the input $\mathbf{x}$ into the internal representation $\mathbf{z}$ and the decoder reconstructs the input from $\mathbf{z}$.

In general, autoencoders are used as a modeling tool for learning lower-dimensional representations of data by using a lower-dimensional internal representation, relative to its input dimensionality. They are trained to perform the best possible reconstructions of the inputs they are presented. Since it is limited to reconstructing the inputs only from $\mathbf{z}$, it must come up with ways of capturing meaningful information from $\mathbf{x}$ and encoding it

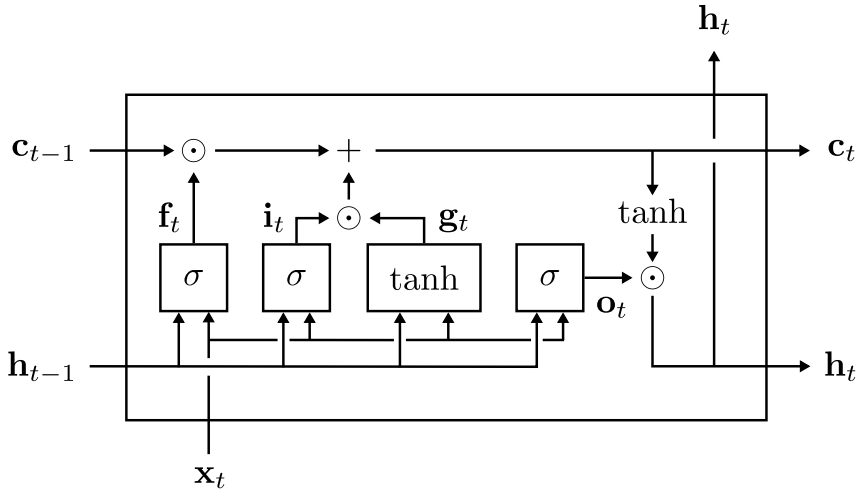


Figure 2.4: A long short-term memory (LSTM) cell. Given a cell state $\mathbf{c}_{t-1}$ and hidden state $\mathbf{h}_{t-1}$, this highlights how the new cell state $\mathbf{c}_t$ and hidden state $\mathbf{h}_t$ are produced from the current input element $\mathbf{x}_t$ of the sequence being processed. Boxes around the $\sigma(\cdot)$ and $\tanh(\cdot)$ functions indicate a linear layer with the respective (point-wise) activation function. The $\odot$ symbol denotes the Hadamard (i.e., point-wise) product.

into $\mathbf{z}$. As it improves on this task, its internal representation becomes better at reflecting the original data.

2.3 Graphs and neural networks

We consider a graph as a pair $G = (\mathcal{V}, \mathcal{E})$, with each $v_i \in \mathcal{V}$, representing a vertex (node or entity), $|\mathcal{V}| = n$, and each edge $(v_i, v_j, w_{ij}) \in \mathcal{E}$ representing a relation between v_i and v_j with a real-valued weight w_{ij}. We also use a shorthand notation $(v_i, v_j) = (v_i, v_j, 1)$ whenever necessary and assume that G is undirected.

In this section, we discuss recent work focused on employing NN models to graph-related learning problems. We start with geometric deep learning methods, which share many concepts in their formulation. Following that, we present some alternative methods that approach the problem from other perspectives, pointing out their differences. Lastly, we discuss the very few proposed methods that tackle these problems on time-varying domains.

2.3.1 Geometric deep learning

Geometric deep learning is an umbrella term for recent methods that generalize neural networks to non-Euclidean domains [10], particularly attempting to take advantage of the success attained by CNNs and their properties (Section 2.2.1). While geometric deep learning is concerned with a variety of non-Euclidean domains, such as graphs and manifolds, we will focus our discussion only on graph domains. The main difficulty for doing so is the fact that convolutions are not as easily expressed on those domains. One of the attempts at translating this key ingredient makes use of the *convolution theorem*. The theorem states that the Fourier transform of the convolution between two functions and the (point-wise) product between their individual Fourier transforms are equivalent. By employing a *graph Fourier transform*, we can use the theorem to perform convolutions

in the frequency (Fourier) domain, then apply an inverse graph Fourier transform on the result to obtain the graph convolution.

Suppose that the feature associated with each vertex is given by a function $f \in L^2(\mathcal{V})$, representing a signal over the graph vertices. We denote f as a column vector $\mathbf{f} = (f_{v_i})^T$, for all $i = 1, 2, ..., n$. Let W be the matrix of edge weights, where $[W]_{ij}$ denotes the weight of the edge between v_i and v_j and is 0 when there is no connection. Also let $D = \text{diag}(d_1, d_2, \ldots, d_n)$ be the degree matrix, with $d_i = \sum_j [W]_{ij}$. The graph Fourier transform of $\mathbf{f}$, is expressed as $\hat{\mathbf{f}} = \Phi^T \mathbf{f}$, where Φ is the matrix of eigenvectors of the graph Laplacian $L = D - W$, as L is symmetric (G is not directed) and positive-semidefinite, hence has n orthonormal eigenvectors. We write the eigendecomposition of L as $L = \Phi \Lambda \Phi^T$, with $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, $0 = \lambda_1 \leq \lambda_2, \leq \ldots \lambda_n$, and $\Phi = (\phi_1, \phi_2, \ldots, \phi_n)$. Since $\Phi \Phi^T = I$, the inverse graph Fourier transform is $\mathbf{f} = \Phi \hat{\mathbf{f}}$. We can now express a convolution $f * g$ in a graph domain as

$$\mathbf{f} *_G \gamma = \Phi \, \text{diag}(\hat{\gamma}) \Phi^T \mathbf{f}, \tag{2.9}$$

with $\hat{\gamma} = (\hat{\gamma}_1, \hat{\gamma}_2, \ldots, \hat{\gamma}_n)$ being the spectral representation of the filter γ, $g \in L^2(\mathcal{V})$.

Using these results, Bruna et al. [11] proposed the *Spectral CNN* (SCNN), adopting a convolutional layer capable of performing convolutions in a way analogous to a regular CNN, but on a graph domain. The *spectral convolutional layer* is defined as

$$\mathbf{g}_j = \xi \left(\sum_{i=1}^{p} \Phi_k \Gamma_{ji} \Phi_k^T \mathbf{f}_i \right), \tag{2.10}$$

where $\mathbf{f}_i$ is the i-th channel of the p-dimensional input signal on vertices, $\mathbf{g}_j$ is the j-th channel of the q-dimensional output signal on vertices, Γ_{ji} is a $k \times k$ diagonal matrix of multipliers acting as a filter in the frequency domain, Φ_k denotes the matrix of the first k eigenvectors of the graph Laplacian, and ξ is a point-wise activation function. By taking only the k first eigenvectors, the convolution sets a cutoff frequency, which depends on the features of the graph. Since they are the ones describing the smooth structure of the graph, in practice we typically have $k \ll n$ [10].

The spectral construction of a SCNN, however, is domain-dependent. This means that, if we train a SCNN on some graph, the trained filters will be specific to the structure of that graph since they are dependent on the basis (eigenvectors of L) used. While it is possible to circumvent this using compatible basis, this requires knowing beforehand some correspondence between the domains in order to determine this common basis. Therefore, the SCNN is much more adequate for applications where the domain remains mostly the same, such as social networks. In addition, the need to perform both the multiplication by Φ_k^T and then by Φ_k results in $O(n^2)$ complexity, which can be even more prohibiting.

We can overcome those limitations by avoiding the explicit computation of Laplacian eigenvectors. From (2.9), it follows that a polynomial applied on L behaves as a polynomial applied only on Λ, that is,

$$\gamma(L; \boldsymbol{\theta}) = \gamma(\Phi \Lambda \Phi^T; \boldsymbol{\theta}) = \Phi \gamma(\Lambda; \boldsymbol{\theta}) \Phi^T,$$

where

$$\gamma(\Lambda; \boldsymbol{\theta}) = \sum_{k=0}^{r-1} \theta_k \Lambda^k \tag{2.11}$$

denotes a polynomial function with r coefficients $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_{r-1})$. This results in filter matrices of the form $\Gamma_{ji} = \gamma(\Lambda; \boldsymbol{\theta}_{ji})$ whose entries are expressed in terms of the Laplacian eigenvalues.

This filter representation has an interesting property for convolutions. The Laplacian is known to be a local function that operates on 1-hop neighborhoods. Thus, the k-th power extends it to k-hops neighborhoods. Since the filter defined by (2.11) is a linear combination of powers of the Laplacian, it behaves as a diffusion operator limited to r-hops neighborhoods.

