Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITE GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Yagmur Gizem CINAR

Thèse dirigée par **Eric GAUSSIER, PREX, Université Grenoble Alpes**

préparée au sein du **Laboratoire Informatique de Grenoble** dans **l'École Doctorale MSTII**

Prédiction de Séquences basée sur des Réseaux de Neurones Récurrents dans le Contexte des Séries Temporelles et des Sessions de Recherche d'Information

Thèse soutenue publiquement le **Février 2019** ,
devant le jury composé de :

Lynda TAMINE-LECHANI
Professeure des Universités, Université Paul Sabatier, Rapportrice
Patrick GALLINARI
Professeur des Universités, Sorbonne Université, Rapporteur
Fabio CRESTANI
Professeur, Università della Svizzera Italiana, Examinateur
Vadim STRIJOV
Professeur, Moscow Institute of Physics and Technology, Examinateur
Ahlame DOUZAL
Maître de Conférences**,** Université Grenoble Alpes, Examinatrice
Julien PEREZ
Scientifique Senior, Naver Labs Europe, Invité
Eric GAUSSIER
PREX, Université Grenoble Alpes, Directeur de thèse

Acknowledgements

Firstly, I would like to thank my PhD thesis supervisor Prof. Eric Gaussier for his guidance and support during my thesis. I am fortunate to be his student, and have his infinite support and invaluable guidance. I am very grateful for the precious knowledge and experience that I obtained under his supervision. He is a great researcher and an excellent person.

I would like to thank the reporters of my PhD thesis, Prof. Patrick Gallinari and Prof. Lynda Tamine for their assessment, scientific feedback and insightful comments. I would like to also thank, the rest of my defense jury, Prof. Fabio Crestani, Prof. Vadim Strijov, Prof. Ahlame Douzal, and Dr. Julien Perez for being part of my jury and their enriching scientific discussions.

I would like to thank Dr. Hamid Mirisaee and Dr. Parantapa Goswami for his scientific contributions to my research with their collaborations. Their collaborations and scientific discussions were very enriching.

I would like to express my gratitudes to my previous supervisors during my studies of master's and bachelor's: Prof. Marie-Francine Moens, Prof. Tinne Tuytelaars, and Prof. Mujdat Cetin, and mentors during my internships, Prof. Metin N. Gurcan, Prof. Aytul Ercil, and Dr. Ceyhun Burak Akgul for the precious knowledge that I obtained under their supervision.

I would like to also thank to (previous and current) members of the AMA team, for the friendly environment, scientific discussions and their encouragement and support.

I would like to thank my friends for their invaluable support and encouragement. I would like to especially thank to those endure long distance.

I would like to also thank Mathieu for his endless support and encouragement.

This thesis is dedicated to my parents and family. I am very grateful for their endless love and support. I am very fortunate to have such a great family, I could only go through all these challenge with their love and support.

Abstract

This thesis investigates challenges of sequence prediction in different scenarios such as sequence prediction using recurrent neural networks (RNNs) in the context of time series and information retrieval (IR) search sessions. Predicting the unknown values that follow some previously observed values is basically called sequence prediction. It is widely applicable to many domains where a sequential behavior is observed in the data. In this study, we focus on two different types of sequence prediction tasks: time series forecasting and next query prediction in an information retrieval search session.

Time series often display pseudo-periods, i.e. time intervals with strong correlation between values of time series. Seasonal changes in weather time series or electricity usage at day and night time are some examples of pseudo-periods. In a forecasting scenario, pseudo-periods correspond to the difference between the positions of the output being predicted and specific inputs.

In order to capture periods in RNNs, one needs a *memory* of the input sequence. Sequence-to-sequence RNNs (with attention mechanism) reuse specific (representations of) input values to predict output values. Sequence-to-sequence RNNs with an attention mechanism seem to be adequate for capturing periods. In this manner, we first explore the capability of an attention mechanism in that context. However, according to our initial analysis, a standard attention mechanism did not perform well to capture the periods. Therefore, we propose a period-aware content-based attention RNN model. This model is an extension of state-of-the-art

sequence-to-sequence RNNs with attention mechanism and it is aimed to capture the periods in time series with or without missing values. Our experimental results with period-aware content-based attention RNNs show significant improvement on univariate and multivariate time series forecasting performance on several publicly available data sets.

Another challenge in sequence prediction is the next query prediction. The next query prediction helps users to disambiguate their search query, to explore different aspects of the information they need, or to form a precise and succint query that leads to higher retrieval performance. A search session is dynamic, and the information need of a user might change over a search session as a result of the search interactions. Furthermore, interactions of a user with a search engine influence the user's query reformulations. Considering this influence on the query formulations, we first analyze *where the next query words come from?* Using the analysis of the sources of query words, we propose two next query prediction approaches: a set view and a sequence view.

The set view adapts a bag-of-words approach using a novel feature set defined based on the sources of next query words analysis. Here, the next query is predicted using learning to rank. The sequence view extends a hierarchical RNN model by considering the sources of next query words in the prediction. The sources of next query words are incorporated by using an attention mechanism on the interaction words.

Contents

List of Figures

List of Tables

Publications

Peer-reviewed Journal Article

Y. G. Cinar, H. Mirisaee, P. Goswami, E. Gaussier, A. Ait-Bachir. (2018). Period-aware Content Attention RNNs for Time Series Forecasting with Missing Values. *Neurocomputing, 312*, 177-186. doi:10.1016/j.neucom.2018.05.090

Y. G. Cinar, P. Goswami, H. Mirisaee, E. Gaussier. Predicting Next Queries in Sessions: a Study on the TREC Search Session Track. (submitted to Information Retrieval Journal).

Peer-reviewed International Conference Article

M. Caneill, N. De Palma, A. Ait-Bachir, B. Dine, R. Mokhtari and Y. G. Cinar. (2018). Online Metrics. Prediction in Monitoring Systems. *8th International Workshop on Big Data and Cloud Performance* (DCPerf) at IEEE INFOCOM.

Y. G. Cinar, H. Mirisaee, P. Goswami, E. Gaussier, A. Ait-Bachir, V. Strijov. (2017). Position-based Content Attention for Time Series Forecasting with Sequence-to-Sequence RNNs. *International Conference on Neural Information Processing* (ICONIP).

Chapter 1

Introduction

1.1 Predicting Sequences

Many sources of data have a sequential structure in which observations are organized in a certain order. Order in the sequence constitutes an important aspect of the data. Sequence of words in a sentence, sequence of financial transactions, pitches of a speech, frames of a video, and values of a time series are typical examples of sequences of observations.

This thesis focuses on sequence prediction in the context of time series forecasting and information retrieval (IR) search session. Sequence prediction is an interesting topic in many aspects in various applications. For example, predicting accurately electricity consumption helps both consumers, producers and environment. More generally, it could be applied to any resource and product consumption prediction to avoid waste of natural resources and wrong investment of energy and time. It can also be used to increase productivity by predictive assistance in user activities, e.g. by automatically completing an email, message, query in a search session or by generating ambient music based on the time of the day, environment, or activity. Sequence prediction could be also used for a social project by predicting the expected number of refugees from a specific destination to certain parts of the world. This would enable to redirect the refugees according to the matching between local needs and theirs, *e.g.*, in terms of job/labour force.

Depending on the specific sequence prediction application in literature different machine learning approaches are adapted, *e.g.*, random forests (RFs) [107], support vector machines (SVMs) [147, 166], hidden Markov models (HMMs) [138, 87, 191] and recurrent neural networks [61, 72, 183]. Random forests and support vector machines do not explicitly model time (order in sequence) and they cannot represent the time dependence in data that is required in many applications e.g. in a dialogue system, self-driving cars, question answering, etc. To explicitly represent the sequential dependencies in a sequence, one might consider using HMM approaches [180]. Here, each hidden state only depends on the previous hidden state ($P(s_t|s_{t-1}) = P(s_t|s_{t-1}, ..., s_1)$). Hence, to be able to represent a longer window of dependence one can create a new state as product of all possible states at each time in the window which would grow exponentially with the size of window and not be practical for modeling long-range dependencies [144, 130, 76].

Recurrent neural networks capture dynamics of the sequence – information throughout the sequence – with a cyclic connection. RNNs pass the information through the sequence via recurrent states and process the sequence by one element at a time. The input at step t is the current input x_t and the previous recurrent hidden state h_{t-1}; the recurrent hidden state at step t is then defined as $h_t = f(x_t, h_{t-1})$. Here, the recurrent hidden state can carry information over a nearly arbitrary length window and represent sequential dependencies on multiple scales. The number of distinct states of RNNs, which corresponds to their expressive power, increases exponentially with the number of nodes in one recurrent layer, but training and inference complexity increases at most quadratically [130]. RNNs of a fixed size are able to approximate and simulate functions of sequences up to a fixed length [85]. Another interesting property of RNNs is that they are parametric methods, not instance-based methods – they do not implement template-based matching of training data to make predictions. With their predictions they do not suffer from curse of dimensionality and better model data than exact matching approaches [75].

Furthermore, a recent RNN architecture for sequence prediction which received considerable research interest is sequence-to-sequence, also called encoder-decoder, RNNs, initially proposed for machine translation (in [183, 36] and further described

in Sections 2.4.3). Bahdanau et al. [8] added an alignment unit – attention mechanism – to align the input values to output predictions. Attention mechanisms gained state-of-the-art results in sequence modelling tasks, *e.g.*, machine translation [194]. Here, the attention mechanism acts as a memory unit by re-accessing, reusing specific (representations of) input values to predict output values. Attention mechanism gives access to all the input sequence during output prediction and relaxes the assumption (dependence) of the the recurrent unit of RNNs to represent all the necessary information in history. Furthermore, attention mechanism is considered to improve gradient flow during the training: here, attention mechanism in a way acts like skip connections. All of these properties, thus, make sequence-to-sequence networks with attention an interesting approach for sequence learning.

In this thesis we are particularly interested in two types of sequence prediction scenarios: first, time series forecasting and second, next query prediction in an information retrieval (IR) search session. The following section describes the research questions addressed in this thesis, and summarizes our contributions in these two sequence prediction tasks.

1.2 Research Questions and Contributions

Here, we present the research questions and our contributions to address these questions for two sequence prediction tasks: time series forecasting in Section 1.2.1 and next query prediction in Section 1.2.2.

1.2.1 Time Series Forecasting

Time series are sequential numerical values ordered by time and time series forecasting is about predicting future values of such a temporal variable. Time series forecasting was extensively studied in the literature [70, 123]. Time series forecasting is important for optimal decision making in many fields: e.g. forecasting the temperature and precipitation enables better organization of outdoor events, or similarly, forecasting product consumption enables adaptation of business plan decisions [201].

Financial time series forecasting can be beneficial for business and individual interest [117]. Predicting the demand of a product [159] is favorable for the consumer, retailer and producer: it prevents wasting resources/material used in the production, and it ensures having enough reserve for the demand of consumers. It is especially useful for essential products such as basic food and the end products whose shelf lives are shorter than their constituents', *e.g.*, some medical products. Predictive analysis also leads to efficient cloud computing capacity and demand management [68], as well as efficient thermal management [127].

Time series forecasting enables one to anticipate forthcoming circumstances by using precautious signals of future forecasting. One example is estimating consumption of an electricity source during a winter period. Here, predicting accurately both the temperature and precipitation during the winter season would enable a local responsible to react and plan in advance to accommodate the adequate amount of electricity need without waste of resources or shortage of electricity. Accurately forecasting weather conditions for more distant future would also improve agriculture outcome since the result of harvest in each year is highly influenced by weather conditions and climate [86]. Furthermore, an unexpected earthquake can destroy a whole city and might lead to many losses; here, seismic time series forecasting could be life-saving and might even avoid the economical loss caused by an earthquake [160, 146].

The time series forecasting task was extensively studied by using various stochastic [70] and machine learning based [19] approaches. A particular class of approaches that has recently received much attention for modeling sequences is based on sequence-to-sequence Recurrent Neural Networks (RNNs) [75, 183, 8]. With their modeling capabilities, sequence-to-sequence RNNs are well adapted to time series forecasting when the task is formulated as predicting future values on the basis of a sequence of past values. Furthermore, they achieve state-of-the-art results in different applications (as machine translation [8, 136, 24] or image captioning [200]). Here we investigate methods that can be used to adapt sequence-to-sequence RNNs with attention mechanism to general time series prediction tasks.

One can also analyze the time series according to the number of sources: observations originating from one single source are called univariate and multiple related observations from different sources are called multivariate time series. For instance, the temperature variable is closely related to the consumption of electricity as the usage of air conditioners is increased during a hot day or the usage of electrical heating is increased during a cold day. It was also observed that the temperature affects the demand of certain items, *e.g.*, water [159]. Many time series, univariate or multivariate, exhibit *pseudo-periods*: time intervals across which there is a strong correlation, *positive or negative*, between the values of the time series. For example, pseudo-periods can be due to environmental factors such as seasonality or to the patterns underlying the activities measured. The workload on professional email servers has both weekly and daily periods: a higher workload during the working hours of a week day and a lower workload outside of working hours. Similarly, more email activities occur during work days and much less during the weekend. This is not specific to the workload of a professional email server. In fact, many products and resources display daily and weekly periods, depending on their consumers' routine, *e.g.* their working hours. The electricity consumption of a region or a country during the day or the number of customers to serve in a restaurant located in a work-site are some examples of time series exhibiting pseudo-periods. Throughout this thesis, the term period may refer to either a true period or a pseudo-period.

When forecasting sequence of future values based on the basis of a sequence of past values, pseudo-periods represent the difference between the positions of the predicted output and specific inputs. Periods of a time series is an important element in efficient forecasting, since one might observe similar behaviour between two points, which corresponds to a period. Hence, it is interesting to investigate whether a point in the history would be useful in the forecasting of a point in the future by using the periodicity information. In this respect, the following question, which is addressed in this thesis, arises: *Can sequence-to-sequence RNNs model periods in time series?*

Periods in time series using RNNs can be captured by having a mechanism to reuse specific (representations of) input values – a *memory* of the input sequence – to predict output values. For that purpose, the sequence-to-sequence model with an

attention mechanism proposed in [8] and described in Section 2.4.3 is particularly well suited. In addition, longer-term memories that store information pertaining to past input sequences (as proposed in, *e.g.* [207, 206, 77] and described in Section 2.4.3) are not required considering the input sequence (of time series) is usually longer than the underlying periods in time series. This model allows one to reuse the content of the input sequence to predict the output values by adding a mechanism which calculates an alignment score between input sequence values and the output value to predict. Although this model was proposed for machine translation, and it can be relevant to investigate its capacity to capture position-based periods in time series.

Time series are mostly recorded by a fixed-interval sensor which results in a fixed sampling rate (discretization) of the time series. For multivariate time series, underlying variables have different sampling rates. The differences in the recording time interval lead to a multi-scale time series. Furthermore, lost records and/or mistakes during data entry or due to faulty sensors, which appears often in a real time series dataset, might leads to missing values or intervals of missing values – gaps. Standard approaches deal with missing values by explicitly estimating or inferring them, *e.g.* interpolation and data imputation methods. After applying these approaches, standard forecasting techniques can be used. In the context of RNNs, one can repeat the hidden state of the last observed value over the missing values, which is called padding.

The missingness in the data creates a challenge for time series forecasting. For real time series forecasting applications, it is important to have a time series forecasting model robust to missing values and gaps. In addition, a periodic time series might have some missing values or gaps. However, these approaches make strong assumption on the missing data, especially for long gaps. Hence, we secondly address the following question: *Do sequence-to-sequence RNNs handle well missing values?*

The main contributions of this work address the above two questions and are based on:

- A complete and general framework for time series forecasting by using a period-aware content attention mechanism;

- An experimental study showing the incapability of RNNs in detecting pseudo-periods and periods in time series;
- Several new models and architectures to efficiently handle missing values and gaps in time series. These models are combined with the period-aware content attention mechanism mentioned above and yield state-of-the-art performances on several benchmark data sets.

1.2.2 Next Query Prediction in an IR Search Session

Information retrieval query prediction, also known as query suggestion or recommendation, was extensively studied in the literature [6, 150, 165, 178]. An IR search session is composed of the consecutive interactions of a user with a search engine. Next query prediction in a search session enables users to explore different aspects of the information they need [6]. It helps users to disambiguate their search query, to explore different aspects of their information need or to form a precise and succint query that leads to higher retrieval performance. One approach adapted for query prediction to accommodate users is leveraging *wisdom of crowds* through the analysis of IR search logs: records of users' search actions with a search engine. For some specific information need or search tasks, users might need to be acquainted with a specific terminology on the search topic. Or at a point in an IR search session – a sequence of queries one user issues for a specific search task which might be heuristically defined in a certain time frame – the information need of a user might be evolved based on the information gathered during earlier search interactions with the search engine. Here, a user might have little knowledge on this topic, can be an expert or might not be knowing *what to search for*. One particular use of next query prediction is to suggest users potentially helpful queries during their search session.

One can also use next query prediction to simulate a search session. By simulating a search session, one can augment information retrieval datasets by expanding from single query to a session of queries. It was showed that using session information improves the retrieval performance compared with a single query information retrieval

[30]. Hence, these datasets, which are extended to a session of queries from a single query, can be used to learn IR systems adapted to sessions.

A relatively complex information need require users to formulate multiple queries to cover all the aspects of their search, *e.g.* to prepare a touristic trip to a foreign city or learn about a Swahili dish. A user typically starts with a first query and gradually modifies the query over the search session. Interactions of a user with an IR search engine are dynamic. The information need of a user might be also altered during their search session. For instance, a user might not be an expert of the field, she might grasp the field-specific search words or an acquired information during her search might arouse a different information need. For example, a search engine user travelling to Paris for touristic reasons might have heard about the Eiffel tower beforehand and might be initially interested in learning more about the Eiffel tower. During her search session for her trip, this user could possibly come across with other touristic attractions in Paris, *e.g.*, Sacré-Cœur by reading "At the top of the Montmartre hill, the Sacré-Cœur Basilica (Sacred Heart), provides truly breathtaking views of the capital and the Eiffel Tower." [1]. Following this information, this user might formulate a query to explore more about Sacré-Cœur and Montmartre. Queries for this example might look like the following: ["Paris Eiffel Tower", "Paris Eiffel Tower visiting hours", "Eiffel tower best viewpoints", "Sacré-Cœur","Montmartre"].

Furthermore, a user might not know what to search for. For instance, a user might search for a disease by typing some symptoms of this disease without knowing the name of this disease. During the search session, the user can learn the name of the disease and reformulate her query by using this disease name.

The interactions with the search engine has a significant influence on the formulations of the users' queries. A complete session data is composed of several consecutive interactions, where each interaction contains a query, a set of retrieved documents (with titles and snippets visible to the user) and a (potentially empty) set of clicked documents. Considering the potential influence of search interactions on the query formulations, we first analyze the session search data: (i) the general statistics of search session dataset (*e.g.* how the query length evolves through a session), (ii) the

evolution of topic through a search session, (iii) the sources of next query words, *i.e where the next query words come from.*

Our goal here is to assess whether one can predict, at any point in a session, the next query using the information collected from the previous interactions, namely previous queries, retrieved documents, snippets of retrieved documents, and clicked documents in this session. We propose here two methods to predict next queries in sessions: set view and sequence view approaches. The set view adapts a bag of words representation of previous word interactions and predicts next queries using learning to rank. The sequence view extends an hierarchical encoder-decoder RNN approach of query recommendation (proposed in [178], described in Section 2.6) approach by taking into account the sources of the next query words. Here, to incorporate the sources of the next query words, we add an attention mechanism on top of the previous interaction words.

The main contributions of this work are listed as follows:

- Utilizing a complete search session information including previous queries, retrieved documents, snippets of retrieved documents and clicked documents in the session;
- Analysis of the next query words' source in a search session;
- A novel feature set is defined using the analysis of the sources of query words;
- An extension of the hierarchical encoder-decoder RNN for sequence prediction architecture by considering the sources of words in the prediction of the next query.

1.3 Thesis Overview

This thesis is organized as follows:

Chapter 2 provides necessary background on sequence prediction. It introduces hidden Markov models, Markov decision processes, neural networks, recurrent neural networks, sequence-to-sequence recurrent neural networks and attention mechanism concepts as well as the literature of sequence learning using these concepts (Sections 2.1-Section 2.4). Following the description of these learning approaches, we provide the background on the two sequence prediction applications, time series forecasting (Section 2.5) and next query prediction (Section 2.6). The evaluation of time series forecasting and next query prediction are given in Section 2.5.1 and Section 2.6.1, respectively.

Chapter 3 investigates sequence prediction in the context of time series forecasting. It describes the proposed period-aware content attention RNN model to capture periods and to handle missing values for univariate and multivariate time series forecasting (Sections 3.3).