Defferrard et al. [13] propose the Graph CNN, which uses filters of the form given by (2.11), but uses Chebyshev polynomials to express them, generated by the recurrence relation

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), \tag{2.12}$$

and $T_0(x) = 1$, $T_1(x) = x$. We will henceforth refer to this architecture as ChebNet to avoid confusion. In addition, it also uses a rescaled Laplacian $\tilde{L} = 2\lambda_n^{-1}L - I$ and $\tilde{\lambda} = 2\lambda_n^{-1} - I$, mapping the eigenvalues from the interval $[0, \lambda_n]$ to $[-1, 1]$, since Chebyshev polynomials form an orthonormal basis in $[-1, 1]$. Thus, filters are parametrized uniquely by the r coefficients $\boldsymbol{\theta}$, leading to filters of the form

$$\begin{aligned} \gamma(\tilde{L}; \boldsymbol{\theta}) &= \sum_{k=0}^{r-1} \theta_k \Phi T_k(\tilde{\Lambda}) \Phi^T \\ &= \sum_{k=0}^{r-1} \theta_k T_k(\tilde{L}). \end{aligned} \tag{2.13}$$

The Graph Convolutional Network (GCN) [32] further simplifies this construction by assuming $r = 2$ and $\lambda_n \approx 2$. This assumption on the maximum value of the eigenvalues is expected to have little effect, however, as the NN can adapt to it by learning appropriate parameters $\boldsymbol{\theta}$ that account for this [32]. Using those assumptions and expanding the Chebyshev polynomial, we have

$$\begin{aligned} \gamma(\mathbf{f}; \boldsymbol{\theta}) &= \theta_0 \mathbf{f} + \theta_1 (L - I)\mathbf{f} \\ &= \theta_0 \mathbf{f} - \theta_1 D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \mathbf{f}. \end{aligned} \tag{2.14}$$

The authors further constrain the parameters by assuming $\theta = \theta_0 = -\theta_1$, reducing the filter to a single parameter: $\gamma_\theta(\mathbf{f}) = \theta(I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}})\mathbf{f}$. This transformation, however, makes the eigenvalues of $I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ lie in $[0, 2]$. Repeated application of this filter can be numerically unstable, motivating the use of the following filter instead

$$\gamma(\mathbf{f}; \theta) = \theta \tilde{D}^{-\frac{1}{2}} \tilde{W} \tilde{D}^{-\frac{1}{2}}, \tag{2.15}$$

where $\tilde{W} = W + I$, $\tilde{D} = \operatorname{diag}(\tilde{d}_1, \tilde{d}_2, \ldots, \tilde{d}_n)$, and $\tilde{d}_i = \sum_{j \neq i} \tilde{w}_{ij}$.

The GCN formulation has the advantage of being very computationally efficient. Moreover, one can achieve similar r-hop neighborhood filters by stacking r layers of filters in succession. Bronstein et al. [10] note that while we can derive ChebNet and GCN starting from the spectral domain, as shown here, those two NNs rely on simple spatial (not spectral) filters acting on the r- or 1-hop neighborhood of the graph. Thus, they can be viewed as a special case of the Graph Neural Network (GNN) [17, 54], which is among the first proposed NN architectures for graph data. As before, given a p-dimensional input signal over the vertices, denoted in vector form by $\mathbf{f}$, a GNN learns the parameters $\boldsymbol{\theta}$ of a

generic function η applied to $W\mathbf{f}$ and $D\mathbf{f}$. The p-dimensional output $\mathbf{g}$ of this operation is given by

$$\mathbf{g}_i = \eta(W\mathbf{f}_i, D\mathbf{f}_i; \boldsymbol{\theta}) \tag{2.16}$$

for every $i = 1, 2, \ldots, p$. The general formulation of GNNs allows them to emulate many other operations, such as the graph Laplacian (by setting $\eta(W\mathbf{f}_i, D\mathbf{f}_i; \boldsymbol{\theta}) = D\mathbf{f} - W\mathbf{f}$), Chebyshev polynomials by stacking r layers, each with η emulating a specific degree of the polynomial.

2.3.2 Implicit relational reasoning

Alternative approaches implicitly incorporate the graph structure instead of defining new operators over them. One of these is the Differential Neural Machine (DNC), which combines NNs with external memories. Models with this architecture learn to manipulate an external memory through a learned control mechanism. Then, a graph can be stored in this memory in order to to answer queries about it, devising solutions to simple graph problems, such as shortest paths [18].

Relation Networks (RNs) [53] handle relational (graph) data with two neural networks $f_1(\cdot; \boldsymbol{\theta}_1)$ and $f_2(\cdot; \boldsymbol{\theta}_2)$. Each input sample is a graph, $G = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V} \subset \mathbb{R}^m$. In its original form, we assume $\mathcal{E} = \mathcal{V} \times \mathcal{V}$, that is, G is a complete directed graph. The first step is to produce a representation for each edge (relation), using f_1. Then, those representations are summed and this sum is used as input for f_2. The output of a RN is expressed as

$$\mathbf{y}(\mathcal{G}; \boldsymbol{\theta}_1, \boldsymbol{\theta}_2) = f_2(\mathbf{z}; \boldsymbol{\theta}_2), \quad \mathbf{z} = \sum_{(\mathbf{u},\mathbf{v}) \in \mathcal{E}} f_1(\mathbf{u}, \mathbf{v}; \boldsymbol{\theta}_1) \tag{2.17}$$

This way of processing relationships has several advantages. First, RNs implicitly learn to infer relations, as the original formulation assumes that all nodes are connected to all other nodes. This means that an RN can learn when a connection is relevant or not, being useful when connections are not known beforehand. Second, RNs compute a representation for each relation using the same function. Compared to approaches where the graph structure is represented by the NN itself, an RN can process each pair of objects on an independent feed-forward pass, reducing the amount of parameters and the chance that g overfits to a particular relation. Third, they are invariant to the order in which relations are given as input, since f needs the sum of all values produced by g on each pair of objects.

Compared to previous architectures, RNs have a very flexible formulation, allowing generic functions over the edges to be learned, combined with a decision function over a graph. The experiments performed by the authors [53] show that RNs outperform other methods for solving tasks involving relational reasoning while also being better at non-relational tasks on the same data. They highlight this flexibility by coupling each f_1 step with additional information about the graph that is global. Those experiments, however, lack comparisons between RNs and the methods discussed in Section 2.3.1. Moreover, in their original formulation, RNs are more computationally costly compared to them.

2.3.3 Time-varying domains

The NN architectures presented so far for learning from graph data were designed for static scenarios, where the signals or graph structure does not change over time. As previously stated, in this research we are interested in problems where time is also an

important factor to account for. In this case, we have three possible scenarios [38]: (i) the signals associated with either nodes or edges change over time; (ii) the nodes or edges themselves may be added or removed over time; (iii) both (i) and (ii) happens. There are solutions proposed in the NN literature for the first scenario, which we discuss below. For the other two scenarios, however, we have not found a NN perspective to tackling them.

Seo et al. [57] tackled this problem by combining ChebNet with LSTM cells, with two different variants. The first variant consists of using ChebNet to extract features from the graph signals, followed by an LSTM for sequence learning. The biggest drawback of this approach is that the number of parameters of the LSTM is directly related to $|\mathcal{V}|$. This limits the application of this model to smaller graphs. To avoid this limitation, the second variant generalizes a previously proposed extension of LSTMs, called ConvLSTM [58]. ConvLSTM is essentially a replacement of the linear operations $\mathbf{z}$ in (2.8) with regular convolutions. Analogously, Seo et al. [57] replace $\mathbf{z}$ with ChebNet convolution filters from (2.13).

While the second approach improves upon ConvLSTM in video prediction, it cannot be employed in tasks where the input dimensionality is too large, as the cell state is directly proportional to it. Additionally, Seo et al. [57] show in a text prediction experiment that their first approach outperforms regular LSTMs by adding a 4-nearest neighbors graph on a word vector representation as input to their model. While notable for being one of the first approaches to the problem of time-varying signals over graphs, the experiments only show slight improvements over alternative solutions. Moreover, the video experiment is limited to very simple shapes over a single black background, which is not very challenging.

Structural RNNs (S-RNN) [26] were also designed for this problem, however on a more specific type of graph. Spatio-temporal graphs (st-graphs) encode interactions between objects over time by representing each object in a certain problem as a vertex and their interactions as edges on this graph, thus objects' spatial information are encoded as signals over time on the vertices. S-RNNs are built from the st-graph before training is performed. This is done by unrolling the st-graph over time, identifying individual factors that are independent components, and then building one RNN for each factor. S-RNNs are relatively scalable and performs well on the domains envisioned by the authors. However, we argue that it lacks the elegance of naturally handling the graph structure from within the NN architecture, like other solutions presented in the previous sections.

2.4 Discussion

There are still many opportunities for employing NNs in graph-structured data, particularly in time-varying domains, where many assumptions of existing methods do not hold, since this data representation is notably more complex than static graphs. However, as one can express increasingly more complex scenarios with them, we believe they are the next step in employing NNs on graph domains, as having the appropriate tools for handling the temporal aspect of dynamic graphs enables better understanding of those scenarios.

This research direction can lead to new state-of-the-art methods for: link prediction, that is, predicting which edge will be added or removed from a graph at a certain time step; predicting changes in node or edge attributes; classifying a graph in a certain configuration as being in one of many possible states; predicting changes in the graph structure; and generative models for new graphs.

Chapter 3

Methodology

As previously stated, we aim to conceive and make use of NN architectures with two main requirements to attain our objectives: (i) they must be able to handle graph data; and (ii) they must be able to handle data that changes over time. In other words, we are focused on methods that can learn from dynamic graphs. We say that dynamic graph data is data where objects and their relationships are expressed implicitly or explicitly and how those, either the objects, relationships, or both, change over time. In this chapter, we present our methodology for accomplishing this task in the two envisioned scenarios: temporal link prediction for recommendation and sports analytics with dynamic graphs.

While these two tasks use similar data (dynamic graphs), each involve different underlying phenomena. Consequently, we intend to investigate how the different NN components for graphs and for sequences presented in Chapter 2 can be combined to handle dynamic graph data. This investigation will be useful as a first step for tackling for both tasks, as it will highlight the most promising approaches by identifying their limitations and advantages.

3.1 Temporal link prediction as recommendation

In a recommendation system, users are offered a set of items they might be potentially interested in. One common approach to this problem is collaborative filtering [15], where the collective ratings users give to items are used to predict unseen ratings. This is commonly represented as an undirected bipartite graph, where observed ratings are weighted edges linking one user to one item, with the weight representing the rating value. Following recent progresses on recommender systems based on collaborative filtering, we also approach this problem as a link prediction task. The goal in link prediction is to learn, from a set of edges, a model capable of predicting the weight of unseen edges. Here, predicting this weight is equivalent to predicting what rating the user would give to an item, thus estimating how likely they are to be interested in it.

Suppose that instead of just being given a collection of edges linking users and items, we also could know *when* each edge has appeared. We extend these concepts to dynamic graphs by considering our hypothesis for this research. In this case, we could leverage this information to produce models that know how users change their tastes, or even how some product is perceived by other users, over time. This enables new types of collective information about them to arise, such as when some movie is relevant only for a short period of time or users discover some new genre of music they are now becoming increasingly interested in.

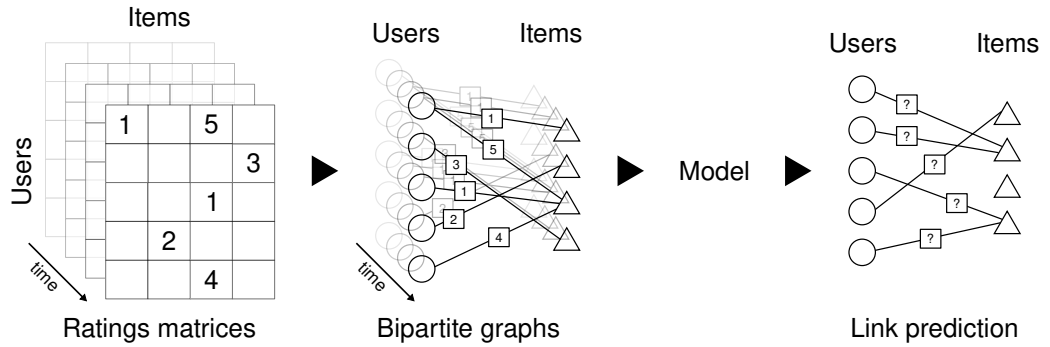


Figure 3.1: Overview of a temporal link prediction task for recommendation. A series of user-item ratings matrices represent ratings as they happen over time. This data is interpreted as a dynamic bipartite graph, where nodes represent users and items, while weighted edges represent ratings. The task is to predict the weights of unseen edges.

Formally, let $\mathcal{U} = \{u_i | 1 \leq i \leq n_u\}$ and $\mathcal{V} = \{v_j | 1 \leq j \leq n_v\}$ denote the set of users and items, respectively, with n_u being the number of users and n_v being the number of items. Also let $G = (\mathcal{W}, \mathcal{E}, \mathcal{R})$ be the ratings bipartite graph, where $\mathcal{W} = \mathcal{U} \cup \mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, representing all observed ratings, and $\mathcal{R}$ is the set of possible rating values (edge types). Each observed rating is denoted by $(u_i, r, u_v) \in \mathcal{E}$, with $u_i \in \mathcal{U}$, $v_j \in \mathcal{V}$, and $r \in \mathcal{R}$. This graph is represented by an adjacency matrix M, where each element $M_{ij} = r$ corresponds to the edge (u_i, r, v_j). We also consider a set of feature vectors for users and items, denoted as $\mathcal{X} = \{\mathbf{x}_w | w \in \mathcal{W}\}$.

Future edges represent the unknown ratings we want to predict and, since this is used for recommendations, we are not interested in predicting *when* some rating will take place, but rather the *value* of such rating. We represent the dynamic graph as a series of rating matrices $M^{(1)}, M^{(2)}, \ldots, M^{(T)}$, with $T > 1$. Equivalently, this sequence represents the states of the dynamic graph over time $G^{(1)}, G^{(2)}, \ldots, G^{(T)}$. The sequence is such that it contains all observed ratings. Figure 3.1 illustrates this task.

One subtlety to this approach is that this sequence can represent the ratings over time in a number of ways. This sequence can be incremental, such that each step in it consists of new edges plus all edges from previous steps; or disjoint, where each step contains only new edges, representing a time "window". Moreover, varying the amount of information in one step can influence the performance of trained models, since one can have steps with a large number of edges, but shorter sequences of ratings graphs (matrices); or steps with a small number of edges, but longer sequences of ratings graphs (matrices). As a consequence, this trade-off between how much information and how many steps we want this information to be distributed is an important aspect to study.

For this task, we will use the same experimental protocol adopted in previous research [65, 32, 33, 55]. We use an incomplete set of edges $\hat{\mathcal{E}} \subset \mathcal{E}$ as training data, while the remaining edges, $\mathcal{E} - \hat{\mathcal{E}}$, are used as validation and test data. Naturally, due to the temporal nature of our setting, we emphasize that training edges $\hat{\mathcal{E}}$ must be older than validation or test edges, as we want only past information to be used to predict future edges. The selected datasets for these experiments are summarized in Table 3.1. Those are selected specifically for having temporal information about the ratings available while also representing real scenarios. Additionally, we include a synthetic dataset, based on the same algorithm proposed by Kalofolias et al. [29]. This dataset will allow experiments with more control over the behavior of users.

Table 3.1: Datasets for the temporal link prediction task.

Dataset	Users	Items	Ratings
Synthetic [29]	–	–	–
ML100k [20]	943	1,682	100,000
ML1M [20]	6,040	3,900	1,000,209
ML10M [20]	71,567	10,681	10,000,054
AMZN-movie [42]	889,176	253,059	7,911,684
AMZN-music [22]	478,235	266,414	836,006

3.2 Soccer match learning with dynamic graphs

For this step, we envision a less structured and more exploratory scenario, compared to the previous. This is part of a bigger research project, jointly funded by the Sao Paulo Research Foundation (FAPESP) and the Netherlands Organization for Scientific Research (NWO). The project focuses on advancing our understanding of soccer using data-driven approaches in a collective effort between Brazilian and Dutch researchers of many disciplines, such as computer science, sports science, physiology, and mathematics.

Suppose matches are represented by dynamic graphs of players, where each player is a node and interactions between them can be modeled by connections between nodes. Furthermore, suppose there is a fixed time step between each graph change. First, the total number of changes for each dynamic graph does not vary much, as matches have a predefined duration that is not considerably different between them. Second, the number of nodes (players) also does not change significantly. Third, attributes such as the player positions are bounded by the rules of the game. Finally, all graphs represent an event that can assessed by a human, therefore establishing a plausible ground-truth for experiments. Soccer matches are composed of two teams of eleven players with a large number of possible interactions between them. In this scenario, we have a highly complex system of interactions that is still constrained to some extent, due to the rules in place for soccer matches, restricting possible states.