Chapter 4 presents sequence prediction in the context of IR search sessions – next query prediction. In this chapter, we analyze the search sessions, especially where the query words come from in a search session (Section 4.3). Later, based on the search session analysis we propose two next query prediction approaches: a set view model (Section 4.4.1) and a sequence view model (Section 4.4.2).

In Chapter 5, we present the conclusions and perspectives of this thesis on time series forecasting (Section 5.1) and IR session search next query prediction (Section 5.2).

Chapter 2

Background

This chapter provides an overview of related literature on sequence prediction tasks in the context of time series and IR search sessions. Section 2.1 introduces the sequence prediction task. Section 2.4 starts with describing briefly neural networks (Section 2.4.1), the evolution of recurrent neural networks (Section 2.4.2) and ends with some recent advancements in recurrent neural networks (Section 2.4.3) which constitute the basis of the models proposed in Chapter 3 and 4. Section 2.5 reviews earlier work of stochastic and machine learning models on time series forecasting. Here, we focus on machine learning models, more specifically RNNs that underlie the contributions of this thesis. Section 2.6 covers the related literature of the IR next query prediction task and recent approaches with recurrent neural networks as well.

2.1 Sequence Prediction

Sequence prediction consists in predicting the next symbol or value based on the previous symbols or values, which is a subtask of sequence learning [182]. Sequence prediction is a broad task applicable in many domains: healthcare, finance, text, video, aerospace, etc. Some sequence prediction applications can be listed as time series forecasting, text generation, text summarization, next query prediction in a search session, next item recommendation in a recommender system, user action prediction, next frame prediction in a video, speech generation. The main sequence prediction

approaches directly modelling the sequences are Hidden Markov Models (HMMs), Markov Decision Processes (MDPs) and Recurrent Neural Networks (RNNs). As discussed earlier in Chapter 1 considering the representational limitations of long-range dependency of HMMs computation, we adopted RNNs in our contributions.

2.2 Hidden Markov Models

In the beginning of 1900s, Andrei Andreyevich Markov studied a process which models the influence of an outcome on the next outcome, which is called Markov chain. Markov chain is a simple model of the stochastic changes in an environment (which includes the events one wishes to model). Hidden Markov Models (HMMs) define a Markov chain on some unobserved variables named hidden states on which depend the observations. They are generative sequence models, a particular type of Bayesian networks [65]. They model a probability distribution over sequences of observations. The HMM model assumes that the state generating the observation is hidden from the observer [153]. At each time step an output of this hidden process is observed while the hidden state remains latent. The second assumption is that the state and the output of HMMs satisfy the Markov property: the current state S_t depends only on the previous state S_{t-1} and is independent of all the preceding states. Hence, the state at a time captures all the history to predict the future of the process. Third, the hidden state variable is discrete: S_t can take on a finite number of values, $\{1, ..., |S_{\text{HMM}}|\}$. The state transition matrices of HMMs, which define the probability distributions, are usually time invariant – they do not depend on t. Input variables can be added to the HMMs, in [14, 141] an input-dependent state transition probability is defined.

HMMs were extensively applied in many fields: natural language processing [138, 105, 144, 225, 192, 191], computational biology [9, 116], bioinformatics [10] and time series forecasting [87, 35].

With the discrete state space assumption, the capacity of HMMs hidden states are limited [144, 123]. HMMs require $|S_{\text{HMM}}| = 2^N$ hidden states to represent N bits of history which is impractical for many applications.

2.3 Markov Decision Processes

Markov Decision Process (MDP) is a mathematical model of sequential decision making [13, 184]. An MDP is composed of the set of all possible states, the set of actions that an agent might take, the transition probability function which refers to the transition probability to the next state given the current state and the action taken on this current state, and the reward function. The reward function is a type of performance criterion. In an MDP setting an agent observes the state and take an action and receives a reward until it reaches a terminal state. A policy defines how an agent acts given a state, and it can be a deterministic function or a stochastic distribution over actions. In Markov decision processes, states are assumed to be fully observable. In case of partial observation, partially observable Markov decision processes (POMDP) are used [106].

MDP can be solved by using linear or dynamic programming, e.g. the value iteration [13] or policy iteration [94], when the transition and reward functions are known and there is a finite number of states and actions. Q-learning [204, 185] or policy gradient [186] can be used to approximate MDP models based on experience when the model is unknown.

Markov decision processes are mainly applied for sequential decision making, sequence generation and prediction tasks in many research domains: finance [209, 28], water resources [219, 133] and communication networks [179, 3] (further application domains are detailed in [208]).

2.4 Modelling Sequences with Neural Networks

2.4.1 Neural Networks

Neural networks (NNs) are initially proposed to mathematically model a biological brain learning mechanism [139, 162]. NNs without feedback loop are also called feedforward neural networks (FNNs): a feedforward network with one layer is called a perceptron [161] and with multiple layers is called a multi-layer perceptron (MLP)

[163]. Later, it was shown that they are universal approximators, i.e., they can approximate any continuous function, if the activation function is continuous and not a polynomial function [93, 125]. The universal approximation theory for MLPs does not provide any bounds on the size of the network. However, [11] studied some bounds on the size of a single layer network. In addition, NNs are less sensitive to the curse of dimensionality issues since the approximation error is independent of input signals dimension [11]. These theoretical properties make NNs an interesting learning approach and this latter study suggests there is an existing solution for any function using NNs. In practice, one needs to train a network to learn parameters leading to this solution. Indeed, the universal approximation theory does not provide the parameters leading to a solution.

The parameters of a neural network can be learned though error propagation which is called back-propagation by calculating partial derivatives from loss/objective function with respect to the network weights of the successive layers [163]. Recently many different gradient descent algorithms have been proposed to learn network parameters: Adagrad [48], Adadelta [227], RMSprop [190], Adam [112].

2.4.2 Recurrent Neural Networks

Recurrent neural networks are neural networks with a cyclic (feedback) connection which aims to capture dynamics of sequences [163, 121, 122, 49, 103]. The most used cyclic connection is a recurrent connection between hidden states [69]. Recurrent neural networks share parameters across time dimension which enables them to process variable length of sequences and relaxes the pre-defined fixed length input requirement of feed-forward neural networks. An RNN applies the same recurrent function on each element of input sequence from the first element to the last one while capturing the dynamics of sequence through time. This recurrent function takes typically two inputs: the element in the sequence at that time step and the previous hidden state (recurrent connection between hidden states). RNN hidden states are also called recurrent states.

Recurrent neural networks are able to approximate and simulate functions of sequences. [172, 171, 173] show that RNNs are Turing-Complete: a finite sized recurrent neural network (with a hidden-to-hidden recurrent connection) can simulate any Turing machine function. An RNN of a fixed size can approximate sequences up to a fixed length [85]. Having a large enough RNN ensures to have the capability to approximate sequences for a desired task.

Parameters of an RNN can be learned though an error propagation from the loss function to the inputs. Here the same chain rule applies as feed-forward networks with the extension in the dimension of time. The major algorithm to learn parameters of RNNs is hence named back-propagation through time (BPTT) [205, 213].

In practice standard (vanilla) RNNs cannot store information about past inputs occurred many time steps earlier [91]. Prediction only based on recent past is not sufficient in many applications, e.g. forecasting of a time series with a yearly period or automatic long text summarization. Many researchers independently studied the vanishing gradient problem with vanilla RNNs [90, 15, 91]. Because of the vanishing gradient problem, standard RNN cannot capture the dependencies longer than 10 - 20 timesteps [91, 69] (see Figure 2.1). Recurrent connection relation – in the simplest case possible without an input and non-linearity – can be described by the power method [69] in which the hidden state at time t is simply represented by a matrix multiplication of the recurrent weight W_{rec} to the power of t and the initial hidden state: $h^t = W_{rec}^t h_0$. Hence, since the weight is a matrix in this case, the gradient explodes if the absolute values of the matrix eigenvalues are larger than 1 and the gradient diminishes if the absolute values of the matrix eigenvalues are lower than 1.

To solve the vanishing gradient problem Hochreiter and Schmidhuber [92] proposed long short term memory (LSTM) networks which are composed of a self-memory cell, an input node, an input gate and an output gate. Later, Gers et al. [62] added the forget gate which enables to remove parts of the self-memory cell. Gers and Schmidhuber [61] added peephole connections in which the self-memory cell is directly incorporated in the calculation of input and output gates. Hence, information can also flow from self-memory cell by not only depending on the information

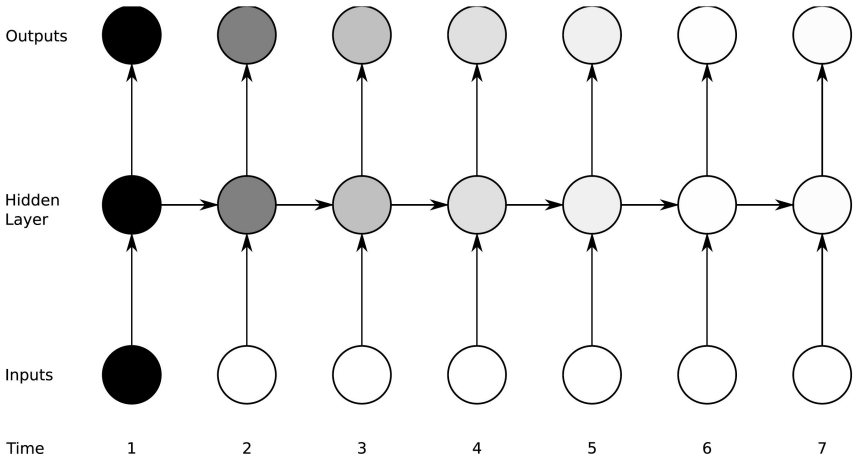


Figure 2.1: The vanishing gradient problem. From darker shade to lighter the sensivity decreases exponentially over time (taken from [74]).

flow from the previous time step output gate. More recently the gated recurrent unit (GRU) was proposed by [36, 40] which is composed of two gating mechanisms: update and reset gates. The update gate can pass on the previous state or replace it by the candidate recurrent state (activation), and the reset gate controls which parts of the previous state are used to compute the target state. There are multiple versions of GRUs and LSTMs [223, 115, 148]; although some of them gain higher performance on some tasks, they do not yield clear improvement [79, 104].

Schuster and Paliwal [168] proposed a recurrent network architecture in which recurrent input representation contains the information from both past and future. This architecture is not applicable for online processing where the future values are unknown. Later, Graves and Schmidhuber used bidirectional LSTMs for phoneme classification [73] and handwriting recognition [72].

Recurrent neural networks can be used to model input and/or output sequences of same or different lengths [108]: one-to-many, many-to-one, many-to-many. One-to-many – in which a fixed length input vector is mapped to an output distribution over sequences – can be applied in the image captioning task where a fixed length input image mapped by an RNN to an output distribution over words constituting the corresponding caption [47, 109]. Many-to-one – in which an input sequence is mapped to an output distribution over a single variable – can be a sentiment analysis

or time series classification where the input is a sequence of words or time series and the output is a categorical label, e.g., positive, negative or class 1, 2, etc. [152]. Many-to-many RNNs map an input sequence to an output distribution over sequences of the same length or variable length. Named entity recognition or video frame classification are examples of the many-to-many case with the same length of input and output sequences in which each word/frame in a sentence/video have a categorical label [224]. Machine translation, time series forecasting are examples of many-to-many with variable-length input and output sequences where the input is a sequence of words in the source language and the output is the distribution of words in the target language [183, 136]. Many-to-many variable-length input/output sequences are typically addressed by using encoder-decoder networks, which are also called sequence-to-sequence architectures [183, 36].

2.4.3 Sequence-to-sequence and attention models

The sequence-to-sequence (encoder-decoder) architecture for sequence prediction was proposed independently by multiple researchers [54, 183, 36]. In this architecture, the network is composed of a reader/encoder which reads the input and produces a context/embedding of the input and a decoder which decodes this representation to an output distribution. While [183, 36] uses RNNs for encoding and decoding and the last recurrent hidden unit constitutes the context – summary of input sequence – provided to the decoder, [54] uses FNN as encoding/decoding units. This context is used by the decoder to generate an output distribution over a sequence which is not necessarily the same length as that of the input sequence. The main drawback of this approach is that the whole input sequence is expected to be represented with a fixed-size vector, which lacks interpretability. Bahdanau et al [8] proposed a soft search over the set of input encoding and aligns the important parts of them to the output at each step of decoding. This soft search over input is also called *attention mechanism*. Attention is calculated using a FNN which takes the recurrent hidden state of the encoder at time j – which is the representation of the j^{th} element in the input sequence – and the previous recurrent hidden state of the decoder at time

$(i-1)$ – which is the representation of the $(i-1)^{\text{th}}$, *i.e*, the previous, element in the output sequence. This feedforward network calculates the alignment of input at j to next target output i given previous recurrent hidden state of the decoder $(i-1)$ which represent previous output prediction and possibly the summary of the decoded sequence till $(i-1)$.

Attention mechanism is initially proposed for the machine translation task [8] and inspired many researchers in various fields of applications and many variants of attention were proposed [136, 215, 24, 195]. [218] studied two different types of attention models for the image captioning task: the first one is stochastic hard attention that is non-differentiable and can be trained by variance reduction or the REINFORCE technique [212] and the second one is deterministic soft attention which is differentiable and can be trained by standard back-propagation. Here, the soft attention mechanism is similar to the work of Bahdanau et al [8]. While soft attention results in a wider focus area, hard attention results in a much narrow focus.

Later, [136] studied local and global attention mechanisms which are similar to hard and soft attention, respectively, according to their attention coverage. Here, the focus area of local attention is fixed by a predefined fixed window-size and the center of the window is determined by a FNN with a sigmoid activation function which is multiplied with the input length. Attention weights are smoothed with a Gaussian function centered around the center of window. They explore different alignment scores, namely: dot, general and concat. Chorowski et al [39] propose smoothing and sharpening for the normalization of attention weights to overcome the noise coming from irrelevant feature vectors (recurrent hidden states). They obtained a sharpening effect using an inverse temperature $\beta > 1$ as a multiplicand to the attention weights which can be seen as an alignment score. Here, they obtained a smoothing effect by replacing the unbounded exponential function of the softmax function with a bounded logistic sigmoid function. Their sharpening mechanism leads to a sparse attention weights similar to the hard attention mechanism studied in [218].

Up to this point, we talked about content-based attention mechanisms which focus on the region of input according to the spatial, temporal or spatio-temporal relevance.

In comparison to content-based attention which returns a context as a weighted summary of input encoding, location-based attention returns an index, address to focus on. Location-based attention mechanism is also used in object recognition [4, 143], classification [124] and combinatorial optimization [199]. Larochelle and Hinton [124] studies visual glimpses to mimic human sequences of fixations to recognize an object. They propose a controller network which determines where to look next. Zheng et al [231] studied an alternative auto-encoder network again with a controller network which determines where to look. [4, 143] also use glimpses for object detection but instead they use an LSTM to process the glimpses and here, location attention is calculated using a feed-forward neural network which predicts the next image patch to check based on lossy summary of image patches generated by an LSTM. Graves [75] uses location-based attention in a synthesis network to generate handwriting by a window layer.

Some studies add location information in the content-based attention calculation [39, 229]. These approaches differ from location-based attention mechanisms since they return the weights to calculate a context vector instead of returning the location prediction. Chorowski et al. [39] add a position vector which is the filtered previous attention weights (filtering is done by a convolution with a matrix that is learned with other parameters). Zhang et al. [229] uses relative object and subject positions with recurrent hidden states and last hidden state to compute a content-based soft attention for relation extraction task. Recent work on neural machine translation uses feedforward networks and attention mechanism. This way, the computational complexity is reduced but the information about the order of the sequence is lost. Hence authors proposed to enrich the input representation by adding sinusoidal positional encoding [194]. [59] adds absolute position of input elements to input representations for a convolutional encoder-decoder neural machine translation architecture.

For tasks requiring long dependency, the internal memory of RNNs (and LSTMs) are not sufficient, many researchers thus proposed to use an external memory and have read and write operations on the external memory and they use attention mechanism operations on memory [76, 206]. Neural Turing machines [76] use both content-based and location-based attention for read and write operations on the external memory.

Content-based attention is regularized by a coefficient β in the softmax normalization, which resembles the sharpening mechanism used in [39] except here β is not restricted to be larger than 1. They use a gating mechanism to determine the type of attention to be used among content-based or location-based attention. Graves et al. [77] extended neural Turing machines by using again content-based attention and dynamic memory allocation instead of location-based addressing. [206] uses an attention mechanism in the output of the end-to-end memory network for a given query. Here, attention weights are considered as probabilities of output vectors for this given query. By using the attention mechanism they are able to train their network with back-propagation, which facilitates the training compared to their earlier approach of memory networks [207].

Other uses of attention appears in Pointer Networks [199], the set encoder-decoder [197] and one-shot learning [198]. [199] use attention weights obtained over input encodings as probabilities of inputs. This allows the network to solve combinatorial optimization problems where the output length depends on the input sequence length. [197] proposes to use attention over a set of inputs, without accounting for the order in the input set. To do so, they define an RNN whose recurrent state is enriched by a reading vector at each time step. This is done by concatenating reading vector to the recurrent state following each iteration of LSTM. The reading vector can be a soft content over the set of input values. In one-shot learning [198], attention is used to measure alignment between training instances and corresponding labels to describe the output of a new class. Furthermore, [222] extends the encoder input representation in the calculation of attention with a recurrent state of previous attention weights.

Early attention mechanisms are used to align two different sources, e.g., in neural translation, alignment is done between two different languages [8, 24, 215] or in image captioning alignment is done between part of an image and a textual word [109, 200, 47, 4]. Later, researchers proposed to use an attention mechanism on the same source, e.g., between the words in a single sentence [131, 128]. This type of attention calculation on the same source is called self-attention or inner-attention.

[131] uses an inner-attention mechanism to obtain a sentence context only using input sentence words for language inference. They calculate attention weights between the bi-directional LSTM recurrent states of an input sentence and their mean-pooling using addition. [128] also suggest to use attention on sentence words to obtain a sentence representation in the context of author profiling, sentiment classification and textual entailment. Their self-attention mechanism calculation is very similar to [8]. They calculate self-attention over the bi-directional LSTM representations of words with multiple hops to extract different parts of the sentence in which resulting attention weights is a matrix (number of hops $\times$ sequence length) instead of a vector ($1 \times$ sequence length).

Vaswani et al. [194] uses self-attention and standard attention in a neural machine translation network called transformer. Instead of one global attention on input space, they propose to use multi-head attention which is a combination of several single attention units each jointly attending to different positions. They use self-attention both in the encoder and decoder sides using solely the input representation or the previous layer states. Self-attention on the decoder side includes a mask which enables to enforce the auto-regressive property (dependency in sequential order) of the output sequence. Hence, the conditional probability of a word to be generated depends only on the previous words. Decoder additionally contains a standard attention which aligns encoder and decoder representations. As opposed to earlier work which adopted RNNs or convolutional neural networks (CNNs) to capture sequential dependency, [194] uses attention mechanism in fully connected feedforward networks and they augment the input representation with sinusoidal positional encoding.

2.5 Time Series Forecasting

Time series forecasting is a challenging task that is extensively studied in the literature [70]. Initially, researchers analyzed time series by decomposing to four main components: trend, seasonality, cycles, and irregularities. Trend corresponds to a systematic evolution which might itself remain constant or change. Trends can be incorporated in the model (implicitly or explicitly) or can be chosen to be removed

as pre-processing step [33]. Seasonality is repeating patterns within a certain period, *e.g.*, temperature in certain season of a year – this is also called periodicity. For the sake of generality, we will use the term *period* here. For instance, restaurants might illustrate a weekly pattern: *e.g.*, a restaurant located in an industrial environment would have peaks during lunches on week days, and a restaurant located in a residential area would likely have peaks in the weekends and evenings. We can also consider an example of yearly period if the restaurants are located in a touristic neighbourhood, depending on the tourism season it would have higher peaks during winters or summers. In many cases, seasonal adjustments are applied by removing the source of variation [81, 43], to increase interpretability by simplifying the data [12]. However, this causes some information loss. Cycles are any non-periodic recognizable patterns, they do not have a fixed time span (duration of occurrence). Any other variations without a particular pattern, mostly as a result of noise, constitute irregularities.

Many researchers studied various ways of decomposing a time series into its components by using an additive or multiplicative decomposition [41, 119, 188, 33]. Additive decomposition is well-suited for a time series with constant trend and seasonal components and multiplicative decomposition is well-suited for a time series with components varying in magnitude proportionally over time.