As such, we expect that this leads to novel solutions, from a machine learning perspective, for tackling many problems under the same framework. For example, suppose we aim to predict match events. Which signals should be associated with nodes or edges? What types of connections between players lead to better predictions of such events? For building representations of match clips, which dynamic graph structure represents desirable properties so that the representations can be rich enough to retrieve similar match states, in some sense, from other matches? Previous approaches focus on edges summarizing passes between players [8] or even how a player is "close" to another, taking into account the positions of the opposing team's players [51]. While both representations have their merits, it is not obvious at first sight which would be the most promising approach with dynamic graphs. Because of this, in contrast to the previous stage, we plan a more exploratory approach at this stage. Based on this representation of a soccer match, we contemplate three initial learning tasks, described below.

3.2.1 Learning tasks

The first task is related to prediction of time-varying signals along the vertices representing players, either for vertex-level prediction or graph-level prediction. For instance, we aim

at predicting, the positions of a player or subgroup of players at the next or next few time steps. We are also interested in investigating how the performance of the prediction is influenced by different criteria for connections among players.

For the second task, we envision an information retrieval application. Specifically, our intent is to make use of representations of temporal slices of graph configurations, henceforth referred to as *clips*, in order to perform retrieval of similar configurations. A typical scenario would be the identification of a particular tactical interaction between players. From there, one is interested in retrieving, from a large database of soccer matches, similar tactical interactions, so that they can be compared. One advantage of using graphs, in this case, is that we alleviate the challenges of performing the same task over a video stream directly. Camera angles, weather conditions, image acquisition quality are all factors that can influence the performance of such system, significantly increasing the difficulty of the task. Since a graph representation would be independent of these factors, we expect that such scenario is more attainable than a computer vision solution.

The third task is a more ambitious learning scenario, aiming at learning emergent behavior from the time-varying graph. In this task, we are interested in determining what kinds of player interaction patterns are more likely to lead to important events in a match, such as a goal shot. This task provides several challenges as these depend not only on one team's performance, but how both teams are interacting with teammates and opponents.

3.2.2 Data and evaluation

For performing experiments and conducting the research, we initially plan to use data already collected [51] from around 10 soccer matches where player positions are collected at a rate of 30Hz, resulting in about 81,000 snapshots for one 45 minutes match (half-time). Along with the player positions, we also have a log of relevant match events, such as goals, passes and fouls.

Evaluation of the first two scenarios is considerably easier than the third. Since both of them are similar to tasks in the literature, we can adopt existing methodologies from previous research. The third scenario, however, is less well-defined with respect to the learning task. Thus, we refrain from establishing an evaluation methodology due to its exploratory nature.

Chapter 4

Preliminary results

In this chapter, we present some preliminary results as evidence supporting our motivation for this research. Since our objectives revolve around investigating our hypothesis (see Section 1.1) for both recommender systems and sports analytics, specifically soccer, we present two main experiments.

In Section 4.1, we show how relational reasoning can be used in a problem related to the use of word tags associated with multimedia documents to identify which of them are of a specific category. While at first glance this problem does not seem to require relational reasoning, the competitive performance attained by Relation Networks (Section 2.3), not only by itself, but also augmenting a multimodal scenario, highlights RNs as a versatile framework for relational reasoning. This supports previous evidence [53] for its use in a variety of domains and is particularly encouraging with our intended use in sports analytics in mind. Furthermore, RNs most prominent limitation, namely the number of entities that can be processed at a time, is not a concern in soccer, where we have even less entities than many of the examples.

In Section 4.2, we report some preliminary experiments to build recommender systems that take into account how user ratings take place over time. Inspired by previous work where the task of recommending items to users is posed as a matrix-completion problem [47, 3], we show how one can use temporal information associated with ratings to attain competitive performance compared to a state-of-the-art method that uses a similar framework for recommendation tasks. This experiment highlights a promising research direction for models that can understand how users and items are related over time, so that better recommendations can be provided. In addition, it suggests that dynamic graphs indeed have the potential to provide more detailed representations of the underlying phenomenon.

4.1 Flood detection from social media data

The Multimedia Satellite Task [6], which was part of the Multimedia Evaluation Benchmark (MediaEval) 2017, proposed a retrieval task of flooding evidence from social media. In this task, participants were required to rank a collection of multimedia items, either images, their associated metadata, or both, such that those depicting flooding events should be ranked higher than others. Bischke et al. [5] proposed a solution using a Support Vector Machine (SVM) with a radial basis function kernel. The SVM was trained using visual features extracted via a pre-trained CNN based on the X-ResNet architecture [28] and the metadata was represented using word2vec [43] embeddings trained on the metadata itself.

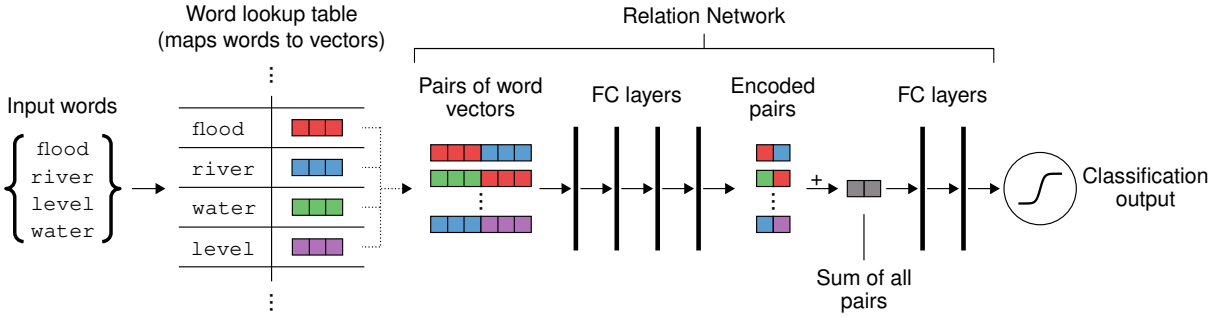


Figure 4.1: Our architecture using a Relation Network (RN) to perform relational reasoning on unstructured text data from social media streams. The RN learns which relationships between words are relevant, producing a decision of whether a particular set of words describe a flooding event.

For handling both at the same time, they used a simple concatenation of both visual and textual features. In what follows, we describe our approach using a RN to handle the tags present in the metadata.

Let V be the set containing all words in the training set. The first step is to map each one of them into t-dimensional vectors, which the RN uses as object representations. We adopted two different strategies for this, namely a word lookup table and distributed word representation, which work as follows.

For the word lookup table, we initially represent each word as a 1-hot vector of dimension $|V|$. We then map each word vector using a linear projection (matrix) that is continuously updated as part of the learning process. This mapping, also known as a word lookup table, produces a vector of $t = 100$ dimensions for each word. As the vectors are learned from scratch, this approach has two main advantages. First, vector representations are constantly improved while training the neural network. Second, we do not require a large number of parameters dedicated to learning word representations. This comes with the drawback that our vocabulary is limited to what is seen in the training data, thus any unseen words are mapped to vectors of zeros and the network must learn how to ignore objects represented by vectors of zeros. An overview of the architecture, illustrating this strategy, is shown in Figure 4.1.

In the case of distributed word representations, we replace our word lookup table with fastText word embeddings, a state-of-the-art distributed continuous representation model, pre-trained on a corpus of 16 billion tokens [44]. Since this involves using a fixed vector for each word, the intent is to evaluate the performance of the RN for relational reasoning with words when the object representation is high quality.

We use the same base architecture for RNs, where f_1 processes relations and f_2 produces the decision over the aggregation of each output from f_1. Specifically, f_1 consists of 5 fully-connected layers of 256 SELUs each, while f_2 consists of 4 fully-connected layers of 256 SELUs each, followed by a single sigmoid unit for classification. We chose this architecture based on two main results from previous work. First and foremost, Santoro et al. [53] successfully used a 4- and 3-layer fully-connected ReLU networks for f_2 and f_1 for relational reasoning involving 25 objects. Since we had almost double the amount of objects in some cases, we chose to add one more fully-connected layer to both f_2 and f_1. Second, Klambauer et al. [34] have recently shown on a large number of datasets that SELUs, when used in NNs of fully-connected layers, consistently outperform alternatives, such as ReLUs combined with techniques such as BatchNorm [25].

For the multimodal scenario, where we have both images and their metadata, we use a larger architecture that incorporates a CNN for image data and the same RN-based architecture previously described for the metadata. The CNN we employ is a ResNet-18 [21] pre-trained on ImageNet, but with its classification layer replaced by a fully-connected layer of 512 SELUs. The RN also has its classification layer replaced by a fully-connected layer of 256 SELUs. The output of both networks is concatenated and then fed into a fully-connected layer of 512 SELUs, followed by a layer with a single sigmoid unit for classification.

4.1.1 Experimental setup

Dataset

We test our proposed architecture on the dataset for the "Disaster Image Retrieval from Social Media" subtask of the Multimedia Satellite Task. The dataset is composed of 6,600 images from the YFCC100M dataset [61] alongside their metadata, such as description, tags, and title, as text. The images were selected by querying the YFCC100M dataset using common words for describing flooding events. Following that, human annotators refined and labeled those results, then added other unrelated images to increase the diversity of the data. The challenge lies in differentiating pictures of places where the presence of water is expected, such as rivers and lakes, from pictures of places where the water levels are not expected to be the one depicted. The dataset also has a predefined test set, which corresponds to 20% (1,320) of the samples. We also set aside 1,054 samples from the training set for validation purposes.

The set of words for each image is the union of all individual words found in all the tags combined. Each individual word is obtained by employing the NLTK [4] word tokenizer to extract the individual words from each tag.

Evaluation

We adopt the same evaluation protocol as the Multimedia Satellite Task competition [6]. It consists of three different scenarios, each of them evaluating one data modality: in **run 1**, only images are provided as input; in **run 2**, only the metadata are provided; finally, in **run 3**, both images and metadata are used. For each of them, we evaluate the test set items ranked by confidence that they depict a flooding event, from the most to the least confident. They are evaluated using the Average Precision (AP) at cut-off 480 and the mean AP at cut-offs 50, 100, 250, and 480.

4.1.2 Baselines

We compare our method to the approach proposed by Bischke et al. [5], described earlier. The authors originally used word2vec [43] embeddings trained on the Satellite Task dataset. Since our model uses fastText embeddings pre-trained on a much larger corpus, we also employ them on this baseline, thus its performance reported here is better than those originally reported by the authors. Moreover, for this baseline, we use the best SVM hyperparameters found via a grid search over the validation set.

Table 4.1: Average Precision at 480 (AP@480) and Mean Average Precision (MAP) at cut-offs 50, 100, 250 and 480 on the test set. Values presented for our models are the mean and standard deviation of 10 distinct training procedures with random initialization. Results with * indicates those are not the originally reported results, as they are obtained using fastText embeddings instead of word2vec.

		AP@480 (%)	**MAP (%)**
Run 1	Bischke et al. [5]	86.64	95.71
	CNN	84.79 ± 1.32	95.12 ± 0.82
Run 2	Bischke et al. [5]	83.11*	90.05*
	RN	80.75 ± 3.00	83.60 ± 6.27
	RN + fastText	84.87 ± 0.52	86.69 ± 0.76
Run 3	Bischke et al. [5]	90.45	97.40
	CNN + RN	97.23 ± 0.37	99.02 ± 0.22
	CNN + RN + fastText	97.50 ± 0.27	99.17 ± 0.17

4.1.3 Training details

Our models are trained using Adam [30] with a learning rate of 2.5×10^{-4}. We also employ a weight decay of 10^{-4}. We train our models for 100 epochs using mini-batches of 32 samples. The reported results use models that achieved the best MAP on the validation set during the training procedure.

4.1.4 Results and discussion

The results for each scenario are summarized in Table 4.1. While we are focused on the performance of the RN for learning from the text data, we still consider the performance of the CNN on the image-only scenario (run 1). This enables us to compare how the CNN performs by itself and when used jointly with the RN in the multimodal scenario (run 3). Even though our CNN, a ResNet-18, has almost 4 times fewer parameters than the one used by Bischke et al. [5] as a feature extractor for their SVM, both achieve similar performance in run 1.

In run 2, we see that the RN architectures achieved competitive results with respect to the baseline SVM. The RN with word representations learned from scratch via a linear embedding, however, has a large variance both in terms of AP@480 and mean AP. When compared with the same architecture using fastText word embeddings as input, we see that the latter is more consistent while also outperforming the former. For a better comparison between them, we also computed their precision at each cut-off from 1 to 480, shown in Figure 4.2. The RN with fastText representations as input has results very similar to the plain RN model at the first few items, on average, before pulling ahead consistently. However, the plain RN shows a very large variance. Despite both of them having the same overall architecture, this shows that training a RN might be considerably more practical with a good representation for the objects.

As already mentioned in the previous sections, the lookup table is restricted to the words seen during training. Since it cannot produce representations for unknown words, these words must be ignored. However, even by adopting an open vocabulary model similar to the one proposed by [39], our model could not achieve the same results. This suggests either that the amount of training data is not enough to train the open vocabulary model or this limitation is not severe enough to produce a noticeable impact.

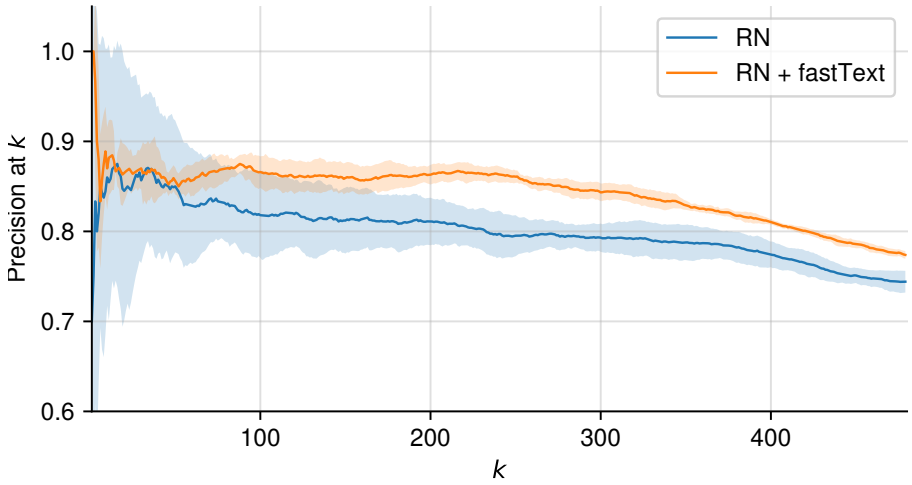


Figure 4.2: Precision scores at each cut-off (k) from 1 to 480 attained by the RN architectures in run 2 (see text for more details). The scores shown are the average of 10 models trained with different initial weights; standard deviations are represented by the shaded area.

Finally, in run 3, the models combining a CNN and a RN produced the best results, being above the baseline by a significant margin. Interestingly, the performances of the RN using a lookup table and the RN using word embeddings were similar. As was done for run 2, we also computed the precision scores for them, shown in Figure 4.3. These results highlight the advantage of using neural networks in data fusion: the CNN and RN were trained jointly in an end-to-end manner, requiring no significant changes to their individual architectures. Upon inspection of precision scores, we see that both were indeed very close in performance and had relatively small variances. An explanation for these results might lie in the fact that we have two modalities. Both models had already performed well on their own and, while the RN using a lookup table lagged behind the one using word embeddings, the CNN reduces this gap significantly due to a possible overlap of cases the CNN and the RN using word embeddings correctly classifies them while the RN using lookup table does not.

While the RNs were competitive in these experiments, one might argue that it was due to the presence of certain words instead of the relationships between them being taken into account. Because of this, we also attempted to build a model replacing the RN with a DeepSets [67], since it is an architecture designed to operate on sets of objects. While it had roughly the same number of parameters as our aforementioned RN models, the DeepSets models did not converge in these experiments. This provides further evidence that the RN is an essential component and a relational approach to this problem is adequate.

4.2 Temporal Link Prediction as Recommendation

The second experiment explores the use of dynamic graphs for building recommender systems. In particular, we are interested in investigating how the temporal aspect can be leveraged to improve existing approaches that pose the task of recommending items to users as a link prediction task (Section 3.1). To do this, we extend GCMC [3], a method for matrix completion using GCNs (Section 2.3), to incorporate dynamic graphs.

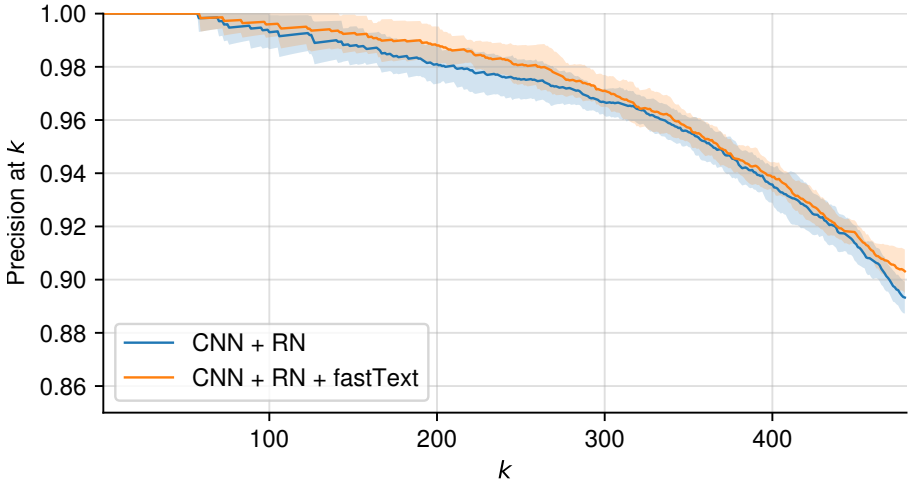


Figure 4.3: Precision scores at each cut-off (k) from 1 to 480 attained by the multimodal (CNN + RN) architectures in run 3 (see text for more details). The scores shown are the average of 10 models trained with different initial weights; standard deviations are represented by the shaded area.