Earlier stochastic time series forecasting approaches were built on the stationarity assumption of time series. A time series is strongly stationary when its distribution is invariant to time shifts and it is weakly stationary when its first and second moments are invariant to time shifts. Stationarity in time series simplifies the forecasting task by assuming these properties of time series remain constant over time. The autoregressive (AR) analysis of time series formulates forecasting of future values of a time series as a linear combinations of previous p values and a noise term of current prediction – an uncorrelated zero-mean random variable which models the uncertainty – and the moving average (MA) analysis formulates forecasting of future values of a time series as a linear combination of previous q white noise terms [202, 177, 33]. The autoregressive moving average analysis formulates forecasting of future values as a linear combination of both autoregressive and moving average terms and it is generalized to non-stationary time series by differencing approaches by Box et al

[211, 22]. The autoregressive integrated moving average (ARIMA) model initially transforms the time series to be stationary by differencing approaches. Then, the stationarized time series is modelled with autoregressive and moving average components. This way, ARIMA provides a solution for non-stationary time series. Many variants of ARMA and ARIMA models were studied in the literature extensively [51, 18, 26, 21, 71, 196]. Generalized autoregressive conditional heteroskedasticity (GARCH) models non-stationary variance as a stochastic process [51] which models the non-stationarity in the second moment. For multivariate time series forecasting, generalized vector ARMA (VARIMA) models are studied [189]. State space models are another approach of time series forecasting [33] which is a continuous version of HMMs.

Apart from the stochastic generative approaches, many machine learning models are utilized for time series forecasting: feed-forward neural networks [96, 228], random forests [23, 107], support vector machines [147, 64, 166], hidden Markov models [87, 35], partially observed Markov decision processes [7], vanilla RNNs [205, 42, 67], LSTMs [60].

The periods or seasonality in time series [81, 175, 56, 55, 196] and missing values in time series [167, 129, 34, 37, 110] were extensively studied to improve time series forecasting task. Some approaches to deal with missing data infer missing values by smoothing or interpolation [137], imputation [210], spectral analysis [145], kernel methods [157].

Predicting time series with missing values was also addressed using RNNs [129, 37, 110, 57, 34]. [129, 37, 110] replaced missing values with zero, the mean or the latest value and addressed the informative missingness with an input composed of concatenation of measurements, sampling information and time intervals. [57] used a Gaussian process as an imputing step before an RNN. [34] addresses informative missingness by integrating masking and time interval in a GRU architecture.

2.5.1 Time Series Forecasting Evaluation

Here we present two standard time series forecasting evaluation metrics, namely mean square error (MSE) and symmetric mean absolute percentage error (SMAPE). The proposed time series forecasting methods (described in Section 3.3) are evaluated by using the MSE and SMAPE scores.

The MSE and SMAPE of a time series forecasting of a horizon T' for a test set of N number of instances are given in Equation 2.1 and Equation 2.2, respectively.

$$\text{MSE} = \frac{1}{NT'} \sum_{n=1}^{N} \sum_{t'=1}^{T'} (y_n^{t'} - \hat{y}_n^{t'})^2 \tag{2.1}$$

$$\text{SMAPE} = \frac{1}{NT'} \sum_{n=1}^{N} \sum_{t'=1}^{T'} \frac{|\hat{y}_n^{t'} - y_n^{t'}|}{(|y_n^{t'}| + |\hat{y}_n^{t'}|)/2} \tag{2.2}$$

In Equation 2.1 and Equation 2.2, $y_n^{t'}$ is the ground-truth target value at horizon t' of test instance n and $\hat{y}_n^{t'}$ is the corresponding predicted value.

2.6 Query Prediction

Next query prediction is predicting the next query of the user, which is also called query suggestion or recommendation. A common approach to query suggestion is to leverage *wisdom of crowds* by analyzing IR search logs [6, 16]. Various sequential approaches are used to model a search session in a structured way: Markov decision processes [83], partially observed Markov decision processes[135, 134], a graph structure [5, 16], adversarial learning [203], recurrent neural networks [178, 46]. Reformulation patterns [83, 135, 134, 176] and the probability of a query to follow the previous query(ies) or the previous interaction(s) [5, 16, 178, 46] are the two main perspectives adapted to analyze/formulate an IR search session.

Guan et al. [83] defined a Markov decision process to model an IR search session with two cooperative agents: user agent and search engine agent. The queries are defined as MDP states, user actions are reformulations of query terms, e.g., removing

or adding query terms, and the search engine actions are adjustments of the term weights, e.g., increasing, decreasing and maintaining the term weight. In this work, they do not consider the clicks of a user as user actions. Luo et al. [135] proposes a partially observed Markov decision process which encompasses the decision states, query changes, clicks and rewards as a cooperative game between the user and search engine agents. Both [83] and [135] use a restricted number of states and actions to reduce model complexity and to improve efficiency. To improve the complexity-efficiency trade off, [134] used direct policy learning with gradient descent which enables to directly map observations to actions.

Search queries can be also organized in a graph-like structure [5]. Boldi et al. [16] studied a query-flow graph in which the nodes are the queries appearing in search-logs and the directed edges between the two query nodes are the indicators of the same *search mission*. Two different weighting schemes are associated with the directed edge from query i to query j: probability of the two queries being in the same search mission and the probability of the query j following query i. For the query recommendation task they defined the weights based on the frequency of q_j following q_i. Similarly, Mei et al [140] also used random-walk on a bipartite click graph in which the weights are computed based on the time taken to visit the suggested query node for a random walk starting from the first query. Later, Boldi et al. [17] proposed a different weighting by using random walks on different slices of the original graph – those determined by classification of the edges based on the transition type from q_i to q_j: generalization, specialization, error correction and parallel move. [20] extended query-flow graph by adding entity nodes in the graph. [187] leveraged WordNet [142] and adapted a query template approach by replacing query entities with their type to overcome the issue of data sparsity using random walks on query graphs.

Many researchers addressed the query prediction task as a ranking problem [6, 150, 165]. [6] clustered semantically similar queries by also utilizing clicked documents. The queries are recommended for a given input query by ranking similar queries, clustered together with input query, according to their relevance. Ozertem et al [150] used learning to rank for query prediction rankings. They defined lexical and result set features (the set of retrieved documents) for co-occurring sequences

to generate candidate sequences. They also determined a utility score of reformulations based on the common URLs between candidate and original queries. Another learning to rank approach for ranking candidate queries are studied by Santos et al. [165]. They enriched the feature space by leveraging textual similarity of queries and co-occurrence in common sessions or common clicked URLs. Their comparison of the usage of different sources of information suggests using the session and click information improve query prediction [165].

Neural network approaches were extensively studied for various IR tasks in the recent years [149]. As discussed in Section 2.4.3, the sequence-to-sequence, encoder-decoder, architecture is extensively used in various domains, e.g., computer vision, NLP. Sordoni et al. [178] studied an extension of the sequence-to-sequence approach ([183, 36]) – an hierarchical recurrent encoder-decoder. To best of our knowledge, it was the first usage of a recurrent network architecture for the session-based query prediction. They proposed an hierarchical encoder which captures the granularity of the query and session levels in two hierarchical levels: a query-level encoder and a session-level encoder. The query-level encoder takes query word embeddings as input and the output of this encoder is fed to the session-level encoder. A decoder is used to infer the probability of the next word given a sequence of words. Here, a decoder takes as input the session-level encoder state and a *go* symbol (which is a token defined in the vocabulary, a signal to recurrent network to produce prediction). The candidate queries are predicted using the hierarchical encoder-decoder network using beam-search algorithm. They ranked the candidate queries by using a list-wise learning to rank method, LambdaMART [27] .

Another session-based query prediction approach using RNNs adapted the query reformulation approach by using attention, copy and generation mechanisms [46]. Their model contains a switch mechanism that allows the model to choose to copy a word from the input query or generate a new query word. They used attention on the encoder part of the two-level hierarchical encoder-decoder network, extending [178] by adding attention, copy and switch mechanisms. To generate a new query word they followed the approach of [178]. The attention is calculated on the first-level encoder, between the first-level encoder recurrent states and the decoder's previous

step hidden state. It is also calculated on the second-level encoder, between the second-level encoder recurrent states, the decoder's previous step hidden state and the word predicted (output of the decoder) in the previous step. The resulting attention combines the two attentions by multiplying the two weights and normalizing. The decoder of the network is composed of a copier block, a generator block and a switch gate. The generator estimates the conditional joint probability of words given session information (as in the earlier work [178]). The copier determines the words to be copied among the query words by using a pointer network [199] (more details can be find in Section 2.4.3). The switch mechanism chooses which block to be used: copier or generator by levering some predefined rules on the outputs of copier and generator. The network is trained by using a multi-objective function [46].

[214] used a feedback memory network in which the click through information of the earlier interactions is explicitly modelled. They adopted the hierarchical architecture which was proposed in [178] and enriched the input of the session encoder with positive and negative feedback memories along with the query-level encoder output. Here, the clicked documents are considered as positive feedback and the non-clicked documents ranked higher than the clicked documents considered as negative feedback documents. Feedback memories are formed using document embedding, positional embedding of respective document and attention weights. Here, they calculated a multiplicative attention between query embedding and document embeddings for both positive and negative feedback. Each type of documents have a separate softmax over attention (similarity/alignment) scores which results in attention weights summing to 1 for the positive documents and attention weights summing to 1 for the negative. [2] utilizes multi-task learning for retrieval and query suggestion tasks by using an end-to-end neural network architecture. They extend the architecture from Sordoni et al. [178] by adding a document encoder and document ranker which are trained on user query logs and simulated user behavior on the query logs. Both their document and query encoders are bi-directional LSTMs with a max pooling layer where the document encoder has a higher number of units than the query encoder.

2.6.1 Next Query Prediction Evaluation

The evaluation of a next query prediction system is performed by assessing the quality of the next query proposals. For that purpose, one can use measures based on n-grams, on edit distance between the predicted next query and the target (original) next query. Furthermore, the retrieval performance of the predicted query can indicate the quality of the proposed query in the context of the topic of the search session. This can be done by assessing the ranking performance of the predicted query based on the topic of the IR search session.

Lexical Evaluation

Here we present the average Jaccard similarity, the sentence-level bilingual evaluation understudy (BLEU) score and the Levenshtein distance between the predicted and original queries, which can be adopted to assess next query proposals and to form features. The average Jaccard similarity aim to evaluate the lexical similarity between the predicted and original queries.

- The Jaccard similarity is defined as the ratio between the intersection and the union of the predicted and original queries. In other words, it computes the proportion of the number of intersecting words divided by the number of all words in the set of original and predicted query words:

$$\text{Jac}(\hat{Q}, Q) = \frac{|\hat{Q} \cap Q|}{|\hat{Q} \cup Q|} \tag{2.3}$$

  where $\hat{Q}$ and Q are the set of predicted and original query words.

- The BLEU score is used to evaluate the aptness of a candidate text compared to the reference text. It is initially proposed for machine translation to evaluate the candidate translations [151]. It is calculated based on the number of position-independent n-gram matches in predicted and reference text (queries in our case).

$$\text{BLEU}(\hat{Q}, Q) = \text{BP}(\hat{Q}, Q) \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n(\hat{Q}, Q)\right) \tag{2.4}$$

where p_n is the modified n-gram precision (Equation 2.5), w_n is a positive weight summing to 1, and BP is the brevity penalty (Equation 2.6). The modified precision uses a clipped count of word occurrences in the candidate query. This determines the number of time it should appear at most in the candidate and avoid biased calculation of precision by limiting the count of a repeating word. Here, the clip threshold is the maximum number of a word appearing in the reference text (the clip threshold is only based on 1 text = 1 query). Hence, a specific word appearing in candidate text more frequently than any reference text is penalized.

$$p_n(\hat{Q}, Q) = \frac{\sum_{n\text{-gram}\in\hat{Q}} \max\Big(|\{n\text{-gram} \in \hat{Q}\}|, |\{n\text{-gram} \in Q\}|\Big)}{\sum_{n\text{-gram}\in\hat{Q}} |\{n\text{-gram} \in \hat{Q}\}|} \tag{2.5}$$

The precision is inversely proportional to the total number of n-grams in the candidate text. Brevity penalty enables to adapt the length of the candidate text to the length of the reference text (there is a 1-to-1 correspondence in your case).

$$\mathrm{BP}(\hat{Q}, Q) = \begin{cases} 1 & \text{if } |\hat{Q}| > |Q| \\ \mathrm{e}^{(1-|Q|/|\hat{Q}|)} & \text{if } |\hat{Q}| \leq |Q| \end{cases} \tag{2.6}$$

where $|\hat{Q}|$ and $|Q|$ are the candidate and effective reference query lengths, respectively.

- The Levenshtein distance (*Lev*) is also called word error rate. It can be summarized as the number of editing steps to transform a candidate text into the reference text. Editing steps are substitutions (replace a word with another), insertions (add a word) or deletions (remove a word). *Lev* is calculated as the total number of editing steps divided by the reference length:

$$Lev = \frac{\#\text{ substitutions} + \#\text{ insertions} + \#\text{ deletions}}{|Q|} \tag{2.7}$$

Rank-based Evaluation

The rank-based measure evaluates how well the ranking performance of a predicted query is compared to the original query. Here, we present the two most common ranking scores: mean Average Precision (MAP) error and normalized Discounted Cumulative Gain (nDCG).

Precision at K (P@K) is the proportion of relevant documents among the top K ranked documents.

$$\texttt{P@K} = \sum_{i=1}^{K} \frac{rel(D_i)}{K} \tag{2.8}$$

Average precision (AP) is the average of the precision at different cutoff positions in which the relevant documents ranked. Because of these cutoff points, AP takes into account both recall and precision.

$$\texttt{AP} = \sum_{i} \frac{P@i \ rel(D_i)}{|rel|} \tag{2.9}$$

where $|rel|$ is the number of relevant documents for the query on which the evaluation is conducted.

MAP is the mean of AP over the set of N evaluations.

$$\texttt{MAP} = \sum_{n=1}^{N} \frac{\texttt{AP}_n}{N} \tag{2.10}$$

- nDCG@K uses graded relevance judgment [101] by discounting the relevance of the document by the logarithm of its rank. nDCG@K is normalized by dividing DCG@K of the ranked list by DCG@K of the ideal ranked list (iDCG@K). iDCG@K corresponds to DCG@K of the best rank list possible for a given query.

$$\texttt{nDCG@K} = \frac{\texttt{DCG@K}}{\texttt{iDCG@K}} \tag{2.11}$$

$$\texttt{DCG@K} = \sum_{i=1}^{K} \frac{2^{rel(D_i)-1}}{\log_2(i+1)} \tag{2.12}$$

2.7 Conclusion

In this chapter, we presented the background on sequential learning approaches, namely hidden Markov models, Markov decision processes, recurrent neural networks, which is necessary for the following chapters. Furthermore, we detailed the background of sequence prediction in the context of time series forecasting and next query prediction. Considering the sequence learning capabilities of recurrent neural networks, we further explore recurrent neural network models on these two sequence prediction scenarios in Chapter 3 and Chapter 4.

Chapter 3

Period-aware Content Attention RNNs for Time Series Forecasting with Missing Values

Recurrent neural networks (RNNs) recently received considerable attention for sequence modeling and time series analysis. Many time series contain periods, e.g. seasonal changes in weather time series or electricity usage at day and night time. Here, we first analyze the behavior of RNNs with an attention mechanism with respect to periods in time series and illustrate that they fail to model periods. Then, we propose an extended attention model for sequence-to-sequence RNNs designed to capture periods in time series with or without missing values. This extended attention model can be deployed on top of any RNN, and is shown to yield state-of-the-art performance for time series forecasting on several univariate and multivariate time series.

3.1 Introduction

Forecasting future values of temporal variables is termed as *time series forecasting* and has applications in a variety of fields, as finance, economics, meteorology, or customer support center operations. A considerable number of stochastic [70] and machine learning based [19] approaches have been proposed for this problem. A

particular class of approaches that has recently received much attention for modeling sequences is based on sequence-to-sequence Recurrent Neural Networks (RNNs) [75]. Sequence-to-sequence RNNs constitute a flexible class of methods particularly well adapted to time series forecasting when one aims at predicting a sequence of future values on the basis of a sequence of past values. They have furthermore led to state-of-the-art results in different applications (as machine translation or image captioning). We explore here how to adapt them to general time series.

Time series often display *pseudo-periods*, *i.e.* time intervals at which there is a strong correlation, *positive or negative*, between the values of the time series. Pseudo-periods can be due for example to environmental factors as seasonality or to the patterns underlying the activities measured (the workload on professional email servers for example has both weekly and daily periods). In a forecasting scenario, pseudo-periods correspond to the difference between the positions of the output being predicted and specific inputs. In the remainder of the thesis, we will use the term *period* to refer to either a true period or a pseudo-period.

A first question one can ask is thus: *Can sequence-to-sequence RNNs model periods in time series?* In order to capture periods in RNNs, one needs a *memory* of the input sequence, *i.e.* a mechanism to reuse specific (representations of) input values to predict output values. As the input sequence is usually longer than the periods underlying the time series, longer-term memories that store information pertaining to past input sequences (as described in *e.g.* [207, 206, 77]) are not required. A particular model of interest here is the content attention model proposed in [8] and described in Section 3.3. This model allows one to reuse the content of the input sequence to predict the output values. However, this model was designed for text translation and one can wonder whether it can capture position-based periods in time series.

Furthermore, missing observations and gaps in the data are also common, due to *e.g.* lost records and/or mistakes during data entry or faulty sensors. In addition, for multivariate time series, underlying variables can have mixed sampling rates, *i.e.* different variables may have different sampling frequencies, resulting again in values missing at certain times when comparing the different variables. The standard

strategy to tackle missing values is to apply numerical or deterministic approaches, such as interpolation and data imputation methods, to explicitly estimate or infer the missing values, and then apply classical methods for forecasting. In the context of RNNs, a standard technique, called padding, consists in repeating the hidden state of the last observed value over the missing values. All these methods however make strong assumptions about the functional form of the data, especially for long gaps, and can introduce sources of errors and biases. The second question we address here is thus: *Do sequence-to-sequence RNNs handle well missing values?*

The main contributions of this chapter address the above two questions and are based on:

- A complete and general framework for time series forecasting using a period-aware content attention mechanism;
- An experimental study showing the incapability of RNNs in detecting (pseudo-)periods in time series;
- Several new models and architectures to efficiently handle missing values and gaps in time series. These models are combined with the period-aware content attention mechanism mentioned above and yield state-of-the-art performances on several benchmark data sets.

The remainder of the chapter is organized as follows. Related studies are discussed in Section 3.2. We then conduct, in Section 3.3, simple experiments to illustrate how state-of-the-art RNNs behave on time series. Next, in the same section, different extensions of the attention mechanism are provided in order to handle periods and missing values for both univariate and multivariate time series. To evaluate the proposed approach, an extensive set of experiments has been conducted. This set of experiments is presented and discussed in Section 3.5. Finally, Section 3.6 concludes this chapter.

3.2 Related Work

The notion of stochasticity of time series modeling and prediction was introduced long back [226]. Since then, various stochastic models have been developed, notable among these are autoregressive (AR) [202] and moving averages (MA) [177] methods. These two models were combined in a more general and more effective framework, known as autoregressive moving average (ARMA), or autoregressive integrated moving average (ARIMA) when the differencing is included in the modelling [22]. Vector ARIMA or VARIMA [189] is the multivariate extension of the univariate ARIMA models where each time series instance is represented using a vector.

Neural networks are regarded as a promising tool for time series prediction [228, 45] due to their data-driven and self-adaptive nature, their ability to approximate any continuous function and their inherent non-linearity. The idea of using neural networks for prediction dates back to 1964 where an adaptive linear network was used for weather forecasting [96]. But the research was quite limited due to the lack of a training algorithm for general multilayer networks at the time. Since the introduction of backpropagation algorithm [164], there had been much development in the use of neural networks for forecasting. It was shown in a simulated study that neural networks can be used for modeling and forecasting nonlinear time series [120]. Several studies [205, 170] have found that neural networks are able to outperform the traditional stochastic models such as ARIMA or Box-Jenkins approaches [22].

With the advent of SVM, it has been used to formulate time series prediction as a regression estimation using Vapnik's insensitive loss function and Huber's loss function [147]. The authors showed that SVM based prediction outperforms traditional neural network based methods. Since then different versions of SVMs are applied for time series prediction and many different SVM forecasting algorithms have been derived [155, 53]. More recently, random forests [23] are used for time series predictions due to their performance accuracy and ease of execution. Random forest regression is used for prediction in the field of finance [44] and bioinformatics [118], and are shown to outperform ARIMA [107].