4.2.1 Graph convolutional matrix completion (GCMC)

Put simply, GCMC predicts ratings in an autoencoder (Section 2.2.3) fashion: it first encodes each user and item, then uses a decoder that takes a pair of encoded user $\mathbf{z}_{u_i}$ and item $\mathbf{z}_{v_j}$ to predict the rating the user would provide to the item. Recall the definitions for link prediction in the context of recommendation (Section 3.1). In GCMC, the encoding step for each user is comprised by a GCN-like graph convolution step followed by a fully connected layer:

$$\mathbf{h}_{u_i} = \xi\left(\text{accum}_{r\in\mathcal{R}}\left\{\sum_{j\in\mathcal{N}_{i,r}}\frac{1}{c_{ij}}W_r\mathbf{x}_{v_j}\right\}\right), \tag{4.1}$$

$$\mathbf{z}_{u_i} = \xi(W\mathbf{h}_{u_i}), \tag{4.2}$$

where c_{ij} is a normalization constant, accum$(\cdot)$ denotes an accumulation function that can be either a vector concatenation or sum operation, $\xi(\cdot)$ is an activation function (Section 2.2), both $W_r \in \mathbb{R}^{d_{\text{hidden}}\times d_{\text{input}}}$ and $W \in \mathbb{R}^{d_{\text{output}}\times d_{\text{hidden}}}$ are learned during training, and d_{input}, d_{hidden}, and d_{output} are the dimensionalities of the input features ($\mathbf{x}$ vectors), encoder hidden layer (4.1) and encoder output layer (4.2), respectively. Each item v_j is also encoded in an analogous way, possibly sharing parameters W_r and W with the user encoder.

The ratings are then predicted using one bilinear decoder per rating value, followed by a softmax operation. The predicted probability of the rating from user u_i with respect to the item v_j being r is:

$$P(\hat{M}_{ij} = r) = \frac{\exp\left(\mathbf{z}_{u_i}^T Q_r \mathbf{z}_{v_j}\right)}{\sum_{s\in\mathcal{R}}\exp\left(\mathbf{z}_{u_i}^T Q_s \mathbf{z}_{v_j}\right)}, \tag{4.3}$$

where each $Q_r \in \mathbb{R}^{d_{\text{output}}\times d_{\text{output}}}$ is learned during training. With both the encoder and decoder defined, we train the model to minimize the negative log likelihood of the predicted ratings:

$$\mathcal{L}(M, \hat{M}) = -\sum_{(u_i,r,v_j)\in\mathcal{E}} \log P(\hat{M}_{ij} = r). \tag{4.4}$$

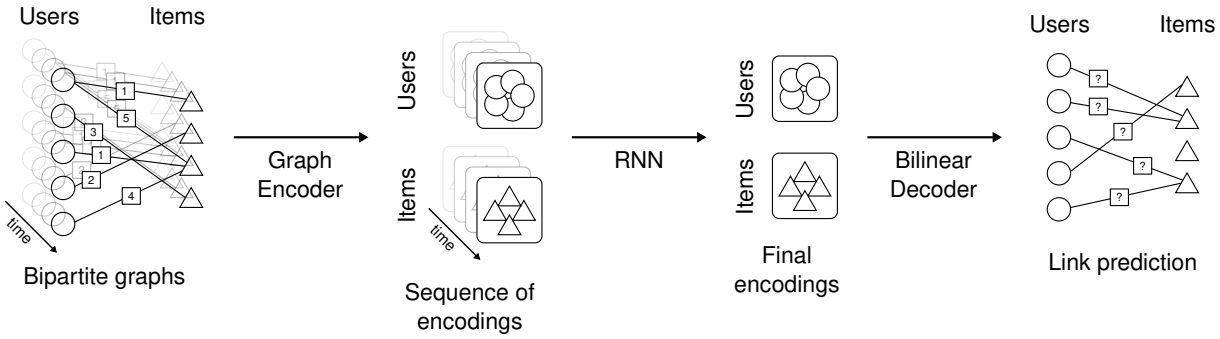


Figure 4.4: Our pipeline for temporal link prediction, inspired by GCMC [3]. The model uses T matrices representing ratings as they happen over time. Those can be either incremental, that is, every new rating is incorporated in the next step, or disjoint, where each rating matrix has approximately the same number of ratings.

4.2.2 Temporal GCMC

Under the framework presented, we view ratings as new edges being added to the dynamic graph, which is our representation of all observed ratings between users and items. In this experiment, we consider two possible representations. For both, we use a history of fixed size T, independent of the total number of ratings. In the first strategy, the dynamic graph is thought of as a sliding "window" over the series of ratings, and we represent each part of the history as a series of disjoint sets of edges where each $M^{(t)}$ contains approximately the same number of edges. In contrast, the second strategy adopts a more explicit representation of rating history, as each $M^{(t+1)}$ contains the new ratings observed so far, plus all ratings observed in the previous time step $M^{(t)}$, with approximately the same number of ratings added at each step.

In order to handle the sequence of matrices, we introduce a new step in the encoding process. First, each $M^{(t)}$ is encoded using the same (shared) encoder, producing the user $\mathbf{z}_{u_i}^{(t)}$ and item $\mathbf{z}_{v_j}^{(t)}$ encodings for each time step t. Following that, a recurrent layer (Section 2.2.2) with d_{hidden} units processes this sequence of user (or item) encodings. Then, the final states of this recurrent layer are used by the decoder to predict the ratings. For simplicity, we adopt the same decoder as GCMC (4.3) since, as already mentioned, we are interested in producing the same kinds of ratings. An overview of this approach is illustrated in Figure 4.4.

4.2.3 Experimental setup

Following the results reported by Berg et al. [3], we use the same number of parameters for both the encoder and decoder, namely $d_{\text{hidden}} = 500$ and $d_{\text{output}} = 75$, and the same weight-sharing scheme for the W_r and Q_r parameters. We also employ dropout ($p = 0.7$) operations before the first hidden layer of the graph encoder (4.1) and immediately after it. All activation functions used are ReLUs, except for the recurrent layer. The accum$(\cdot)$ function we used was a vector concatenation operation. All models were trained using Adam [30] for 1000 epochs using a learning rate of 10^{-2}. We performed experiments both with a LSTM and with a GRU layer.

Dataset

We used the ML-100k dataset [20] (Table 3.1), each user has rated at least 20 movies. This dataset is relevant because it is relatively smaller, allowing more experimentation. In addition, state-of-the-art methods also include results on this dataset, which facilitates comparison. In order to build the dynamic graph, we adopt $T = 10$.

While there is evidence that they can improve recommendations [3], we did not use the feature data for users or movies in order to minimize the influence of other aspects when comparing the performance of models. We are focused on how well they perform using the dynamic graph alone. Hence, we use a 1-hot encoded vector of dimensionality $|\mathcal{W}|$ instead to represent each node.

It should be noted that the test set of the ML-100k dataset contains ratings that happened before some from the training set. Consequently, we define new train, validation, and test sets where we guarantee that every rating in the training set happened before every rating in either the validation or test sets, thus avoiding the use of future ratings to predict past ratings. The test set comprises 20% of the whole dataset (original train and test splits), the validation set is 20% of the remaining data, while the rest is used for training.

Evaluation

We follow the same evaluation protocol found in recently proposed models for the same task [47, 3]. The final predicted rating we use is the expected value of the rating probabilities computed by the model (4.3). It is defined as:

$$\mathbb{E}[\hat{M}_{ij}] = \sum_{r \in \mathcal{R}} r P(\hat{M}_{ij} = r). \tag{4.5}$$

With these values, we compute the root mean square error (RMSE), defined as:

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{E}|} \sum_{(u_i, r, v_j) \in \mathcal{E}} (M_{i,j} - \mathbb{E}[\hat{M}_{i,j}])^2}, \tag{4.6}$$

which is used as a measure of performance, where smaller values represent better results, as this means the predicted ratings are close to the originally observed ratings. Results are computed using the exponential moving average of parameters over every epoch, with a decay of 0.995. All results are the average of 10 distinct weight initializations.

Baselines

We compare the performance of our models to GCMC [3]. Since GCMC attains state-of-the-art results on the dataset we use, it still is a relevant baseline. We note that while methods such as MGCNN [47] could be used for comparison, they make use of a framework very close to GCMC, thus, not adding significant variety to the analysis.

As this is a preliminary experiment, we report results obtained by the models on the validation set, refraining from using the test set to avoid overfitting in the future. Note that we report results for the baseline using both the original training data (with a randomly sampled 20% validation set) and the new training and validation sets.

Table 4.2: RSME values for all models on the ML-100k dataset using the new training and validation sets (RSME values on validation set). The * denotes results on the original ML-100k training set (with a random 20% validation set); † denotes experiments using disjoint edge sets in every $M^{(i)}$; ‡ denotes the representation where every step contains all edges from previous steps.

	RMSE
GCMC*	0.9135 ± 0.0052
GCMC	1.1799 ± 0.0004
GCMC-LSTM†	1.1493 ± 0.0056
GCMC-GRU†	1.1469 ± 0.0054
GCMC-LSTM‡	1.1057 ± 0.0163
GCMC-GRU‡	$\mathbf{1.0927} \pm 0.0060$

4.2.4 Results and discussion

All results are summarized in Table 4.2. At first glance, the most striking result is the difference GCMC displays in performance when the training and validation sets were changed to the new ones. While this might be a sign of overfitting, those results are still useful, since our models using recurrent layers use very similar hyperparameters. This might show how the recurrent layers, along with access to temporal information, might improve even in a setting the original model struggles with.

Moving on to the recurrent models, we see that the disjoint edge representation performs worse on average than its alternative. Furthermore, both LSTM and GRU variants displayed very similar performance. The causes for this might be twofold: (1) either both are underfitting or (2) this representation requires more complex models for a better overview of all ratings, as the model never sees the overall state after the last step. Despite this fact, we see that every recurrent variant outperforms the original GCMC in this setting. While we performed no hyperparameter tuning to avoid overfitting to this dataset at this point, the improvements are still noticeable.

Finally, when using all edges from previous steps at every step, we see performance differences not only comparing the alternative representations, but also between the variants with LSTM and GRU. As the models have the same number of parameters from the previous setting, this suggests that this representation is more promising, since both models perform better with it. Additionally, the better performance shown by the GRU variant can be due to two the fact that LSTMs are more dependent on hyperparameter tuning to achieve better results [19]. As even RSME values for the training set were lower for the GRU variant in this setting, overfitting is not a likely cause.

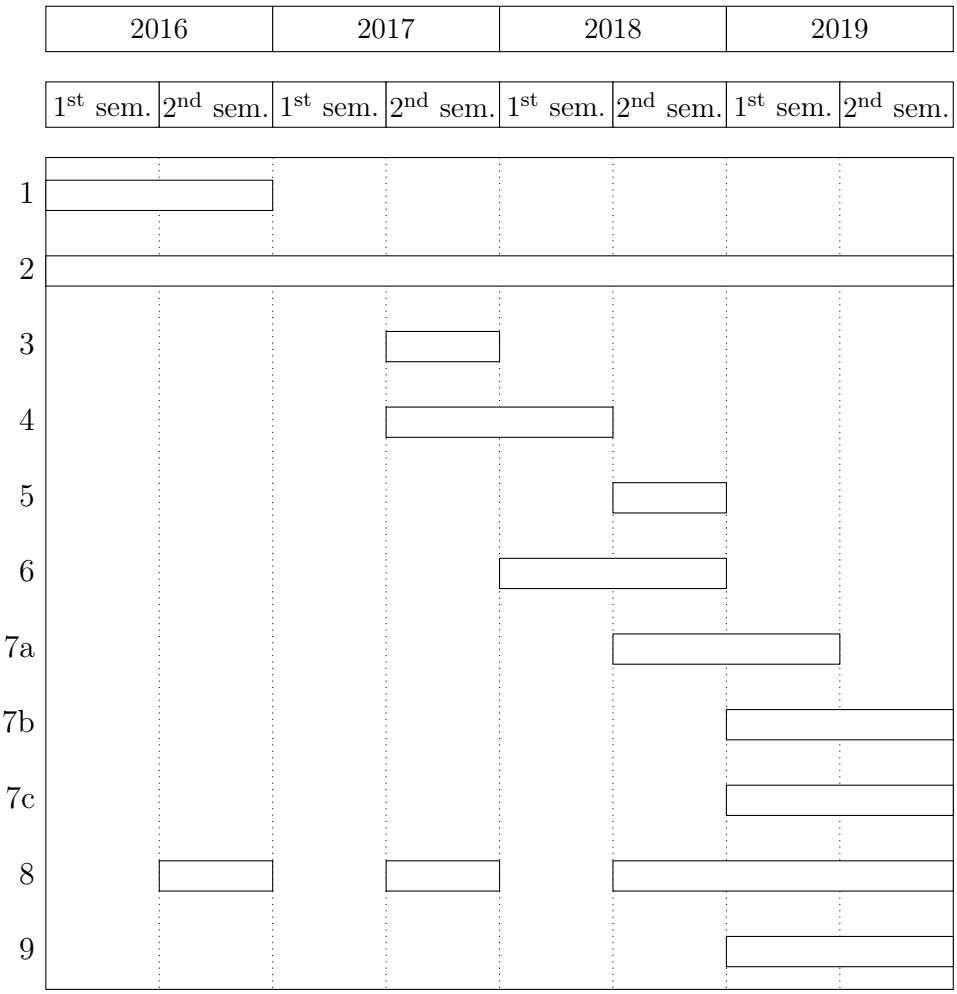
Chapter 5

Schedule and planned activities

The planned activities for this project are listed below, while the schedule for them is summarized in Table 5.1. In particular, we plan to address tasks 7a, 7b, 7c, and 8 in a research internship (BEPE/FAPESP), from October 2018 to October 2019, under the supervision of prof. Ulf Brefeld from Leuphana University, Germany. Prof. Brefeld can contribute significantly to the success of this research, due to experience on sports analytics problems using machine learning, both from a temporal and from a graph representation perspective [8, 35, 9].

1. Fulfillment of course credit requirements;
2. Literature review;
3. Writing and preparation for qualification exam;
4. Teaching internship program (PED);
5. Baselines and comparisons of existing NN for dynamic graphs;
6. Temporal link prediction as recommendation;
7. Soccer match learning with dynamic graphs;
   (a) Data acquisition, preparation, and analysis;
   (b) Retrieval of soccer match clips;
   (c) Match events prediction;
8. Scientific papers, participation in conferences, technical reports;
9. Thesis preparation and writing.

Table 5.1: Scheduled activities.



Bibliography

[1] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. "Relational inductive biases, deep learning, and graph networks". In: (2018). Preprint at `http://arxiv.org/abs/1806.01261`.

[2] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. "Interaction Networks for Learning about Objects, Relations and Physics". In: *Advances in Neural Information Processing Systems 29*. 2016, pp. 4502–4510.

[3] Rianne van den Berg, Thomas N. Kipf, and Max Welling. "Graph Convolutional Matrix Completion". In: (2017). Preprint available at `http://arxiv.org/abs/1706.02263`.

[4] Steven Bird, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[5] Benjamin Bischke, Prakriti Bhardwaj, Aman Gautam, Patrick Helber, D Borth, and A Dengel. "Detection of flooding events in social multimedia and satellite imagery using deep neural networks". In: *Working Notes Proceedings of the MediaEval 2017 Workshop*. Vol. 1984. 2017.

[6] Benjamin Bischke, Patrick Helber, Christian Schulze, Srinivasan Venkat, Andreas Dengel, and Damian Borth. "The Multimedia Satellite Task at MediaEval 2017". In: *Proc. of the MediaEval 2017 Workshop*. Dublin, Ireland, 2017.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[8] Markus Brandt and Ulf Brefeld. "Graph-based Approaches for Analyzing Team Interaction on the Example of Soccer". In: *Machine Learning and Data Mining for Sports Analytics*. 2015.

[9] Ulf Brefeld, Jan Lasek, and Sebastian Mair. "Probabilistic movement models and zones of control". In: *Machine Learning* (2018), pp. 1–21. ISSN: 0885-6125. DOI: `10.1007/s10994-018-5725-1`.

[10] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42. ISSN: 1053-5888. DOI: `10.1109/MSP.2017.2693418`.

[11] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. "Spectral Networks and Deep Locally Connected Networks on Graphs". In: *Proceedings of the International Conference on Learning Representations*. 2013, p. 14.

[12] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274.

[13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Advances in Neural Information Processing Systems 29 (NIPS 2016) pre-proceedings*. 2016, pp. 3837–3845.