Traditional neural networks allow only feedforward connections among the neurons of a layer and the neurons in the following layer. In contrast, *recurrent neural networks* (RNN) [102, 50] allow both forward and feedback or recurrent connections between the neurons of different layers. Hence, RNNs are able to incorporate contextual information from past inputs which makes them an attractive choice for predicting general sequence-to-sequence data, including time series [75]. In this chapter we use RNNs for modeling time series. Early work [42] has shown that RNNs (a) are a type of nonlinear autoregressive moving average (NARMA) model and (b) outperform feedforward networks and various types of linear statistical models on time series. Subsequently, various RNN-based models were developed for different time series, as noisy foreign exchange rate prediction [67], chaotic time series prediction in communication engineering [99] or stock price prediction [95]. A detailed review can be found in [123] on the applications of RNNs along with other deep learning based approaches for different time series prediction tasks.

RNNs based on LSTMs [92], which we consider in our study, alleviate the *vanishing gradient* problem of the traditional RNNs proposed. They have furthermore been shown to outperform traditional RNNs on various temporal tasks [60, 63]. More recently, they have been used for predicting the next frame in a video and for interpolating intermediate frames [156], for forecasting the future rainfall intensity in a region [217], or for modeling clinical data consisting of multivariate time series of observations [129].

Adding attention mechanism on the decoder side of an encoder-decoder RNN framework enabled the network to focus on the interesting parts of the encoded sequence [8]. Attention mechanism is used in many applications such as image description [218], image generation [80], phoneme recognition [39], heart failure prediction [38], as well as time series prediction [159] and classification [38]. Many studies also apply attention mechanism on external memory [76, 78].

The previous work [159] uses attention mechanism to determine the importance of a factor among other factors that affect time series. However, we use extended attention mechanism to model periods and emphasize the interesting parts of input

sequence with missing values. Our approach is applicable for both univariate and multivariate time series prediction.

The attention mechanism has been specialized in [199], under the name pointer network, so as to select the best input to be reused as the output. This model would be perfect for noise-free, truly periodic times series. In practice, however, times series are noisy and if the output is highly correlated to the input corresponding to the period, it is not an exact copy of it.

A recent study [229] uses relative position information respect to object and subject for relation extraction task, more specifically slot filling. They add position information besides the original sequence encoding, whereas we propose a model adjust the sequence encoding according to period and missing values.

However, to the best of our knowledge, no work has focused on analyzing the adequacy of RNNs (based or not on LSTMs) for time series, in particular w.r.t. their ability to model periods and handle missing values (for this latter case, some methods based on *e.g.* Gaussian processes have been proposed to handle irregularly sampled data and missing values [66], but none related to RNNs). To our knowledge, this study is the first one to address this problem.

3.3 Handling periodicity and missing values in RNNs

We first focus on univariate time series. As mentioned before, time series forecasting consists in predicting future values from past, observed values. The time span of the past values, denoted by T, is termed as *history*, whereas the time span of the future values to be predicted, denoted by T', is termed as *forecast horizon* (in multi-step ahead prediction, which we consider here, $T' > 1$). The prediction problem can be formulated as a regression-like problem where the goal is to learn the relation $\mathbf{y} = r(\mathbf{x})$ where $\mathbf{y} = (y_{T+1}, \ldots, y_{T+i}, \ldots, y_{T+T'})$ is the output sequence and $\mathbf{x} = (x_1, \ldots, x_j, \ldots, x_T)$ is the input sequence. Both input and output sequences are ordered and indexed by time instants. Here, we use bold characters for vectors and matrices.

3.3.1 Background

Sequence-to-sequence RNNs rely on three parts, one dedicated to encoding the input, and referred to as *encoder*, one dedicated to constructing a summary of the encoding, which we will refer to as *summarizer*, and one dedicated to generating the output, and referred to as *decoder*. The encoder represents each input x_j, $1 \leq j \leq T$ as a *hidden state* $\overrightarrow{\mathbf{h}}_j = f(x_j, \overrightarrow{\mathbf{h}}_{j-1})$, $\overrightarrow{\mathbf{h}}_j \in \mathbb{R}^n$, where the function f corresponds here to the non-linear transformation implemented in LSTM with peephole connections [63]. The equations of LSTM with peephole connections are given in Eq. (3.1). For bidirectional RNNs [169], the input is read both forward and backward, leading to two vectors $\overrightarrow{\mathbf{h}}_j = f(x_j, \overrightarrow{\mathbf{h}}_{j-1})$ and $\overleftarrow{\mathbf{h}}_j = f(x_j, \overleftarrow{\mathbf{h}}_{j+1})$. The final hidden state for any input x_j is constructed simply by concatenating the corresponding forward and backward hidden states, *i.e.* $\mathbf{h}_j = [\overrightarrow{\mathbf{h}}_j; \overleftarrow{\mathbf{h}}_j]^\intercal$, where now $\mathbf{h}_j \in \mathbb{R}^{2n}$:

$$
\begin{aligned}
\mathbf{i}_j &= \sigma(\mathbf{W}_{\mathrm{i}} x_j + \mathbf{U}_{\mathrm{i}} \mathbf{h}_{j-1} + \mathbf{C}'_{\mathrm{i}} \mathbf{c}'_{j-1}) \\
\mathbf{f}_j &= \sigma(\mathbf{W}_f x_j + \mathbf{U}_f \mathbf{h}_{j-1} + \mathbf{C}'_f \mathbf{c}'_{j-1}) \\
\mathbf{c}'_j &= \mathbf{f}_j \mathbf{c}'_{j-1} + \mathbf{i}_j \tanh(\mathbf{W}_{c'} x_j + \mathbf{U}_{c'} \mathbf{h}_{j-1}) \\
\mathbf{o}_j &= \sigma(\mathbf{W}_o x_j + \mathbf{U}_o \mathbf{h}_{j-1} + \mathbf{C}'_o \mathbf{c}'_{j-1}) \\
\mathbf{h}_j &= \mathbf{o}_j \tanh(\mathbf{c}'_j)
\end{aligned}
\tag{3.1}
$$

The summarizer builds, from the sequence of input hidden states $\mathbf{h}_j, 1 \leq j \leq T$, a context vector $\mathbf{c} = q(\{\mathbf{h}_1, \ldots, \mathbf{h}_j, \ldots, \mathbf{h}_T\})$. In its simplest form, the function q just selects the last hidden state [75]: $q(\{\mathbf{h}_1, \ldots, \mathbf{h}_j, \ldots, \mathbf{h}_T\}) = \mathbf{h}_T$. More recently, in [8], an attention mechanism is used to construct different context vectors $\mathbf{c}_i$ for different outputs y_i ($T+1 \leq i \leq T+T'$) as a weighted sum of the hidden states of the encoder representing the input history:

$$
\mathbf{c}_i = \sum_{j=1}^{T} \alpha_{ij} \mathbf{h}_j \tag{3.2}
$$

where α_{ij} are the *attention weights*. They are computed as:

$$
\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'=1}^{T} \exp(e_{ij'})} \tag{3.3}
$$

where

$$e_{ij} = a(\mathbf{s}_{i-1}, \mathbf{h}_j) = \mathbf{v}_a^{\mathsf{T}} \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a \mathbf{h}_j) \tag{3.4}$$

with a being a feedforward neural network with weights $\mathbf{W}_a$, $\mathbf{U}_a$ and $\mathbf{v}_a$ trained in conjunction with the entire encoder-decoder framework. $\mathbf{s}_{i-1}$ is the hidden state obtained by the decoder (see below) at time $i-1$. One can note that a scores the importance of the input at time j (specifically its representation $\mathbf{h}_j$ by the encoder) for the output at time i, given the previous hidden state $\mathbf{s}_{i-1}$ of the decoder. This allows the model to concentrate, or *put attention*, on certain parts of the input history to predict each output.

The decoder parallels the encoder by associating each output y_i, $T+1 \leq i \leq T+T'$, to a hidden state vector $\mathbf{s}_i = g(y_{i-1}, \mathbf{s}_{i-1}, \mathbf{c}_i)$, where $\mathbf{s}_i \in \mathbb{R}^n$ and y_{i-1} denotes the output at time $i-1$. The function g corresponds again to an LSTM with peephole connections, with an explicit context added [75]. The equations of LSTM with peephole connections and the context are given in Eq. (3.5). Each output is then predicted in sequence through:

$$y_i = \mathbf{W}_{\text{out}} \mathbf{s}_i + b_{\text{out}},$$

where b_{out} is a scalar and $\mathbf{W}_{\text{out}} \in \mathbb{R}^n$:

$$\begin{aligned}
\mathbf{i}_i &= \sigma(\mathbf{W}_{\mathrm{i}} y_{i-1} + \mathbf{U}_{\mathrm{i}} \mathbf{s}_{i-1} + \mathbf{C}'_{\mathrm{i}} \mathbf{c}'_{i-1} + \mathbf{C}_{\mathrm{i}} \mathbf{c}_i) \\
\mathbf{f}_i &= \sigma(\mathbf{W}_f y_{i-1} + \mathbf{U}_f \mathbf{s}_{i-1} + \mathbf{C}'_f \mathbf{c}'_{i-1} + \mathbf{C}_f \mathbf{c}_i) \\
\mathbf{c}'_i &= \mathbf{f}_i \mathbf{c}'_{i-1} + \mathbf{i}_i \tanh(\mathbf{W}_{c'} y_{i-1} + \mathbf{U}_{c'} \mathbf{s}_{i-1} + \mathbf{C}_{c'} \mathbf{c}_i) \\
\mathbf{o}_i &= \sigma(\mathbf{W}_o y_{i-1} + \mathbf{U}_o \mathbf{s}_{i-1} + \mathbf{C}'_o \mathbf{c}'_{i-1} + \mathbf{C}_o \mathbf{c}_i) \\
\mathbf{s}_i &= \mathbf{o}_i \tanh(\mathbf{c}'_i)
\end{aligned} \tag{3.5}$$

Below, we first illustrate that traditional RNNs with the standard attention mechanism fail to capture the periods and, as a result, cannot make use of this point in order to improve the prediction accuracy. We then detail several extensions of the attention mechanism that efficiently integrate periods and handle missing values.

3.3.2 Standard RNNs and periods

To illustrate how RNNs behave w.r.t. periods in time series, we retained two periodic time series, fully described in Section 3.5 and representing respectively (a) the electrical consumption load in Poland over the period of 10 years (1/1/2002-7/31/2012), and (b) the maximum daily temperature, again in Poland, over the period of 12 years (1/1/2002-7/31/2014). The first time series, referred to as `PSE`, has two main periods, a daily one and a weekly one, whereas the second time series, referred to as `PW`, has a yearly period. Figure 3.1 (top) displays the autocorrelation [33] for each time series and shows these different periods.

To verify if RNNs with standard attention mechanism, denoted as **RNN-A** hereafter, are able to capture the periods, we plot, in Figure 3.1 (bottom), the attention weights obtained with RNN-A. In addition, we conducted another experiment where we added the time stamp (or position) information j to the input to help the RNNs capture potential periods. When the positions are added, each input at time j contains two values: x_j and j, $1 \leq j \leq T$. The position is of course not predicted in the output. The results we obtained, evaluated in terms of mean squared error (MSE) and displayed in Figure 3.1 (bottom), show that adding the time stamps in the input does not improve the RNNs. Were the original RNN-A able to capture periods, we should see that the corresponding weights in the attention mechanism are higher. However, as one can note, this is not the case: for RNN-A with time stamp information, a higher weight is put on the first instance, which does not correspond to any of the actual periods of the times series. With standard RNN-A, higher weights are put on the more recent history on `PSE`. This seems more reasonable, and leads to better results, even though the period at one day is missed. On `PW`, the weights with standard RNN-A are uniformly distributed on the different time stamps. This shows that the standard attention mechanism does not always detect (and make us of) the underlying periods of the data.

To further illustrate this point, we show, in Table 3.1, the MSE values for both `PSE` and `PW` datasets with and without position information. Although adding position can improve the results on `PW`, it fails to perform better than RNN-A on `PSE`. This

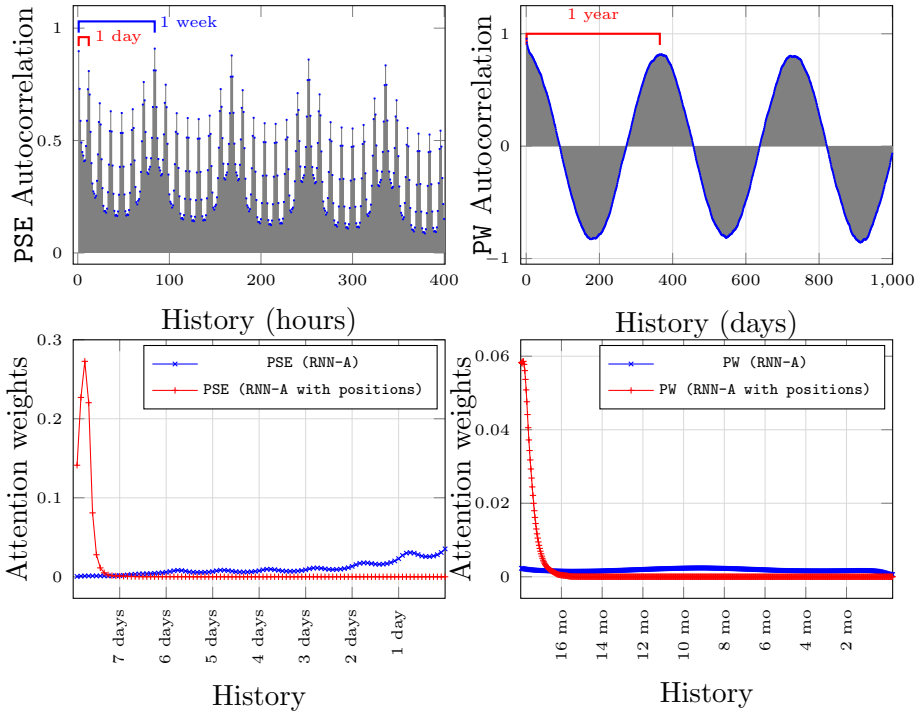


Figure 3.1: Top: autocorrelation function illustrating the periods on Polish electricity (PSE), and Polish weather (PW) time series datasets (details of the datasets are given in Table 3.2). Bottom: weights of the attention mechanism for the RNNs without and with time stamp (or position) information on both PSE and PW (the weights are averaged over all test examples, see Section 3.5).

suggests that the attention mechanism needs to be adapted to a different type of information in order to capture periods. We propose such extensions below.

3.3.3 Period-aware content attention mechanism

We assume here that the periods of a time series lie in the set $\{1, ..., T\}$ where T is the history size of the time series[1]. One can then explicitly model all possible periods as a real vector, which we will refer to as $\boldsymbol{\pi}$, of dimension T ($\in \mathbb{R}^T$), the coordinates of which encode the importance of the relative position between the output and the input. From this, one can modify the weight of the original attention mechanism

[1]This assumption is easy to satisfy by increasing the size of the history if the periods are known or by resorting to a validation set to tune T.

Table 3.1: MSE values for RNN-A with and without position information on PSE and PW.

Dataset	RNN-A	RNN-A with position	RNN-A with sinusoidal position
PSE	0.034	0.048	0.048
PW	0.166	0.154	0.155

relating input j to output i as follows:

$$e_{ij} = \begin{cases} \mathbf{v}_a^\mathsf{T} \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a(\boldsymbol{\pi}_{i-j}\, \mathbf{h}_j)) & \text{if } (i-j) \leq T \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

$\boldsymbol{\pi}_{i-j}$, i.e. the $(i-j)^{\text{th}}$ coordinate of π, is used to increase or decrease e_{ij} and thus the weight α_{ij} relating input j to output i. Note that, as the history is limited to T, there is no need to consider dependencies between an input j and an output i that are beyond T time steps (hence the test: $i - j \leq T$).

The vector $\mathbf{e}_i$ can then be normalized using the softmax operator again, and a context can be built by taking the expectation of the hidden states over the normalized weights. For practical purposes, however, one can simplify the above formulation by extending the vectors with T' dimensions that are set to 0 by considering a scalar $\mathbb{1}_{(i-j\leq T)}$ that returns 1 on the first T coordinates and 0 on the last T' ones. The resulting position-based attention mechanism then amounts to:

$$e_{ij} = \mathbf{v}_a^\mathsf{T} \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a(\boldsymbol{\pi}_{i-j}\, \mathbf{h}_j))\, \mathbb{1}_{(i-j\leq T)} \tag{3.7}$$

As one can note, $\boldsymbol{\pi}_{i-j}$ will either decrease or increase the hidden state vector $\mathbf{h}_j$ for output i. Since $\boldsymbol{\pi}$ is learned along with the other parameters of the RNNs, we expect that $\boldsymbol{\pi}_{i-j}$ will be high for those positions i and j that correspond to periods of the time series. Lastly, note that the original attention mechanism can be obtained by setting $\boldsymbol{\pi}$ to $\mathbf{1}$ (a vector consisting of 1 on each coordinate). We will refer to this model as RNN-π. Figure 3.2 illustrates RNN-π architecture.

In the above formulation, the position information is used to modify the importance of each hidden state in the input side. It may be, however, that some elements

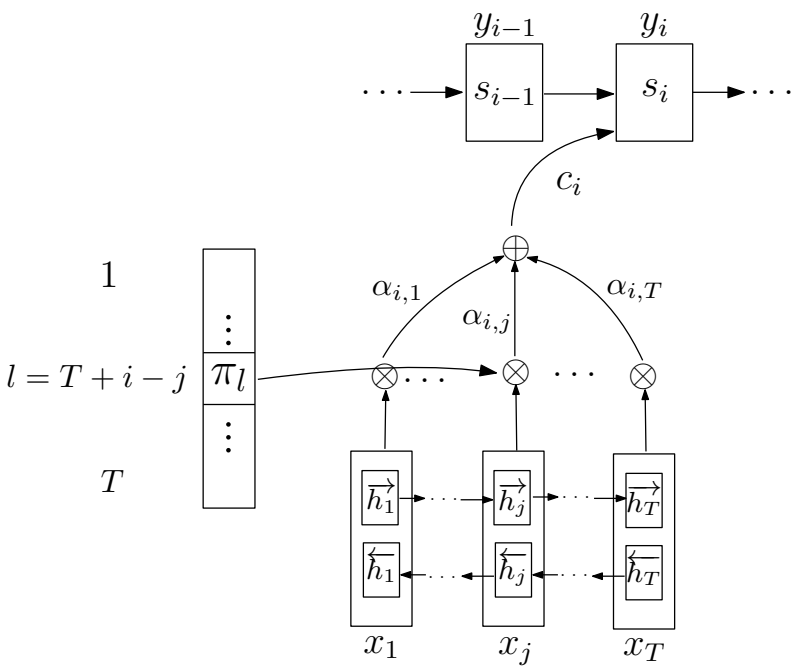


Figure 3.2: Illustration of RNN-π architecture.

in $\mathbf{h}_j$ are less important than others to predict output i. It is possible to capture this by considering a vector in $\mathbb{R}^{2n}$ that can now reweigh each coordinate of $\mathbf{h}_j$ independently. Here, using a matrix, $\mathbf{\Pi}$, one can have a finer control over the encoder hidden states. This leads to:

$$e_{ij} = \mathbf{v}_a^\mathsf{T} \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a(\mathbf{\Pi}_{.(i-j)} \odot \mathbf{h}_j)) \, \mathbb{1}_{(i-j \leq T)} \tag{3.8}$$

where $\odot$ denotes the Hadamard product (element wise multiplication) and $\mathbf{\Pi}$ is a matrix in $\mathbb{R}^{2n \times T}$; $\mathbf{\Pi}_{.(i-j)}$ represents the $(i-j)^{th}$ column of this matrix. We will refer to this model as RNN-Π. To calculate the attention weights α_i and context $\mathbf{c}_i$, we rely on Eqs. (3.2) and (3.3). Figure 3.3 illustrates RNN-Π architecture.

3.3.4 Handling missing values

Provided that their proportion is not too important, otherwise the forecasting task is less relevant, missing values in time series can be easily identified by considering that the most common interval between consecutive points corresponds to the sampling rate of the time series. Points not present at the expected intervals are then considered

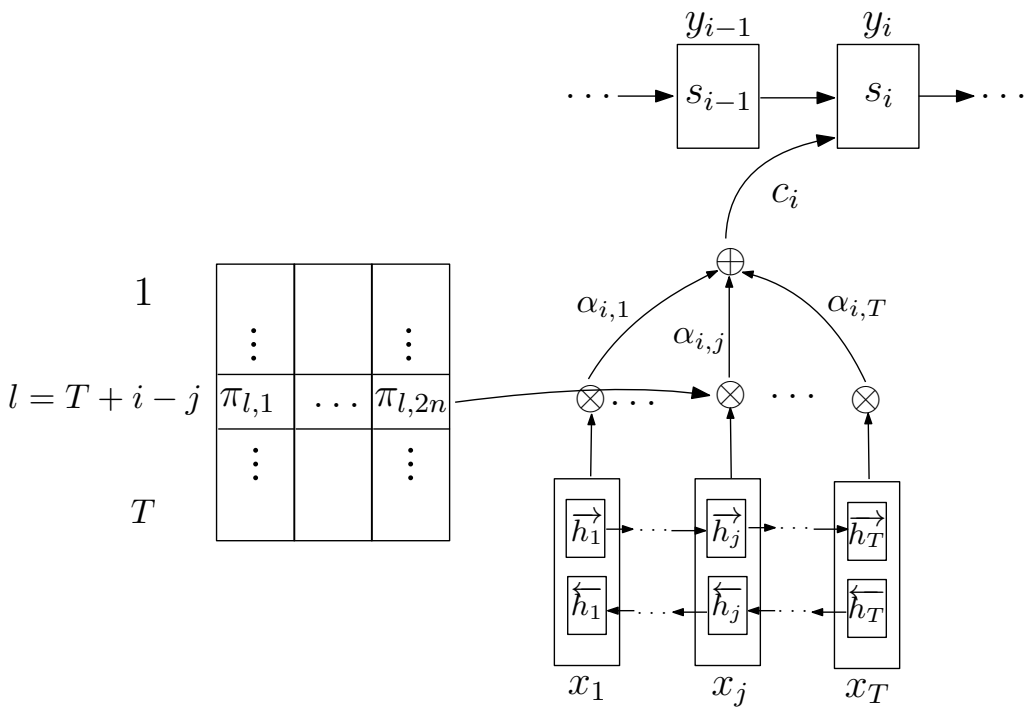


Figure 3.3: Illustration of RNN-Π architecture.

as missing. It is thus possible to identify missing values in time series, and then use padding or interpolation techniques to represent them. Padded or interpolated inputs should however not be treated as standard inputs as they are less reliable than other inputs. Furthermore, be it with padding or interpolation techniques, when the size of a gap, *i.e.* consecutive missing values, is important, the further away the missing value is from the observed values (the last one in case of padding, the last and next ones in case of interpolation), the less confident one is in the padded hidden states or interpolated values. It is thus desirable to decrease the importance of missing values according to their position in gaps.

We propose to do so by another modification of the attention mechanism that again reweighs the attention weights. We consider two reweighing schemes: a first one that penalizes missing values further away from the last observed value through a decaying function, *a priori* adapted to padding, and a second one that penalizes missing values depending on whether they are at the beginning, middle or end of a gap, *a priori* adapted to interpolation methods which rely on values before and after the gap.

The reweighing schemes corresponds to the following equations:

$$\omega(j) = \begin{cases} \exp(-\mu_j(j - j_{\text{last}})) \\ 1 + \boldsymbol{M}_{.j}^{\mathsf{T}} \mathbf{Pos}(j; \theta_1^g, \theta_2^g) \end{cases} \tag{3.9}$$

In the first scheme, j_{last} denotes the last observed value before j and $\boldsymbol{\mu} \in \mathbb{R}^T$ is an additional vector of parameters that is learned with other parameters of the RNN. The exponential reweighing scheme is referred to as RNN-μ. $\omega(j)$ equals to 1 when the value at position j is available. When there is a missing value at position j, down-weighting is applied via $j - j_{last}$: the further in the gap it is located, the more it is down-weighted. $\boldsymbol{\mu}$ enables to differentiate the amount of down-weight according to where j lies in the history $[1, \ldots, T]$.

In the second scheme, RNN-M, each gap is divided into three regions of equal length, corresponding to the beginning, middle, and end, and defined through two hyper-parameters θ_1^g and θ_2^g. $\mathbf{Pos}(j; \theta_1^g, \theta_2^g)$ is a three-dimensional vector that is null if j is not missing; otherwise, denoting the length of the gap by $|g|$, if $\frac{j - j_{\text{last}}}{|g|} < \theta_1^g$, then the first coordinate of $\mathbf{Pos}(j; \theta_1^g, \theta_2^g)$ is set to 1 and the others to 0, if $\theta_1^g < \frac{j - j_{\text{last}}}{|g|} < \theta_2^g$, then the second coordinate is set to 1 and the others to 0 and if $\frac{j - j_{\text{last}}}{|g|} > \theta_2^g$, then the third coordinate is set to 1 and the others to 0.

$\boldsymbol{M} \in \mathbb{R}^{3 \times T}$ is a matrix which is learned with other parameters of the RNN, where each of the coordinates of a column aims at reweighing the impact of missing values on the prediction task according to their position in gaps. As mentioned before, θ_1^g and θ_2^g are hyper-parameters that are set such that the gap g is divided into three equal parts (corresponding to the beginning, middle and end of the gap). Here, the down-weighting corresponding to the missing input at position j is a vector of dimension 3 that can learn different coefficients for the beginning, middle and end of the gap and is specific to position j.

In addition, the aforementioned position-aware down-weighting procedures handle issues with non-uniformly distributed missing values by initializing $\boldsymbol{\mu}$ and $\boldsymbol{M}$ to 0 (*i.e.*, all values are set to 0). Doing so, a missing value at a given position that has never been observed in the training set will be multiplied by 0 as it should not affect the weights.

One can extend the models proposed for handling periods, given in Section 3.3.3, with the proposed missing reweighing schemes. For this, $\mathbf{h}_j$ $\big($Eq. (3.4)$\big)$ is replaced by $(\omega(j)(\boldsymbol{\pi}_{i-j}\,\mathbf{h}_j))$ or $(\omega(j)\,(\boldsymbol{\Pi}_{\cdot(i-j)}\,\odot\,\mathbf{h}_j))$. Eqs. (3.10) summarize the equations of e_{ij} for all models (the names of the models are given at the end of each equation).

$$e_{ij} = \begin{cases} \mathbf{v}_a^\mathsf{T} \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a\, \pi_{i-j}\,(\omega(j)\mathbf{h}_j))\, \mathbb{1}_{(i-j\leq T)} & \\ \qquad\qquad \text{with } \omega(j) = \exp(-\mu_j(j - j_{\text{last}})) & (\boldsymbol{\pi}\text{-}\boldsymbol{\mu}) \\ \mathbf{v}_a^\mathsf{T} \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a\, \pi_{i-j}\,(\omega(j)\mathbf{h}_j))\, \mathbb{1}_{(i-j\leq T)} & \\ \qquad\qquad \text{with } \omega(j) = 1 + \boldsymbol{M}_{\cdot j}^\mathsf{T}\mathbf{Pos}(j;\theta_1^g,\theta_2^g) & (\boldsymbol{\pi}\text{-}\boldsymbol{M}) \\ \mathbf{v}_a^\mathsf{T} \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a\, (\boldsymbol{\Pi}_{\cdot(i-j)}\,\odot\,(\omega(j)\mathbf{h}_j)))\, \mathbb{1}_{(i-j\leq T)} & \\ \qquad\qquad \text{with } \omega(j) = \exp(-\mu_j(j - j_{\text{last}})) & (\boldsymbol{\Pi}\text{-}\boldsymbol{\mu}) \\ \mathbf{v}_a^\mathsf{T} \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a\, (\boldsymbol{\Pi}_{\cdot(i-j)}\,\odot\,(\omega(j)\mathbf{h}_j)))\, \mathbb{1}_{(i-j\leq T)} & \\ \qquad\qquad \text{with } \omega(j) = 1 + \boldsymbol{M}_{\cdot j}^\mathsf{T}\mathbf{Pos}(j;\theta_1^g,\theta_2^g) & (\boldsymbol{\Pi}\text{-}\boldsymbol{M}) \end{cases} \tag{3.10}$$

Note that the calculations of α_{ij} $\big($Eq. (3.3)$\big)$ and $\mathbf{c}_i$ $\big($Eq. (3.2)$\big)$ remain the same.

3.4 Multivariate Extensions

As each variable in a K multivariate time series can have its own periods, a direct extension of the above approaches to multivariate time series is to consider that each variable k, $1 \leq k \leq K$, has its own encoder and attention mechanism. We thus construct, for each variable k, context vectors on the basis of the following equation:

$$\alpha_{ij}^{(k)} = \frac{\exp\left(e_{ij}^{(k)}\right)}{\sum_{j'=1}^{T} \exp\left(e_{ij'}^{(k)}\right)} \tag{3.11}$$

where $e_{ij}^{(k)}$ is given by Eqs. (3.7), (3.8) and (3.10).

The context vector for the i^{th} output of the k^{th} variable is then defined by $\mathbf{c}_i^{(k)} = \sum_{j=1}^{T} \alpha_{ij}^{(k)} \mathbf{h}_j^{(k)}$, where $\mathbf{h}_j^{(k)}$ is the input hidden state at time stamp j for the k^{th} variable and $\alpha_{ij}^{(k)}$ is the weights given by the attention mechanism of the k^{th} variable.

Lastly, to predict the output while taking into account potential dependencies between different variables, one can simply concatenate the context vectors from the

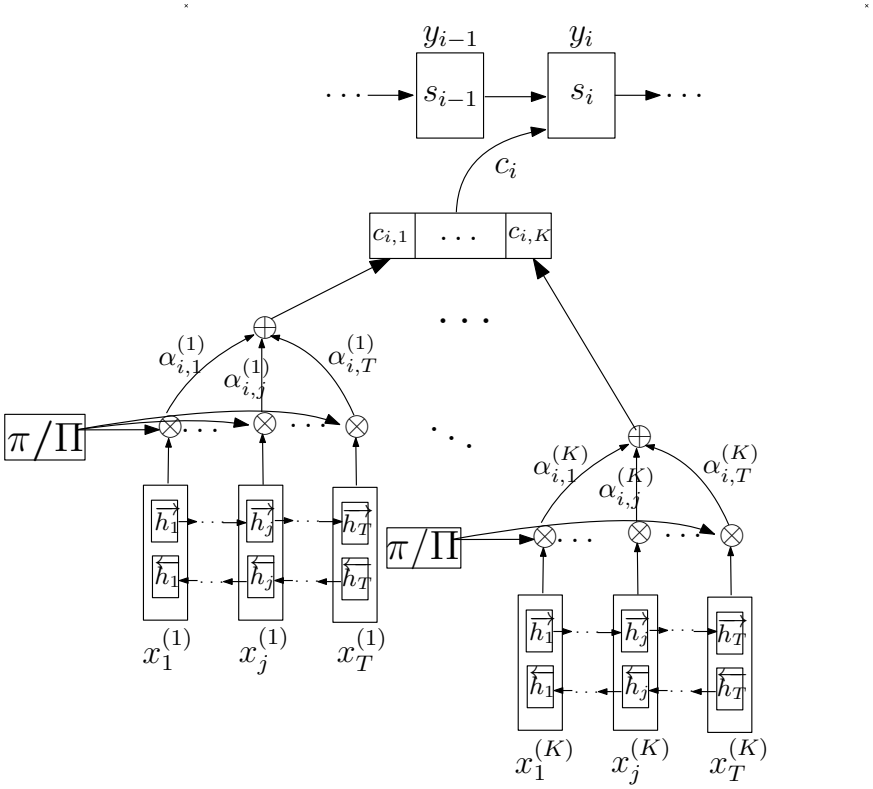


Figure 3.4: Illustration of RNN-Π^m architecture (separate encoder and attention mechanism for each temporal variable k).

different variables into a single context vector $\mathbf{c}_i$ that is used as input of the decoder, the rest of the decoder architecture being unchanged:

$$\mathbf{c}_i = [\mathbf{c}_i^{(1)^\mathsf{T}} \cdots \mathbf{c}_i^{(K)^\mathsf{T}}]^\mathsf{T}$$

As each $\mathbf{c}_i^{(k)}$ is of dimension $2n$ (that is the dimension of the input hidden states), $\mathbf{c}_i$ is of dimension $2Kn$. This strategy can readily be applied to the original attention mechanism as well as the ones based on $\boldsymbol{\pi}$ and $\boldsymbol{\Pi}$ with and without handling missing values.

It is nevertheless possible to rely on a single attention model for all variables while having separate representations for them in order to select, for each output, specific hidden states from different variables. To do so, one can simply concatenate the hidden states of each variable into a single hidden state ($\mathbf{h}_j = [\mathbf{h}_j^{(1)^\mathsf{T}} \cdots \mathbf{h}_j^{(K)^\mathsf{T}}]^\mathsf{T}$) and deploy the previous attention model on top of them. This leads to the multivariate model which we refer to as RNN-Π^m and that is based on the same ingredients and

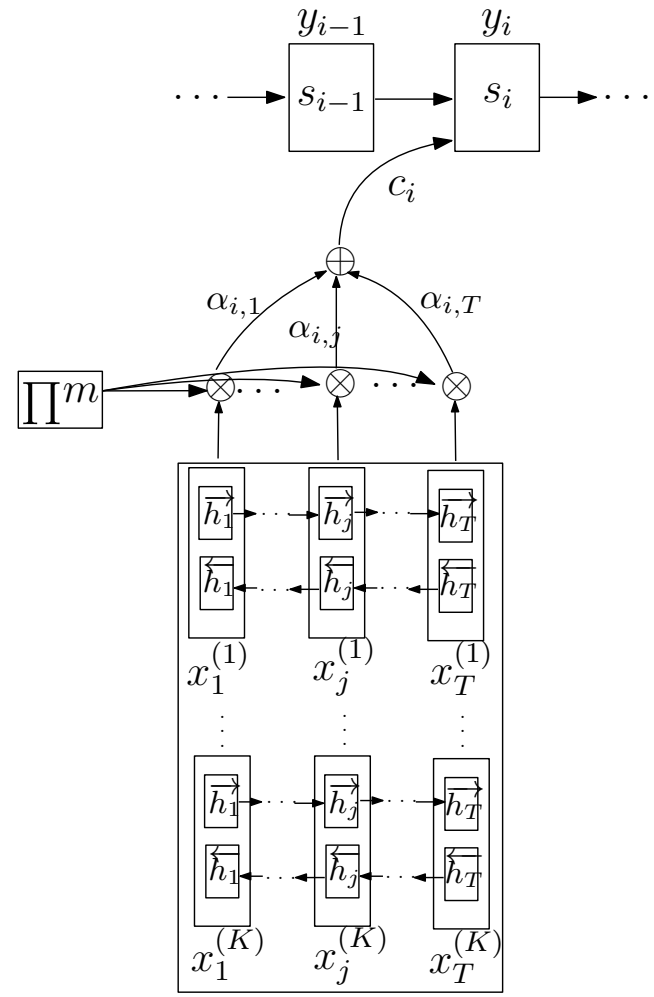


Figure 3.5: Illustration of RNN-Π^m architecture (single attention for temporal variables $1, \ldots, K$).

equations as RNN-Π, the only difference being that one uses now a matrix, $\mathbf{\Pi^m}$, in $\mathbb{R}^{2Kn \times T}$:

$$e_{ij} = \mathbf{v}_a^\top \tanh(\mathbf{W}_a \mathbf{s}_{i-1} + \mathbf{U}_a(\mathbf{\Pi^m}_{\cdot(i-j)} \odot \mathbf{h}_j)) \, \mathbb{1}_{(i-j \leq T)} \tag{3.12}$$

Figure 3.5 illustrates the single attention model for all variables.

Handling missing values can directly be done by replacing $\mathbf{h}_j$ with $(\omega(j)\mathbf{h}_j)$, with $\omega(j)$ given by Eq. (3.9).

In the following section, we illustrate the behaviour of our proposals on several univariate and multivariate datasets.

3.5 Experiments

3.5.1 Datasets and experimental setup

To assess the models proposed, we retained six widely used and publicly available datasets:

1. `PSE`: Polish Electricity (`http://www.pse.pl`).
2. `PW`: Polish Weather (`https://globalweather.tamu.edu`).
3. `NAB`: Numenta Anomaly Benchmark (`https://numenta.com`).
4. `AQ`: Air Quality (`https://archive.ics.uci.edu/ml`).
5. `AEP`: Appliances Energy Prediction (`https://archive.ics.uci.edu/ml`).
6. `OLD`: Ozone Level Detection (`https://archive.ics.uci.edu/ml`).

that are described in Table 3.2. The values for the history size were set so as they encompass the known periods of the datasets. They can also be tuned by cross-validation if one does not want to identify the potential periods by checking the autocorrelation curves. In general, the forecast horizon should reflect the nature of the data and the application one has in mind, with of course a trade off between long forecast horizon and prediction quality. For this purpose, the forecast horizons of these sets along the sampling rates are chosen as illustrated in Table 3.2. All datasets were split by retaining the first 75% of each dataset for training-validation and the last 25% for testing. For RNN-based methods, the training-validation sets were further divided by retaining the first 75% for training (56.25% of the data) and the last 25% for validation (18.75% of the data). For the baseline methods, we used 5-fold cross-validation on the training-validation sets to tune the hyperparameters. Lastly, linear interpolation was used whenever there are missing values in the time series[2].

[2]We compared several methods for missing values, namely linear, non-linear spline and kernel based Fourier transform interpolation as well as padding for the RNN-based models. The best reconstruction was obtained with linear interpolation, hence its choice here.

Table 3.2: Datasets.

Name	Usage	#Instances	History size	Forecast horizon	Sampling rate
PSE	Univariate	46379	96	4	2 hours
PW	Univariate	4595	548	7	1 days
NAB	Univariate	18050	72	6	5 minutes
AQ	Univ./Multiv.	9471	192	6	1 hour
AEP	Univ./Multiv.	19735	216	6	10 minutes
OLD	Univ./Multiv.	2536	548	7	1 day

We compared the methods introduced before, namely RNN-(π/Π/Π^m), with the original attention model (RNN-A) and several baseline methods, namely ARIMA, an ensemble learning method, Random Forests (RF), and support vector regression. Among these baselines, we retained ARIMA and RF as these were the two best performing methods in our datasets. These methods, discussed in Section 3.2, have also been shown to provide state-of-the-art results on various forecasting problems (*e.g.* [107]). Note that for ARIMA, we relied on the seasonal variant [98]. To implement the RNN models, we used theano[3] and Lasagne [4] on a Linux system with 256GB of memory and 32-core Intel Xeon @2.60GHz. All parameters are regularized and learned through stochastic backpropagation (the mini-batch size was set to 64) with an adaptive learning rate for each parameter [112], the objective function being the Mean Square Error (MSE) on the output. For tuning the hyperparameters, we used a grid search over the learning rate, the regularization type and its coefficient, and the number of units in the LSTM and attention models. The values finally obtained are 10^{-3} for the initial learning rate and 10^{-4} for the coefficient of the regularization, the type of regularization selected being L_2. The number of units vary among the set {128, 256} for LSTMs and {256, 512} for the attention models respectively. We report hereafter the results with the minimum MSE on the test set. For evaluation, we use MSE and the symmetric mean absolute percentage error (SMAPE). MSE corresponds to the objective function used to learn the model. SMAPE presents the

[3] http://deeplearning.net/software/theano/

[4] https://lasagne.readthedocs.io

advantage of being bounded and represents a scaled L_1 error.

3.5.2 Handling periods

We first study the behaviour of our proposed methods on univariate time series, prior to consider multivariate time series.

3.5.2.1 Univariate time series

For univariate experiments using multivariate time series, we chose the following variables from the datasets: for `PW`, we selected the *max temperature* series from the Warsaw metropolitan area that covers only one weather recording station; for `AQ` we selected *C6H6(GT)*; for `AEP` we selected the *outside humidity (RH6)*; for `NAB` we selected the *Amazon Web Services CPU usage* and for `OLD` we selected *T3*. Table 3.3 displays the results obtained with the MSE (left value) and the SMAPE (right value) as evaluation measures. Once again, one should note that MSE was the metric being optimized. For each time series, the best performance among all methods is shown in bold and other methods are marked with an asterisk if they are significantly worse than the best method according to a paired t-test with 5% significance level. Lastly, the last column of the tables corresponding to the results, *Selected model*, indicates which method, among RNN-π and RNN-Π, was selected as the best method on the validation set using MSE.

As one can note, except for `AEP` where the baselines are better than RNN-based methods, for all other datasets, the best results are obtained with RNN-π and RNN-Π, these results being furthermore significantly better than the ones obtained with RNN-A and baseline methods, for both MSE and SMAPE. The MSE improvement varies from one dataset to another: between 8% (`PW`) and 26% (`NAB`) w.r.t RNN-A. Compared to ARIMA, one can achieve an improvement ranging from 18% (15%), in `OLD`, to 94% (75%), in `PSE`, w.r.t MSE (SMAPE). In addition, the selected RNN-(π/Π) method (column *Selected model*) is the best performing method on three out of six datasets (`AQ`, `PW`, `PSE`) and the best performing RNN-π method on `AEP`. It is furthermore equivalent to the best performing method on `OLD`, the only dataset on

Table 3.3: Overall results for univariate case with MSE (left value) and SMAPE (right value).