[14] Jeffrey L. Elman. "Finding Structure in Time". In: *Cognitive Science* 14.2 (1990), pp. 179–211. DOI: 10.1207/s15516709cog1402_1.

[15] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. "Using collaborative filtering to weave an information tapestry". In: *Communications of the ACM* 35.12 (1992), pp. 61–70. ISSN: 00010782. DOI: 10.1145/138859.138867.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[17] Marco Gori, Gabriele Monfardini, and Franco Scarselli. "A new model for learning in graph domains". In: *Proceedings of the International Joint Conference on Neural Networks*. Vol. 2. IEEE, 2005, pp. 729–734. ISBN: 0780390482. DOI: 10.1109/IJCNN.2005.1555942.

[18] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. "Hybrid computing using a neural network with dynamic external memory". In: *Nature* 538.7626 (2016), pp. 471–476. ISSN: 0028-0836. DOI: 10.1038/nature20101.

[19] Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* (2016), pp. 1–11. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2016.2582924.

[20] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context". In: *ACM Transactions on Interactive Intelligent Systems* 5.4 (2015), pp. 1–19. DOI: 10.1145/2827872.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[22] Ruining He and Julian McAuley. "Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering". In: (2016). Preprint at https://arxiv.org/abs/1602.01585. DOI: 10.1145/2872427.2883037.

[23] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[24] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8.

[25] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015, pp. 448–456.

[26] Ashesh Jain, Amir R. Zamir, Silvio Savarese, and Ashutosh Saxena. "Structural-RNN: Deep Learning on Spatio-Temporal Graphs". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. eprint: 1511.05298.

[27] Michael I. Jordan. "Attractor dynamics and parallelism in a connectionist sequential machine". In: *Artificial neural networks*. Piscataway, NJ, USA: IEEE Press, 1990, pp. 112–127. ISBN: 0818620153.

[28] Brendan Jou and Shih-Fu Chang. "Deep Cross Residual Learning for Multitask Visual Recognition". In: *Proceedings of the 2016 ACM on Multimedia Conference - MM '16*. New York, New York, USA: ACM Press, 2016, pp. 998–1007. ISBN: 9781450336031. DOI: 10.1145/2964284.2964309.

[29] Vassilis Kalofolias, Xavier Bresson, Michael Bronstein, and Pierre Vandergheynst. "Matrix Completion on Graphs". In: *NIPS 2014 workshop "Out of the Box: Robustness in High Dimension"*. Preprint at http://arxiv.org/abs/1408.1717. 2014.

[30] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (2014). Preprint at http://arxiv.org/abs/1412.6980.

[31] Gabriela Sarti Kinker, Andrew Maltez Thomas, Vinicius Jardim Carvalho, Felipe Prata Lima, and André Fujita. "Deletion and low expression of NFKBIA are associated with poor prognosis in lower-grade glioma patients". In: 6 (2016), p. 24160. DOI: 10.1038/srep24160.

[32] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *arXiv preprint* (2016). eprint: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[33] Thomas N. Kipf and Max Welling. "Variational Graph Auto-Encoders". In: (2016). Preprint at http://arxiv.org/abs/1611.07308.

[34] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. "Self-Normalizing Neural Networks". In: *Advances in Neural Information Processing Systems 30*. 2017, pp. 971–980.

[35] Konstantin Knauf, Daniel Memmert, and Ulf Brefeld. "Spatio-temporal convolution kernels". In: *Machine Learning* 102.2 (2016), pp. 247–273. ISSN: 0885-6125. DOI: 10.1007/s10994-015-5520-1.

[36] Y. Le Cun, L.D. Jackel, B. Boser, J.S. Denker, H.P. Graf, I. Guyon, D. Henderson, R.E. Howard, and W. Hubbard. "Handwritten digit recognition: applications of neural network chips and automatic learning". In: *IEEE Communications Magazine* 27.11 (1989), pp. 41–46. ISSN: 0163-6804. DOI: 10.1109/35.41400.

[37] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539.

[38] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. "Graphs over time". In: *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD '05*. New York, New York, USA: ACM Press, 2005, p. 177. ISBN: 159593135X. DOI: 10.1145/1081870.1081893.

[39] Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. "Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, 2015, pp. 1520–1530.

[40] Linyuan Lü and Tao Zhou. "Link prediction in complex networks: A survey". In: *Physica A: Statistical Mechanics and its Applications* 390.6 (2011), pp. 1150–1170. ISSN: 0378-4371. DOI: 10.1016/J.PHYSA.2010.11.027.

[41] Ulrike von Luxburg and Bernhard Schoelkopf. "Statistical Learning Theory: Models, Concepts, and Results". In: (2008). Preprint at http://arxiv.org/abs/0810.4752.

[42] Julian McAuley and Jure Leskovec. "From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews". In: (2013). Preprint at http://arxiv.org/abs/1303.4402.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space". In: (2013). Preprint available at http://arxiv.org/abs/1301.3781.

[44] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. "Advances in Pre-Training Distributed Word Representations". In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[45] Tom Mitchell. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education, 1997. ISBN: 9780070428072.

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. ISSN: 0028-0836. DOI: 10.1038/nature14236.

[47] Federico Monti, Michael Bronstein, and Xavier Bresson. "Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks". In: *Advances in Neural Information Processing Systems 30*. 2017, pp. 3697–3707.

[48] M. E. J. Newman. "The Structure and Function of Complex Networks". In: *SIAM Review* 45.2 (2003), pp. 167–256. ISSN: 0036-1445. DOI: 10.1137/S003614450342480.

[49] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. "WaveNet: A Generative Model for Raw Audio". In: (2016). Preprint at http://arxiv.org/abs/1609.03499.

[50] Javier López Peña and Hugo Touchette. "A network theory analysis of football strategies". In: (2012). Preprint available at http://arxiv.org/abs/1206.6904.

[51] Daniele C. Uchoa Maia Rodrigues, Felipe A. Moura, Sergio Augusto Cunha, and Ricardo da S. Torres. "Visualizing Temporal Graphs using Visual Rhythms - A Case Study in Soccer Match Analysis". In: *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017) - Volume 3: IVAPP, Porto, Portugal, February 27 - March 1, 2017.* 2017, pp. 96–107. DOI: 10.5220/0006153000960107.

[52] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 0028-0836. DOI: 10.1038/323533a0.

[53] Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. "A simple neural network module for relational reasoning". In: (2017). arXiv: 1706.01427. URL: http://arxiv.org/abs/1706.01427.

[54] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. ISSN: 1045-9227. DOI: 10.1109/TNN.2008.2005605.

[55] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. "Modeling Relational Data with Graph Convolutional Networks". In: (2017). Preprint at http://arxiv.org/abs/1703.06103.

[56] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003.

[57] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. "Structured Sequence Modeling with Graph Convolutional Recurrent Networks". In: (2016). Preprint at http://arxiv.org/abs/1612.07659.

[58] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: *Advances in Neural Information Processing Systems 28*. 2015, pp. 802–810.

[59] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Bradford Book, 1998. ISBN: 9780262193986.

[60] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 1–9. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298594.

[61] Bart Thomee, Benjamin Elizalde, David A. Shamma, Karl Ni, Gerald Friedland, Douglas Poland, Damian Borth, and Li-Jia Li. "YFCC100M: the new data in multimedia research". In: *Communications of the ACM* 59.2 (2016), pp. 64–73. ISSN: 00010782. DOI: 10.1145/2812802.

[62] Didier A Vega-Oliveros, Luciano da F Costa, and Francisco A Rodrigues. "Rumor propagation with heterogeneous transmission in social networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2017.2 (2017), p. 023401. ISSN: 1742-5468. DOI: 10.1088/1742-5468/aa58ef.

[63] Maciel C. Vidal, João R. Sato, Joana B. Balardin, Daniel Y. Takahashi, and André Fujita. "ANOCVA in R: A Software to Compare Clusters between Groups and Its Application to the Study of Autism Spectrum Disorder". In: *Frontiers in Neuroscience* 11 (2017), p. 16. ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00016.

[64] Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li. "Deep Learning for Content-Based Image Retrieval: A Comprehensive Study". In: *Proceedings of the ACM International Conference on Multimedia - MM '14*. MM '14. New York, NY, USA: ACM, 2014, pp. 157–166. ISBN: 9781450330633. DOI: 10.1145/2647868.2654948.

[65] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. "Embedding Entities and Relations for Learning and Inference in Knowledge Bases". In: (2015). Preprint at http://arxiv.org/abs/1412.6575.

[66] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems". In: (2018). Preprint at http://arxiv.org/abs/1806.01973. DOI: 10.1145/3219819.3219890.

[67] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R. Salakhutdinov, and Alexander J. Smola. "Deep Sets". In: *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. 2017, pp. 3391–3401.