Dataset	RNN-A	RNN-π	RNN-Π	ARIMA	RF	*Selected model*
AQ	0.282*(0.694)	0.257(0.661)	**0.25**(0.669)	0.546*(0.962)	0.299*(0.762)	Π
OLD	0.319*(0.595)	**0.271**(0.523)	0.275(0.586)	0.331*(0.619)	0.305*(0.606)	Π
AEP	0.025*(0.085)	0.029*(0.101)	0.027*(0.095)	**0.021**(0.066)	0.021(0.085)	Π
NAB	0.642*(0.442)	**0.475**(0.323)	0.54*(0.369)	1.677*(1.31)	0.779*(0.608)	Π
PW	0.166*(0.558)	**0.152**(0.547)	0.162*(0.565)	0.213*(0.61)	0.156(0.544)	π
PSE	0.034*(0.282)	**0.032**(0.264)	0.033*(0.256)	0.623*(1.006)	0.053*(0.318)	π

which the selection fails being NAB (a failure means here that the selection does not select the best RNN-π method). However, on this dataset, the selected method is still better than the original attention model and the baselines. Overall, these results show that RNN-π and RNN-Π significantly improve forecasting in the univariate time series we considered, and that one can automatically select the best RNN-π method.

Lastly, to illustrate the ability of RNN-π methods to capture periods, we display in Figure 3.6 the average attention weights for PW (right) and PSE (left) obtained with RNN-A and RNN-Π (averaged over all test examples and forecast horizon points). As one can see from the autocorrelation plot, displayed in Figure 3.1, PSE has two main weekly and daily periods. This two periods are clearly visible in the attention weights of RNN-Π that gives higher weights to the four points located at positions *minus 7 days* and *minus 1 day* (these four points correspond to the four points of the forecast horizon). In contrast, the attention weights of the original attention model (RNN-A) follow a general increasing behaviour with more weights on the more recent time stamps. This model thus misses the periods. One should note that although we display attention weights of RNN-Π in this figure, those of RNN-π are similar

3.5.2.2 Multivariate time series

As mentioned in Table 3.2, we furthermore conducted multivariate experiments on AQ, AEP and OLD using the multivariate extensions described in Section 3.3. For AQ, we selected the four variables associated to real sensors, namely C6H6(GT), NO2(GT), CO(GT) and NOx(GT) and predicted the same one as the univariate

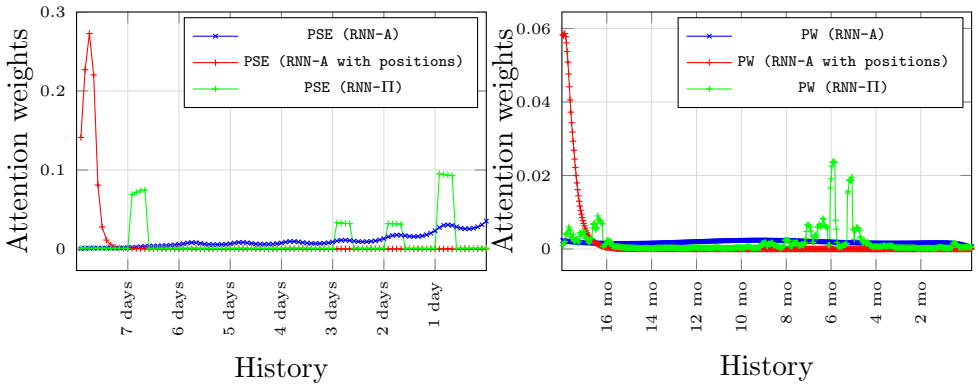


Figure 3.6: Attention weights for PW (right) and PSE (left) for RNN-A, RNN-A with position and RNN-π. See Figure 3.1 for the autocorrelation plots of these datasets.

case (C6H6(GT)). For AEP, we selected two temperature time series, namely T1 and T6, and two humidity time series, RH6 and RH8, and we predict RH6 as in the univariate case. For OLD, we trained the model using T0 to T3 and predicted T3, as we did on the univariate case. As RF outperformed ARIMA on five out of six univariate datasets and was equivalent on the sixth one, we retained only RF and RNN-A for comparison with RNN-π, RNN-Π, and RNN-Π^m.

Table 3.4 shows the results of our experiments on multivariate sets with MSE (SMAPE, not displayed here for readability reasons, has a similar behaviour). As before, for each time series, the best result is in bold and an asterisk indicates that the method is significantly worse than the best method (again according to a paired t-test with 5% significance level). Similarly to the univariate case, the best results, that are always significantly better than the other results, are obtained with the RNN-π methods: for AQ and OLD datasets, RNN-π can respectively bring 24% and 18% of significant improvement over RNN-A. Similarly, for AEP, the improvement is significant over RNN-A (17% with RNN-π). Compared to RF, one can obtain between 11% (AEP) and 40% (AQ) of improvement. As one can note, the selected method is always RNN-Π^m. The selection is this time not as good as for the univariate case as the best method (sometimes significantly better than the one selected) is missed. That said, RNN-Π^m remains better than the state-of-the-art baselines retained, RNN-A and RF.

Table 3.4: Overall results for multivariate case with MSE.

Dataset	RNN-A	RNN-π	RNN-Π	RNN-Π^m	RF	*Selected model*
AQ	0.352*	0.276*	**0.268**	0.3*	0.45*	Π^m
OLD	0.336*	0.328*	0.327*	**0.274**	0.315*	Π^m
AEP	0.029*	**0.024**	0.036*	0.026*	0.027*	Π^m

3.5.3 Handling missing values

In order to assess whether the proposed methods are robust to missing values, we introduced different levels of missing values in the datasets with the following strategy: for each dataset, we added 5, 10, 15, 20, 30 and 40 percent of missing values. As the datasets should contain both random missing values and gaps, we introduced half gaps and half random missing values. For instance, to add 10% of missing values, we first introduce 5% of gaps and then 5% of random missing values that do not have any interference with the previously introduced gaps.

Furthermore, as the gaps could be of different sizes, we let the length of gaps vary from 5 to 100, picked at random. Note that for AQ, which already contains 18% of missing values, we first introduced new missing values until we reach half of the desired percentage, then we introduce the gaps. In our experiments, the percentage of introduced missing values are shown as a number along with the dataset name; for the original dataset we just mention the dataset name.

As the RNN-based models outperformed the others in our previous experiments, we focus on them here. Furthermore, in order to assess the impact of handling missing values, we report the results obtained with the standard RNN-A method, with the period based models, RNN-π/Π, and with the complete models integrating both periods and weighting schemes (RNN-π/Π-μ/M). The results obtained are displayed in Table 3.5. The column *Selected-*π/Π indicates which model for handling periods performs best on the validation set, the column *Selected-*π/Π-μ/M which model, handling both periods and missing values, performs best on the validation set, and the column *Selected model* which model, among RNN-A, RNN-π/Π and RNN-π/Π-

μ/M performs best on the validation set. The MSE/SMAPE scores are computed on the test set.

Table 3.5: Univariate prediction on datasets with missing values and gaps, MSE(SMAPE)

Dataset	Selected-π/Π	Selected-π/Π-μ/M	RNN-A	RNN-π/Π	RNN-π/Π-μ/M	*Selected model*
PSE5	π	Π-μ	0.064*(0.345)	0.059*(0.316)	**0.055**(0.31)	Π-μ
PSE10	π	π-μ	0.066*(0.353)	0.055*(0.318)	**0.052**(0.314)	π-μ
PSE15	Π	π-μ	0.09*(0.36)	0.083(0.357)	**0.081**(0.332)	π-μ
PSE20	π	π-μ	0.079*(0.374)	0.078*(0.369)	**0.074**(0.344)	π-μ
PSE30	π	π-μ	0.104*(0.419)	0.102*(0.386)	**0.098**(0.384)	π-μ
PSE40	π	Π-M	0.113*(0.428)	0.119*(0.411)	**0.106**(0.393)	Π-M
PW5	π	π-M	0.161(0.556)	**0.157**(0.555)	0.164*(0.566)	π
PW10	π	π-M	0.16*(0.557)	**0.153**(0.54)	0.155(0.542)	π
PW15	π	π-μ	0.167(0.569)	0.167(0.556)	**0.163**(0.55)	π-μ
PW20	π	Π-M	0.209*(0.6)	0.182(0.551)	**0.177**(0.553)	Π-M
PW30	π	π-M	0.177*(0.571)	0.163(0.556)	**0.161**(0.549)	π-M
PW40	π	π-μ	0.181*(0.568)	0.172(0.564)	**0.164**(0.534)	π
AQ	Π	π-μ	0.299*(0.74)	**0.249**(0.668)	0.262*(0.665)	π-μ
AQ30	π	Π-μ	0.457*(0.879)	**0.35**(0.768)	0.357(0.797)	Π-μ
AQ40	π	π-μ	0.45*(0.881)	0.435*(0.819)	**0.403**(0.853)	π-μ
AEP5	π	Π-M	0.028*(0.095)	0.027(0.093)	**0.027**(0.096)	Π-M
AEP10	π	π-M	0.038*(0.128)	**0.034**(0.111)	0.038*(0.114)	π-M
AEP15	Π	π-M	0.042*(0.14)	0.044*(0.118)	**0.04**(0.123)	A
AEP20	π	Π-μ	0.036*(0.111)	0.035*(0.11)	**0.035**(0.105)	A
AEP30	π	π-μ	**0.052**(0.119)	0.057*(0.134)	0.06*(0.147)	A
AEP40	π	π-μ	**0.055**(0.124)	0.06*(0.138)	0.08*(0.16)	π-μ
OLD5	π	Π-M	0.333*(0.534)	0.32(0.582)	**0.282**(0.625)	π
OLD10	Π	Π-M	0.457*(0.761)	**0.321**(0.621)	0.332(0.707)	Π
OLD15	Π	Π-M	0.305(0.565)	0.36*(0.611)	**0.288**(0.6)	Π
OLD20	Π	Π-M	**0.297**(0.521)	0.358*(0.657)	0.484*(0.826)	Π
OLD30	Π	Π-M	0.37(0.581)	0.461*(0.836)	**0.358**(0.71)	Π
OLD40	Π	Π-M	**0.348**(0.648)	0.38(0.614)	0.362(0.676)	Π-M
NAB5	π	Π-μ	0.739*(0.523)	**0.702**(0.484)	0.973*(0.536)	Π-μ
NAB10	π	π-M	0.758*(0.512)	0.767*(0.467)	**0.712**(0.465)	π-M
NAB15	π	π-M	0.95*(0.556)	1.014*(0.588)	**0.727**(0.47)	π-M
NAB20	Π	π-μ	**1.152**(0.656)	1.324*(0.631)	1.364*(0.639)	A
NAB30	Π	π-M	1.309*(0.69)	1.1*(0.609)	**0.952**(0.551)	π-M
NAB40	π	π-μ	1.287*(0.695)	1.44*(0.684)	**1.012**(0.547)	π-μ

First, as one can note from the first two *selected* columns of Table 3.5, the different proposals introduced here are important and help to improve the results for time series forecasting. Indeed, from the methods for handling periods, RNN-π is selected 23 times (out of 33 cases) while RNN-Π is selected 10 times. Among the methods for handling both periods and missing values (column *Selected*-π/Π-μ/M), RNN-π-μ is selected 11 times (33%), RNN-π-M 8 times (24%), RNN-Π-μ 4 times (12%), and RNN-π-M 9 times (27%). If RNN-π-μ is the preferred method for handling both periods and missing values for `PSE` and RNN-π-M is the preferred one for `OLD`, there

are no clear preferences for handling both periods and missing values for PW, AEP, and NAB.

We first analyze the forecasting performances by focusing on each dataset separately and later present overall performances. For PSE dataset, from 5% to 40% missing, RNN-π/Π-μ/M outperforms RNN-A significantly in all six cases and RNN-π/Π significantly in five cases. In addition, the *selected* models are concordant with the best performing method in all six settings. Similarly, for PW RNN-π/Π-μ/M and RNN-π/Π are the best performing models and are often significantly better than RNN-A. For PW, in four out of six cases, *i.e.* for 15%, 20%, 30%, and 40% of missing values, RNN-π/Π-μ/M outperforms RNN-π/Π and, except for PW5, it outperforms RNN-A. Furthermore, the selected model mostly agrees with the best performing method. For AQ, RNN-π/Π and RNN-π/Π-μ/M are the best performing ones in all three missing settings (original; up to 20 %, 30%, and 40%) and the best performing one always significantly outperforms RNN-A. RNN-π/Π-μ/M significantly outperforms RNN-π/Π for AQ40, and RNN-A for AQ, AQ30, and AQ40. Lastly, RNN-π/Π is the best predicting method for AQ and AQ30. RNN-π/Π-μ/M is overall the best method for AEP (in particular for AEP5, AEP15 and AEP20), followed by RNN-A and RNN-π/Π. A similar behaviour is observed for OLD where RNN-π/Π-μ/M is the best performing method for OLD5, OLD15 and OLD30, followed by RNN-A and RNN-π/Π. For AEP and OLD, the selected model does not always coincide with the best performing method. We conjecture here that additional data is needed to really assess which model is the best one on the validation set[5]. Lastly, RNN-π/Π-μ/M is the best performing method on NAB with 10%, 15%, 30%, and 40% of missing values whereas RNN-π/Π performs the best on 5% of missing values. Moreover, the selected model mostly agrees with the best performing method (in five out of six cases).

For multivariate prediction with missing data we focused to RNN-Π^m and RNN-Π^m-μ/M methods, since Section 3.5.2.2 showed that RNN-Π^m was the main selected method in a multivariate setting. Table 3.6 illustrates the MSE values of multivariate prediction on the same multivariate datasets as the ones used in Section 3.5.2.2, with

[5]This is especially true for OLD that has few instances and a relatively long history and forecast horizon.

Table 3.6: Multivariate prediction on datasets with missing values and gaps, MSE

Dataset	RNN-A	RNN-Π^m	RNN-Π^m-μ/M	*Selected model*
AQ	0.352*	0.34*	**0.314**	Π^m
AQ30	**0.488**	0.646*	0.539*	Π^m-μ/M
AQ40	0.825*	0.773*	**0.503**	Π^m-μ/M
AEP5	**0.051**	0.089*	0.062*	Π^m-μ/M
AEP10	0.09*	**0.067**	0.072*	Π^m-μ/M
AEP15	0.084*	**0.065**	0.085*	A
AEP20	**0.062**	0.068*	0.081*	A
AEP30	**0.09**	0.1*	0.116*	A
AEP40	0.154*	**0.132**	0.166*	Π^m-μ/M
OLD5	0.379*	0.366*	**0.332**	Π^m-μ/M
OLD10	0.386	**0.343**	0.381*	Π^m-μ/M
OLD15	0.442*	0.333	**0.327**	A
OLD20	0.437	**0.402**	0.661*	Π^m
OLD30	0.359*	0.419*	**0.308**	Π^m-μ/M
OLD40	0.373*	**0.349**	0.398*	Π^m-μ/M

5%, 10%, 15%, 20%, 30%, and 40% of missing values. For `AQ`, RNN-Π^m-μ/M is the best performing method in two cases out of three, followed by RNN-A. In all three cases, RNN-Π^m-M is chosen as the RNN-Π^m-μ/M method. For `AEP`, RNN-Π^m with 10%, 15%, and 40% and RNN-A with 5%, 20%, and 30% outperform the other methods. For `AEP`, RNN-Π^m-μ is chosen in all six cases as the best RNN-Π^m-μ/M method. For `OLD`, RNN-Π^m with 5%, 15%, and 30%, and RNN-Π^m-μ/M with 10%,20%, and 40% are the best performing methods, yielding results significantly better than the ones of RNN-A. Here, the best performing method and the selected model do not agree in most cases.

Overall, the results obtained are better than the ones obtained in the univariate case. Furthermore, in 73% of the cases, RNN-A is outperformed by RNN-Π^m or RNN-Π^m-μ/M. As before, the complexity of the datasets may explain that it is difficult to select the best model on the validation set. In particular, in multivariate prediction, the missing portions of the dataset do not necessarily occur at the same time for the different variables, leading to potentially a high proportion of time stamps where at

least one value is missing.

3.6 Conclusion

We studied in this chapter the use of sequence-to-sequence RNNs, in particular the state-of-the-art bidirectional LSTM encoder-decoder with a period-aware content attention model, for modelling and forecasting time series. If content attention models are crucial for this task, they were not designed for time series and currently are deficient as they do not capture periods. We thus proposed three extensions of the content attention model making use of the (relative) positions in the input and output sequences. These models are combined with novel models and architectures to efficiently handle missing values and gaps in time series. The experiments we conducted over several univariate and multivariate time series demonstrate the effectiveness of these extensions, on time series with either clear periods, as `PSE`, or less clear ones, as `AEP`. Indeed, these extensions perform significantly better than the original attention model as well as state-of-the-art baseline methods based on ARIMA and random forests.

Chapter 4

Predicting Next Queries in Sessions: a Study on the TREC Search Session Track

In this chapter, we study the prediction of next queries in search sessions. To do so, we first analyze how query length and query topics evolve over interactions; we then analyze, among the sources available (past queries in the session, past retrieved documents, past clicked documents), the parts where users tend to select words to form new queries. From these analyses, we develop two different methods to predict next queries in search sessions; the first approach considers queries as bag-of-words and relies on learning to rank to predict next queries while the second approach considers queries as word sequences and makes use of recurrent neural networks. The experimental evaluation conducted on the TREC 2014 session track dataset shows that predicting next queries is feasible and that the final query obtained is as good, in terms of retrieval performance, as the final query formulated by users at the end of their interaction with the IR system.

4.1 Introduction

Search engines are at the core of exploiting information on the Web. A simple search engine applies a matching process between the user's query and web documents.

However, using a single query, users may not find the information they need and additional queries are often necessary in order to retrieve more relevant documents. Starting from a first query and gradually modifying it through interactions with a search engine until one gets documents answering the information need defines the search session paradigm [82]. This paradigm has often been seen as more realistic than the standard paradigm viewing search as the mere problem of finding relevant documents to a single query. This said, the lack of complete search session datasets is an obstacle to the development of the search session paradigm. Some datasets, as the AOL dataset, do contain query logs that constitute an important source of information for query suggestion. However, such datasets do not contain the retrieved documents associated to the queries and are thus limited in the information they carry regarding the interaction of users with an IR system.

To our knowledge, the only complete search session dataset is the one associated with the TREC search session track; we focus on this dataset in this study. In the session track, a session is composed of several consecutive interactions, where each interaction comprises a query, a set of retrieved documents (with titles and snippets visible to the user) and a (potentially empty) set of clicked documents. Our goal here is to assess whether one can predict, at any point in a session, the next query using the information collected from the previous interactions in the session. We will use the term *current* interaction (and similarly *current* query, *current* retrieved documents and *current* clicked documents) to refer to the last observed interaction; the term *past* interactions (and similarly *past* queries, *past* retrieved documents and *past* clicked documents) to refer to the interactions in the session before the current one; and the term *previous* interactions (and again similarly *previous* queries, *previous* retrieved documents and *previous* clicked documents) to refer to both the past and current interactions.

Several studies have shown that query reformulation can have an important impact on the retrieval process (as for example [58, 158]). Our study differs from previous ones in that (a), contrary to most previous studies on query suggestion that solely exploit search or query logs (with the exception of [214]), we assess here whether additional sources, as the snippets of top retrieved documents, are important for

query prediction and (b), contrary to methods developed in the context of the TREC search session track, we do not aim at developing a new IR system for sessions or at proposing an expanded final query that would lead to better retrieval results in the session track dataset; our objective is rather to predict, at any point in a session, the next query.

Our goal is motivated by the fact that reliably predicting next queries is a first step towards building systems that are able to simulate sessions. With such systems, one could possibly expand the current IR datasets based on single queries to sessions, *i.e.*, data augmentation and, from this simulated data, learn IR systems adapted to sessions. Indeed, the scarcity of search session data prevents one from really learning, *e.g.* through reinforcement learning, IR systems operating at the session level. We review here two methods to predict next queries in sessions.

The remainder of the chapter is organized as follows: Section 4.2 gives an overview of the most relevant related studies. Section 4.3 provides an analysis of the elements constituting next queries in search sessions. The results of this analysis will be used in Section 4.4 to define the general framework and the methods we propose to predict next queries. We then present in Sections 4.5 and 4.6 an experimental validation of our approach. Finally, Section 4.7 concludes the chapter.

4.2 Related Work

The majority of studies on query suggestion relies on query/search logs that are available in large amounts as no additional user intervention is required to generate such data. Early studies on query prediction based on query logs utilize similarities and common patterns between consecutive queries [97, 230, 193]. Another common approach considers the queries as states and attempts to model the transitions between these states as a sequence. In state-based approaches, states are defined based on adjacent query patterns derived probabilistically using n-grams, and the transitions are modeled using $(n-1)$ order Markov models [100]. Alternatively, the states and transitions between them can be captured by building a query-flow graph using query

logs [16]. [114] categorized the query reformulations tin terms of new, add, replace, remove and repeat operators, and studied the impact of users' cognitive styles on this matter. Very recent studies use recurrent encoder-decoder network [178] and recurrent sequence-to-sequence network with attention mechanism [46] to model states and transitions between them.

Unlike query logs, click through data and session information logs contain additional information on user activities, such as retrieved snippets, dwell time, clicked results etc. Many studies showed that utilizing search session logs, such as retrieved snippets, improves retrieval performance [154, 30]. Availability of additional information enables to have more accurate analytical approaches such as query understanding and suggestion based on analysis of retention, addition or removal of terms between consecutive queries [176], naive term frequency weighting based lexical query modeling [84], and probabilistic snippet click model on clicked snippets of previous query to predict the current query [132]. It also enables the use of more complex models such as Variable Memory Markov models for sequential query prediction [88], Markov Decision Process (MDP) for a query change model, where queries are states of MDP and user actions are the changes in the queries [83, 221], and Partially Observed Markov Decision Process (POMDP) to model user and search engine actions as a stochastic win-win game [135, 134, 220].

Sequential approaches utilize the co-occurrence of words or any other available information on consecutive queries. This strategy however under-represents rare queries [174]. To alleviate this, [25] proposed a bag-of-word representation of not only the words in the query, but also the words of the queries in the same session, giving a *set* of candidate words. [165] improved this representation by adding information about the source (e.g. related queries in a common session) of every candidate word. [165] learns a ranking function to rank the candidate suggestions based on features containing its source, its popularity in terms of clicks and its performances based on classical IR scoring functions. [176] used words from previous queries, snippets and clicked documents as the source of the new query for a term based query reformulation. Based on this evidence, the current study further extends these studies by exploiting the various sources available in the TREC search session track. To the best of our

knowledge, this is the first study to fully exploit all sources of information present in search sessions to predict next queries. Like [165], one of our proposed method also employs learning to rank methods based here on a carefully crafted novel feature set incorporating the source(s) of a candidate word and its similarity with the past queries.

With the advent of deep learning, and more specifically of Recurrent Neural Networks (RNNs) for developing sequence-to-sequence models, [178] proposed a hierarchical sequential approach where sessions are viewed as sequences of queries which are in turn sequences of words. [46] extended this approach with an attention mechanism. However, these networks are trained using the AOL query logs, not taking into account other possible and rich sources of information as the retrieved document snippets of past queries. [214] extends earlier hierarchical sequential approach ([178]) by leveraging click through information as a relevance feedback. The word embedding in the clicked and the skipped documents respectively represents the positive and negative feedback. We propose here an extension of these models that leverages extensive session information by incorporating words from previous queries, snippets, retrieved and clicked documents.

4.3 Session analysis

Table 4.1: Basic statistics on the TREC 2014 data (entire set)

Length	1	2	3	4	5	6
#sessions (%)	NA	256(25%)	210(21%)	187(18%)	143(14%)	106(10%)
#queries (%)	176(4%)	1174(28%)	1520(36%)	867(21%)	324(8%)	117(3%)

The TREC 2014 session track data contains 1,257 session logs performed on ClueWeb12-Full document collection[1]. Each session contains one or more interactions and associated retrieved documents, click information and sometimes a final

[1]https://lemurproject.org/clueweb12/

query. The ClueWeb12 collection contains approximately 7.3 billion English documents, obtained by crawling the Web. We also use the same corpus for evaluating our methods. For the of query prediction, at least one interaction is needed to predict the next query, thus, requiring the presence of at least two queries in a session. We furthermore require a session to contain a final query, so as to be able to compare the quality of the final query predicted through the methods studied here with the quality of the final query provided in the session track dataset. After cleaning, the dataset used in this study has a total of 1021 sessions which in turn contain 4226 interactions. Table 4.1 illustrates the basic statistics on the entire set.

The table displays the distribution of session and query lengths over the entire data. The session length corresponds to the number of queries in the session, whereas the query length corresponds to the number of words making up the query. As one can observe, most of the sessions and queries are of length 2 to 4. In particular, more than half of the queries have a length of 3 or 4, whereas most sessions contain at most 4 queries. Few sessions and queries have a length exceeding 6.

In the following, we analyze here several properties of queries over successive interactions: query length, query cohesion and the origin of the words forming next queries.

4.3.1 Evolution of query length

Table 4.2: $P(l_{q^{t+1}}|l_{q^t})$ for the TREC 2014 data (training set). Darker cells correspond to higher probabilities.

l_{q^t} / $l_{q^{t+1}} = l_{q^t} + (\cdot)$	-3	-2	-1	0	$+1$	$+2$	$+3$
1 (89 queries)	0.0	0.0	0.0	0.318	0.398	0.239	0.045
2 (701 queries)	0.0	0.0	0.122	0.543	0.239	0.079	0.017
3 (979 queries)	0.0	0.015	0.285	0.498	0.161	0.032	0.009
4 (560 queries)	0.005	0.079	0.373	0.401	0.108	0.029	0.005
5 (218 queries)	0.083	0.176	0.343	0.329	0.060	0.009	0.0
6 (76 queries)	0.097	0.25	0.264	0.347	0.042	0.0	0.0
7 (19 queries)	0.200	0.267	0.333	0.200	0.0	0.0	0.0

To further examine the evolution of query length, we computed the distribution of the next query length given the length of the current query. Table 4.2 displays the proportions of next queries whose length (denoted as $l_{q^{t+1}}$) equals the length of the current query (denoted as l_{q^t}) plus a relative number that varies in $\{-3, ..., 3\}$. Less than 1% of the next queries have a length outside this range and one can not derive reliable estimates for these cases. The proportions displayed in Table 4.2 are normalized so as to correspond to empirical estimates of the probability of the next query length given the current query length. Higher probabilities are shown with darker shades of gray. As one can note, the distribution of the next query length highly depends on the length of the current query. It is for example more likely (16.1%) to form a query of length 4 from a query of length 3 than from a query of length 2 (7.9%). As such, one can only use conditional probabilities to predict the length of the next query.

4.3.2 Evolution of query cohesion

We study here the evolution of queries over consecutive interactions with respect to the topics (in the sense of themes) explored by users. In particular, we are interested in assessing whether users formulate cohesive queries, thus pertaining to few topics, or non-cohesive queries with words pertaining to different topics. To do so, rather than resorting to models aiming at capturing topics as such models are error prone, in particular concerning the number of topics, we compute an average similarity between words within queries and between words in consecutive queries. This approach neither requires the determination of the number of topics nor the identification of the topics.

To compute the similarity between words, we first computed word embeddings for all the words occurring in the queries with the continuous bag-of-words (CBOW) model[2] [**?**]. To do so, we collected the documents occurring in the session data (*ca.* 33K documents) and combined them with another 70K random documents from Clueweb12[3] to have sufficient training data; embeddings were initialized to Google's

[2] radimrehurek.com/gensim/models/word2vec.html

[3] `https://www.lemurproject.org/clueweb12.php/`

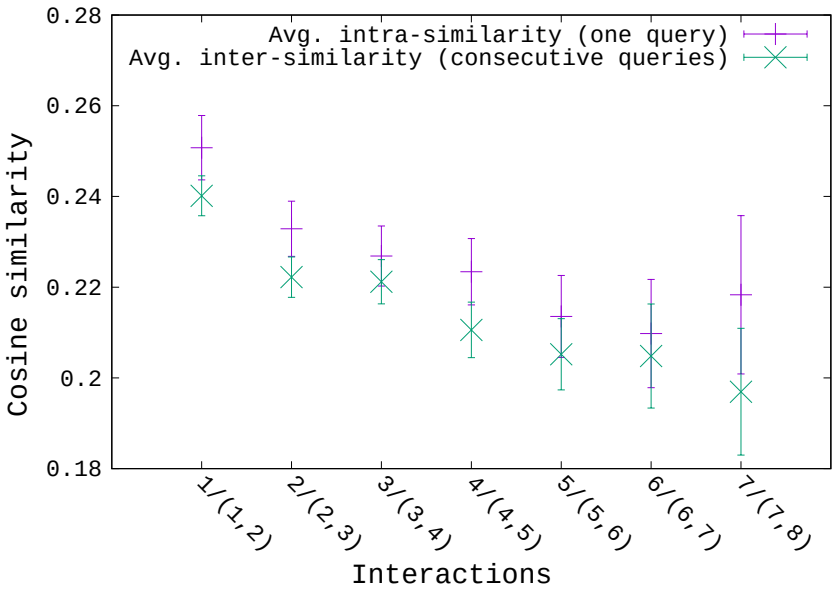


Figure 4.1: Comparison between the inter- and intra-similarity between terms of one query and terms of two successive queries with 95% confidence intervals.

pre-trained embeddings [4] for those words present in this resource, and randomly initialized otherwise. We then computed the average cosine similarity between word embeddings within queries and between consecutive queries. Figure 4.1 displays the results obtained, where the x-axis corresponds to the interaction number in the session (the similarities are averaged over all sessions in the training set). As one can note, the average intra-similarity (within queries) is significantly lower than the average inter-similarity (between consecutive queries) as in successive queries there are often pairs with identical words. Both inter- and intra-similarities decrease during the first two interactions and then stabilize. This corresponds to a typical exploration phase by users at the beginning of the session.

Another interesting point here is the fact that the average similarities are rather high. Indeed, the average cosine similarity between all possible pairs of query words, excluding pairs of identical words, is 0.070 whereas the smallest average similarity displayed in Figure 4.1 is almost three times higher (0.209). This indicates that

[4] http://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/

Table 4.3: Different sources for query words along with their precision, recall and cumulative recall. The sources are sorted in the descending order of their precision. (current interaction (CI), past interactions (PIs), retrieved documents (ret docs), snippets (snips), query words (QWs)).

	# of next QWs	6030		
	words from...	precision	recall	cumulative recall
visible sources	past queries	68.80%	35.69%	61.82%
	current query	60.11%	58.06%	61.82%
	titles of the docs of CI	1.56%	58.51%	69.22%
	titles of the docs of PIs	1.10%	41.09%	72.01%
	snips of the ret docs of CI	0.31%	67.46%	76.40%
	snips of ret docs of PIs	0.22%	47.71%	78.44%
	clicked docs of CI	0.09%	17.15%	79.29%
	clicked docs of PIs	0.08%	15.32%	79.83%
non-visible sources	top 2 ret docs of CI	0.07%	57.71%	82.19%
	top 2 ret docs of PIs	0.05%	43.62%	83.47%
	top 5 ret docs of CI	0.03%	70.03%	84.96%
	top 5 ret docs of PIs	0.02%	51.04%	85.69%
	ret docs of CI	0.02%	77.55%	87.21%
	ret docs of PIs	0.01%	54.51%	87.83%
	# of QWs from no above sources		734 (12.17%)	100%

queries are indeed rather cohesive, and that users do not change radically the topics in between two consecutive queries.

4.3.3 Where do query words come from?

The next step of our analysis is devoted to detecting the source of query words. Having an information need in mind, a user starts with an initial query (interaction 1). In response, the IR system returns a ranked list of documents. For each document, the title and a snippet are visible to the user. Among these retrieved documents some (or none) are clicked. Then the user reformulates the query to proceed with the next interaction, where this chain of events is repeated. Among the sources of information

associated to past interactions, *i.e.* past queries and past retrieved documents, one can distinguish between words that are visible to the user, namely words in the past queries, in the titles and snippets of the past retrieved documents and in the past clicked documents, and those that are not seen by the users, such as the words from an unclicked document that are not part of the title and snippet of that document. Intuitively, visible words play a significant role in the construction of the next query. Of course, users will also use words, when formulating their next query, that are not associated with any of the information sources above. However, we hypothesize here that the majority of the words present in the next query are also found in the past interactions and that, among those words, the ones coming from the visible sources (past queries, titles and snippets of retrieved documents and clicked documents) play a particular role.

To assess this hypothesis, we first divided the sets of information sources according to whether they are visible or not and to whether they relate to the current or past interactions, as current interactions may play a more important role than past ones. The latter division is based on the rationale that words appearing in the top retrieved documents are more likely to appear in next queries as they are more related to the current query words. In the non-visible sources, we further separated the top 2, top 5 and all the retrieved documents to assess the importance of the rank. We then computed the precision, recall and cumulative recall of all sources for the next query words. The precision of a source is defined as the average of the proportion of words from the source that appear in a next query, where the average is computed over all sessions in the training and all next queries, *i.e.* from the query of the second interaction to the final query. Precision provides a measure of the reliability of a source when predicting next query words. The recall of a source is defined as the average of the proportion of words from the next query that are found in the source, where the average is again computed over all sessions in the training set and all next queries. Recall indicates whether the source is an important source to extract next query words. Lastly, the cumulative recall is defined as the average of the proportion of words from the next query that are found in the sources considered so far, where the sources are sorted according to their precision. As before, the average is computed

over all sessions in the training set and all next queries. Cumulative recall thus gives an indication of whether a combination of the most reliable sources provides or not an important proportion of words used in the next queries.

The results obtained for these different measures on the different sources are displayed in Table 4.3. Although the visible sources account for almost 80% of cumulative recall, they miss an accountable portion of words form the next query. Furthermore, using all the sources constitutes 88% of cumulative recall. The sources based on the current interaction are furthermore more important, as measured by recall, the precision being comparable, than the ones from past interactions. The top retrieved documents are also more reliable (higher precision) than all retrieved documents. This said, their recall, not surprisingly, is lower. Lastly, the proportion of words that are not found in previous interactions amounts to 12.17%, confirming our hypothesis that most words are present in previous interactions. We make use of these findings in the next section to select and represent the words that are likely to appear in next queries.

4.4 Next query prediction models

A session s is composed of l_s consecutive queries $Q_s = \left(q_s^1, \ldots, q_s^{l_s}\right)$, of their associated retrieved documents $R_s = \left(R(q_s^1), \ldots, R(q_s^{l_s})\right)$, their associated clicked documents $C_s = \left(C(q_s^1), \ldots, C(q_s^{l_s})\right)$ and the snippets of the retrieved documents $S_s = \big(S(q_s^1), \ldots, S(q_s^{l_s})\big)$. The retrieved documents for query q_s^t at time t is the list of the K top-ranked documents[5] for query q_s^t: $R(q_s^t) = \left(r_1(q^t), \ldots, r_K(q^t)\right)$. The clicked documents for query q_s^t is the (potentially empty) list of the documents from $R(q_s^t)$ clicked by the user. The snippet of the k^{th} retrieved document for q_s^t is denoted as $s_k(q_s^t)$. Lastly, let $V^{s,t} = \{\mathbf{v}_1^{s,t}, \ldots, \mathbf{v}_{|V^{s,t}|}^{s,t}\}$ represent the set of all words from the previous interactions. In the remainder, we will denote by $\mathcal{I}_s^{(1:t)}$ the above information restricted to the first t interactions of session s and by l_q the length of query q. For the sake of simplicity, we drop the index of the session, s, when the context is clear. Lastly, following the

[5]K=10 for the TREC 2014 session track.

results of the previous section, we assume that all next query words originate from one of the sources in $\mathcal{I}_s^{(1:t)}$: previous queries, snippets, clicked documents and retrieved documents.

There are two standard ways to look at a query. The most common one is to consider that a query is a *bag of independent words*. In that case, the probability of the next query q^{t+1} given $\mathcal{I}^{(1:t)}$ takes the form:

$$\begin{aligned} P(q^{t+1}|\mathcal{I}^{(1:t)}) = & \quad P(l_{q^{t+1}}, w_1^{t+1}, \cdots, w_{l_{q^{t+1}}}^{t+1}|\mathcal{I}^{(1:t)}) \\ = & \quad P(l_{q^{t+1}}|\mathcal{I}^{(1:t)}) P(w_1^{t+1}, \cdots, w_{l_{q^{t+1}}}^{t+1}|l_{q^{t+1}}, \mathcal{I}^{(1:t)}) \\ = & \quad P(l_{q^{t+1}}|l_{q^t}) \prod_{i=1}^{l_{q^{t+1}}} P(w_i^{t+1}|\mathcal{I}^{(1:t)}) \end{aligned} \tag{4.1}$$

where $w_1^{t+1}, \cdots, w_{l_{q^{t+1}}}^{t+1}$ is the set of words in query q^{t+1} and where the last equality is based on the fact that words are independent given previous interactions (bag-of-word assumption) and that the length of the next query is here assumed to depend only on the length of the current query. Predicting the next query at time $(t+1)$ thus amounts to selecting the set of $l_{q^{t+1}}$ words that maximizes:

$$\max_{\{w_1^{t+1}, \ldots, w_{l_{q^{t+1}}}^{t+1}\}} P(l_{q^{t+1}}|l_{q^t}) \prod_{i=1}^{l_{q^{t+1}}} P(w_i^{t+1}|\mathcal{I}^{(1:t)}) \tag{4.2}$$

As one can note, this approach requires to determine the length $l_{q^{t+1}}$ of the next query and the set of $l_{q^{t+1}}$ words.

The second approach views a query as a *sequence* of words. In this case, Eq. (4.1) is replaced by:

$$\begin{aligned} P(q^{t+1}|\mathcal{I}^{(1:t)}) \quad = \quad & \prod_{i=1}^{l_{q^{t+1}}} P(w_i^{t+1}|w_1^{t+1}, \cdots, w_{i-1}^{t+1}, \mathcal{I}^{(1:t)}) \\ & \mathrm{P}(\epsilon|w_1^{t+1}, \cdots, w_{l_{q^{t+1}}}^{t+1}, \mathcal{I}^{(1:t)}) \end{aligned} \tag{4.3}$$

where ϵ is a symbol that denotes the end of the sequence. Predicting the next query amounts now to selecting the sequence of words that maximizes:

$$\begin{aligned} \max_{w_1^{t+1} \cdots w_{l_{q^{t+1}}}^{t+1} \epsilon} \quad & \prod_{i=1}^{l_{q^{t+1}}} P(w_i^{t+1}|w_1^{t+1}, \cdots, w_{i-1}^{t+1}, \mathcal{I}^{(1:t)}) \\ & \mathrm{P}(\epsilon|w_1^{t+1}, \cdots, w_{l_{q^{t+1}}}^{t+1}, \mathcal{I}^{(1:t)}) \end{aligned} \tag{4.4}$$

We propose below solutions to these two maximization problems.

4.4.1 Set View: Learning to Rank

The study of the evolution of query length in Section 4.3 provides a direct and simple way to estimate the conditional query length probabilities in Eqs. (4.1) and (4.2): $P(l_{q^{t+1}}|l_{q^t})$ is set to the empirical estimates of Table 4.2. The other quantity in these equations, $P(w_i^{t+1}|\mathcal{I}^{(1:t)})$, induces a ranking of the words (from the different sources in $\mathcal{I}^{(1:t)}$) according to their probability to be part of the next query. We propose here to estimate this probability through learning to rank, a method that has proven successful in different contexts and that is widely used in IR.

To do so, for each interaction and each session, we first represent, based on the results of Section 4.3, each word from the previous interactions as vectors the dimensions of which are associated with properties of the word in the different sources. Table 4.4 gives the 14 features retained in this study. These features aim at capturing the characteristics of a word over the previous interactions. We describe here the first 7 features, pertaining to the current interaction, as the last 7 ones are just generalizations to past interactions. The first feature corresponds to the number of occurrences of the word (w) under consideration in the current query. The second feature corresponds to the term frequency of w in the retrieved documents of the current query whereas the third one is the term frequency in the clicked documents. The fourth feature corresponds to the document frequency of words in the current query; it is thus 0 if w is not part of the current query. The fifth feature is the minimum rank of w in the current retrieved documents; it is set here to 1000 if w does not occur in any retrieved documents. The sixth feature plays the same role but with the average rank. Lastly, the seventh feature is the average cosine with the current query words. Note that $\mathbb{1}_{\text{condition}}$ is 1 if condition is true, and 0 otherwise.

We then form, again for each interaction and each session, N_p pairs of words where each pair contains a randomly chosen *positive* word (*i.e.* a word from the previous interactions that occurs in the next query) and a randomly chosen *negative* word (*i.e.* a word from the previous interactions that does not occur in the next query). Each pair is represented as the difference between the vectors of the words it contains. The label of a pair is $+1$ if the first word of the pair is positive, and -1 otherwise.

1	$tf(w, q^t)$
2	$\sum_{d \in R(q^t)} tf(w, d)$
3	$\sum_{d \in C(q^t)} tf(w, d)$
4	$df(w) \times \mathbb{1}_{w \in q^t}$
5	$\text{argmin}_{i, 1 \leq i \leq K} \mathbb{1}_{w \in r_i(q^t)}$; 1000 if $w \notin R(q^t)$
6	$(1/K) \sum_{i, 1 \leq i \leq K} \mathbb{1}_{w \in c_i(q^t)}$; 1000 if $w \notin C(q^t)$
7	$(1/l_{q^t}) \sum_{i=1}^{l_{q^t}} cos(w_i^t, w)$
8	$\sum_{t'=1}^{t-1} tf(w, q^{t'})$
9	$\sum_{t'=1}^{t-1} \sum_{d \in R(q^{t'})} tf(w, d)$
10	$\sum_{t'=1}^{t-1} df(w) \times \mathbb{1}_{w \in q^{t'}}$; 0 if $w \notin \bigcup_{t', 1 \leq t' \leq t-1} \{q^{t'}\}$
11	$\min_{t', 1 \leq t' \leq t-1} \text{argmin}_{i, 1 \leq i \leq K} \mathbb{1}_{w \in r_i(q^{t'})}$; 1000 if $w \notin \bigcup_{t', 1 \leq t' \leq t-1} \{R(q^{t'})\}$
12	$\frac{1}{K(t-1)} \sum_{t'=1}^{t-1} \sum_{i, 1 \leq i \leq K} \mathbb{1}_{w \in r_i(q^{t'})}$; 1000 if $w \notin \bigcup_{t', 1 \leq t' \leq t-1} \{R(q^{t'})\}$
13	$\sum_{t'=1}^{t-1} \sum_{d \in C(q^{t'})} tf(w, d)$
14	$1/(t-1) \sum_{t'=1}^{t-1} (1/l_{q^t}) \sum_{i=1}^{l_{q^t}} cos(w_i^t, w)$

Table 4.4: Features for the words in previous interactions for next query prediction

Furthermore, to obtain a balanced data set, each time a pair is chosen, its symmetric (switching the two words) is also considered. The collection of all such pairs over all interactions of our training sessions constitutes the training set for the learning to rank system. As we will see in Section 4.5, we make use here of state-of-the-art pair-wise and list-wise learning to rank methods. Note that N_p is an hyperparamater that can be tuned by cross-validation. Considering the same number of pairs for each interaction allows one to avoid the bias induced by long queries [29].

Learning to rank systems provide, for each interaction in each session, a ranked list of words with scores that reflect the likelihood of each word to be present in the next query. Given the length of the new query $l_{q^{t+1}}$, the words that maximize the probability in Eq. (4.2) are the top $l_{q^{t+1}}$ words. It may be beneficial however, as shown in *e.g.* [178], to generate more than one candidate next query and to use a second-level ranker to select the best ones. We adopt this approach and present below

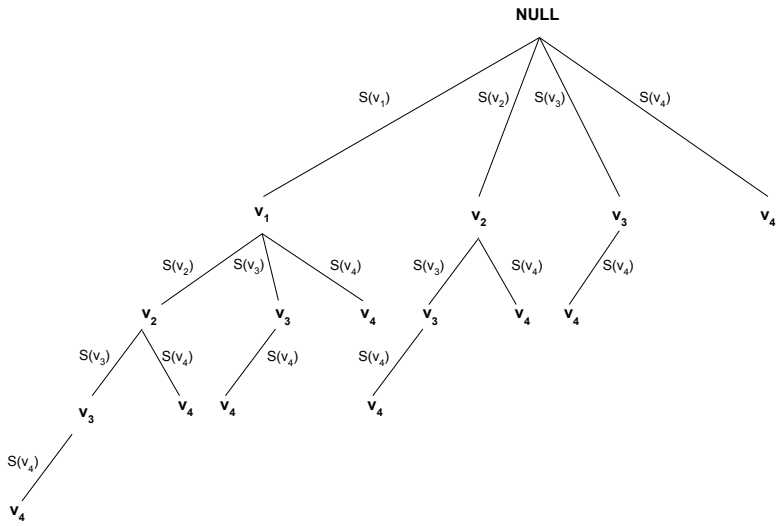


Figure 4.2: An example of the tree with a set of 4 words, $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$, sorted based on the likelihood scores, $\{S(\mathbf{v}_1), S(\mathbf{v}_2), S(\mathbf{v}_3), S(\mathbf{v}_4)\}$.

a way to generate the n best candidate queries of a given length from a ranked list of words with score. We can then generate candidates with different lengths using Table 4.2.

In order to find the best sets of candidates of fixed length, one can represent the sorted list of words as a tree from which the candidates can be generated efficiently. Having the list of words, sorted based on their likelihood scores, a weighted tree can be generated as follows: the root (level-0) is a null node whose children are all words of the list (level-1). The lower levels are then generated recursively by creating the children of each node using all the words which has lower scores than the node itself. The weight of each edge is then defined as the score of the node in the lower level. Figure 4.2 illustrates an example of such tree with a sorted list of words $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$ with scores $\{S(\mathbf{v}_1), S(\mathbf{v}_2), S(\mathbf{v}_3), S(\mathbf{v}_4)\}$.

Finding the best candidate queries of length l_q amounts in this case to finding the paths of length l_q between the root and all other nodes. This problem can be solved in $O(|V|log|V|)$ time [111], where $|V|$ is the size of the tree, corresponding to the number of (candidate) interaction words. To find the n best candidate queries of size l_q, one can repeat n times the process of finding the longest path of size l_q, removing

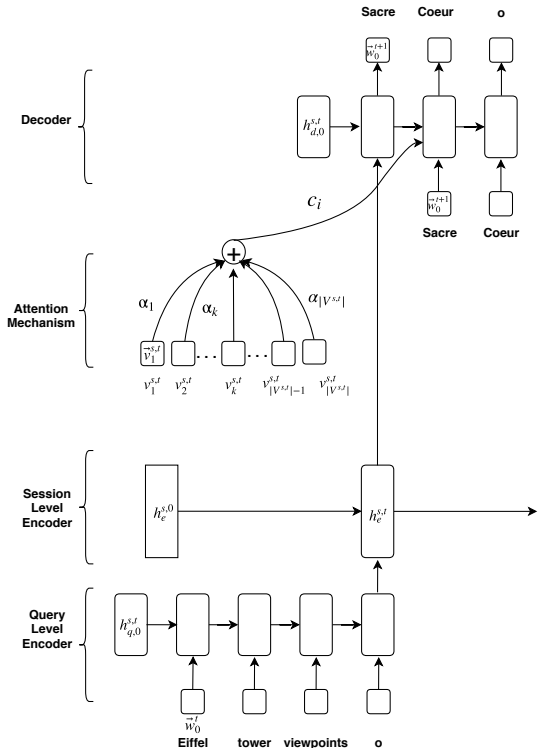


Figure 4.3: Sequence-to-sequence hierarchical RNN with an attention mechanism on embedding vectors of visible words.

the solution each time before repeating. Removing the path can be simply done by removing the edge between the first and the second element of the path. As query length is limited, the size of the tree (and hence the complexity) can be controlled by resorting to the top words in the ranked list. The score of a path (a candidate sequence) is then computed by averaging the weights of edges in the path, *i.e.* the scores of the words.

4.4.2 Sequence View: Hierarchical RNN

For the sequence view, we follow the approach originally proposed in [178] based on a hierarchical sequence-to-sequence RNN. The representation of words we adopt in this method is based on word2vec embeddings [**?**] that are learned using gensim[6] over a set of 103,645 documents from the ClueWeb12 collection, of which 33,647 documents

[6] `https://radimrehurek.com/gensim/models/word2vec.html`

are mentioned in the Session Track dataset, the additional 69,998 documents being selected randomly (see Section 4.5). The vectors are initialized by the Google's pre-trained embeddings which are of dimension 300 [7].

The model proposed in [178] contains a first standard RNN encoder for each query, followed by another encoder at the session level. A standard RNN decoder is then used to predict the next query. The intermediate, session level encoder allows one to pass information, to the decoder, from all previous queries. We augment here this model by explicitly considering, for predicting the next query, the words in the previous interactions. These words are integrated through an attention model aiming at selecting the word to be predicted (see Figure 4.3). While the representation used at the input of the encoder is solely based on word embeddings, the input representation for the attention mechanism is a combined vector whose first 14 dimensions correspond to the vector representation defined in Table 4.4 and whose remaining dimensions correspond to word2vec vector embeddings (of dimension 300). The decoder layer is followed by a linear transformation layer taking the decoder hidden state and the previous word as input to output the next word. The probabilities of the next words are calculated by a softmax over the dot product of the output word embedding and this linear transformation layer.

Let $V^{s,t} = \{\mathbf{v}_1^{s,t}, \ldots, \mathbf{v}_k^{s,t}, \ldots, \mathbf{v}_{|V^{s,t}|}^{s,t}\}$ represent the set of all words from the previous interactions, with $\mathbf{v}_k^{s,t}$ the embedding of the k^{th} word (here, we adopted fine-tuned word2vec embedding) and let $\mathbf{h}_{d,i}^{s,t}$ denote the decoder hidden state for the i^{th} word of the next query. The attention model [8] computes, for each next query word w_i^{t+1}, a context vector $\mathbf{c}_i$ defined by:

$$
\begin{aligned}
e_{i,k} &= \mathbf{u}_a^{\mathsf{T}} \tanh(\mathbf{W}_a^d \mathbf{h}_{d,i-1}^{s,t} + \mathbf{W}_a^v \mathbf{v}_k^{s,t}) \\
\alpha_{i,k} &= \frac{\exp(e_{i,k})}{\sum_m \exp(e_{i,m})} \\
\mathbf{c}_i &= \sum_k \alpha_{i,k} \mathbf{v}_k^{s,t}
\end{aligned}
$$

where $\mathbf{u}_a$ is a vector whose dimension equals the number of hidden units in the attention layer. The cross-entropy loss is used for training the model. Furthermore,

[7] `https://bit.ly/1R9Wsqr`

as for the learning to rank approach, we also generate n best candidate next queries, which can be done efficiently through beam search. Standard beam search is greedy and might lead to similar candidate queries. Hence, we explored the use of diverse beam search [126] and a modification of diverse beam search in which the scores of sibling nodes are re-weighted only at the first iteration (they are not adjusted during later iterations as opposed to standard diverse beam search). Our sequence-to-sequence model can finally be seen as a generalization of the model in [178] by the consideration of additional sources and their integration through an attention mechanism. In addition, to restrict the set of predicted words to the vocabulary of the previous interactions, we add a filter that gives an arbitrary negative values to words absent from this vocabulary just before the softmax operator of the RNN decoder, so that those words get a null value with the softmax function.

4.4.3 Second-Level Ranker

Both the set view (`SET`) and sequence view (`SEQ`) methods provide plausible candidate queries. Following the approach in [178], we employ a second-level ranker to choose the best query among the top n candidate queries generated from the previous approaches. Each of the candidate queries is represented using nine features in total, given below:

Six contextual features: These features measure the similarity between a candidate sequence and its context, *i.e.* true queries from previous interactions. This includes BLEU score, Levenshtein distance and Jaccard coefficient between the candidate query and the current true query q^t, and average BLEU score, average Levenshtein distance and average Jaccard coefficient between the candidate query and all the previous true queries $q^1, \ldots, q^t$.

Two complexity features: These include the length of the candidate queries in terms of characters and words. These features represent the complexity of the candidate queries.

Score of the previous method: The score of the candidate query provided by either `SET` or `SEQ` is also used as a feature representing the preferences of the candidate generator method used.

For each interaction in the training sessions, the true target query q^{t+1} is labeled as relevant, and the top n extracted candidates by `SET` or `SEQ` that differ from the true query as non-relevant. We aimed to assign an higher score to the true query and lower score to the other candidates. Using this training set we learn a ranker and apply it on the test session interactions to select the best candidate. This way we expect the second-level ranker to learn to re-rank the most-probable next query among the top n candidate queries. As in Section 4.4.1, state-of-the-art pair-wise or list-wise rankers are used (detailed in Section 4.5.3).

4.5 Experimental setup

Experiments are performed using the TREC 2014 Session track dataset [31], detailed in Section 4.3.

For the purpose of query prediction, at least one interaction is needed to predict the next query, requiring the presence of at least two queries in a session. We furthermore require a session to contain a final query, so as to be able to compare the quality of the final query predicted through the methods studied here with the quality of the final query provided. Hence, the sessions having only one interaction and the sessions without any final query are ignored. The final filtered dataset used in this study has 1021 sessions containing 4226 interactions. The session dataset contains 60 different topics and the topics are randomly distributed over sessions, giving a mix of topics both for train and test sets.

We divided the 1021 sessions used in this study through a two-level 5-fold cross validation. For outer-level cross validation, sessions are separated into 5 equal subsets of which a single subset is forming the outer-level validation partition and the left-out four subsets are forming the outer-level training partition. Each outer-level training

partition is used to form an inner 5-fold cross validation by dividing each outer-level training partition into 5 inner subsets. Among these 5 inner subsets a single subset forms an inner-level validation partition and the left-out four subsets form the corresponding inner-level training. This procedure is repeated such that each (outer or inner-level) subset once forms a validation set and the remaining four subsets form the corresponding training set. The hyperparameters of models generating candidate sequences (set or sequence view) are selected on the average validation error on the inner-level 5-fold cross-validation sets resulting in 5 separate models for each outer-level training-validation set. The score of the outer-level cross-validation, averaged over these 5 models, is reported in Section 4.6.

All the necessary modules are implemented in Java and Python. For preprocessing and indexing the ClueWeb12 collection, we used Indri[8]. Preprocessing steps include stopword removal and stemming using Porter stemmer. All the experiments are performed on a Linux machine with Intel Xeon CPU @3.40GHz with 128GB of memory. For the RNN models, we used Theano [9] with an NVIDIA TITAN X GPU with 12GB of memory.

4.5.1 Setup for the Set View Approach

The `SET` method assumes a bag-of-words approach in which the set of interaction words are considered as candidate words for predicting query of the next interaction (Section 4.4.1). Each candidate word is represented in a 14-dimensional feature space (Table 4.4). One can note that the candidate words from the first interaction of each session have no *previous interactions*, thus features 8-14 in Table 4.4 are not applicable for these words. To avoid this discrepancy, while applying the learning-to-rank algorithm for ranking the candidate words, we learned two independent rankers. One is for all the first interactions where candidate words are represented with first 7 features, giving a ranking of the words for the query of interaction 2. The other ranker is for the rest of the interactions. Both rankers are trained in parallel, using the training sessions.

[8] `https://www.lemurproject.org/indri.php`

[9] `http://deeplearning.net/software/theano/`

We experimented with two learning-to-rank algorithms based on pairwise and listwise approaches. For the pairwise approach, after choosing the N_p number of positive and negative pairs (Section 4.4.1), a SVM classifier is learned using LIBLINEAR [52]. The hyperparameter N_p, *i.e.* the number of pairs, and C of SVM are fixed by 5-fold cross-validation. We refer to the corresponding model as `SET-LL`, where `LL` is short for LIBLINEAR. For the listwise approach, we used the LambdaMART [27] implementation of the RankLib package[10]. The number of trees of LambdaMART is also chosen by 5-fold cross-validation. The corresponding model is defined as `SET-LM`, where `LM` is short for LambdaMART. The sets of values considered for cross-validation of different hyperparameters are provided in Table 4.5.

Table 4.5: Sets of values used for cross-validating different hyperparameters.

	Hyperparameter	Set of values
SET	N_p (# pairs)	$\{50, 75, 100, 150\}$
	C of SVM	$\{0.0001, 0.001, 0.01, 0.1, 1.0, 5.0, 10.0\}$
	#trees in LamdaMART	$\{250, 500, 1000, 1250, 1500\}$
SEQ	regularization coef.	$\{0.0001, 0.001\}$
	# hidden units (query, session, attention)	$(256, 512, 512), (512, 1024, 1024)\}$
	diversity coef. γ	$\{0.25, 0.5, 0.75, 1\}$

Once word-level probability scores have been obtained, one can build a weighted tree with possible words as explained in Section 4.4.1. The longest path on the tree of words provides candidate sequences of words with a corresponding score. Hence, we generate sufficently large number of candidates (here, we generate 100) for each probable next query length, *e.g.*, for a current query of a length 1, next query can be a length of one, two, three, or four (see Table 4.2). Here, the candidate score, calculated during tree search, does not consider how likely the the next query length given the current query length. Thus, we refine each candidate score according to

[10]https://sourceforge.net/p/lemur/wiki/RankLib/

the distribution of next query length (*i.e*, candidate length) given the current query length. This is done by multiplying the candidate score by the probability of this candidate query length given current query length, given in Table 4.2. For instance, if $l_{q^t} = 3$, the score of all generated candidates with $l_{q^{t+1}} = 4$ is multiplied by $P(l_{q^{t+1}} = 4|l_{q^t} = 3) = 0.161$.

To summarize, once all candidates of all possible lengths are generated and their scores are adjusted, the 20 candidate sequences having the best scores are used in the second-level ranker.

4.5.2 Setup for the Sequence View Approach

The network parameters of the proposed `SEQ` method are learned through stochastic backpropagation with mini-batch size of 16. Adaptive learning rates [113] with L_2 regularization are used for each parameter with an initial learning rate of 10^{-3}. All the network parameters are randomly initialized using a fixed seed. Two hyperparameters, namely the number of hidden units and the regularization coefficient, are fixed by applying a grid search using 5-fold cross-validation over the training set. Note that the number of hidden units refer to the number of units in the query-level (encoder-decoder RNNs), session-level and attention layers, both in the encoder and the decoder side (the sets of values used for each of these hyperparameters are provided in Table 4.5).

To investigate the trade-off between number of parameters and performance, we used vanilla RNN and GRU units in the query and session-level recurrent layer. GRU units achieve finer control over memory (history of sequence) with almost three times more parameters. In contrast, vanilla RNN units have more limited history capabilities but rely on less parameters. Our experiments revealed that using the same hyperparameter space and the same random seed, using GRU units results in better performance than using vanilla RNN units.

The number of interaction words vary drastically over different interactions, e.g., the words in retrieved documents for one query can be much larger than another query. This results in some interactions with a very large number of words, although

a majority of interactions remain with much less number of words. Therefore, for practical reasons we cut off the words in an interaction that account for more than a predefined threshold. This threshold, which is 2499 in our experiments, corresponds to the 75^{th} percentiles of the number of the words in all interactions. Furthermore, to maintain more useful words; we sort the interaction words according to their importance and cut off the words that are exceeding this threshold. The importance of a word is determined according to the values of this word's features $[1, 9, 6, 2, 14, 10]$, which are given in the decreasing order of importance. Features are defined in Table 4.4.

The candidate queries are generated by using standard, diverse beam search [126] and diverse beam search @1 with length normalization. Diverse beam search @1 is the modification of diverse beam search in which the scores of sibling nodes are re-weighted only at the first iteration. Here, we denote diverse beam-search as `DS` and modification of diverse beam- search @1 as `DS`@1. The state-of-the-art length normalization [216] with cumulative negative-log likelihood score is used here. The diverse beam search hyperparameter γ is chosen based on the validation error, the values of γ are presented in Table 4.5. A candidate sequence is generated when the end of sequence symbol is found (a complete hypothesis) or when it reaches the maximum number of iterations. If no complete hypothesis is observed before reaching the maximum number of iterations, the sequences from the root to the leaves of the maximum depth are considered as the candidate predictions. We set the maximum number of iterations to seven since queries longer than that are rarely observed (see Table 4.1). Beam search stops when the maximum number of iterations is reached or when the desired number of candidates is generated. The beam size and desired number of samples are chosen based on the validation error. The (beam, sample) sizes are searched over values of $\{(25, 25), (50, 50)\}$.

In order to select the best candidate sequences we considered three different scores: negative log-likelihood score of candidate query (NLL), average BLEU score between the candidate query and all the previous queries (BLEU-avg) and BLEU score between the candidate and the last query (BLEU-last). We observe that BLEU-avg which priorities next queries similar to all previous queries gives better ranking than

NLL and BLEU-last. Hence, we use BLEU-avg score to rank and select the top 20 candidate queries. These top 20 candidate sequences are used in the second-level ranker.

To assess the effect of using the attention mechanism on the set of interaction words, we compare the proposed `SEQ` method with $\texttt{SEQ}_\texttt{b}$ that corresponds to the model introduced in Sordoni et. al. [178]. $\texttt{SEQ}_\texttt{b}$ is trained over the same training sessions using the same hyperparameter space and the same random seed as `SEQ`. The same technique explained above is used to generate the top candidate sequences from the word scores provided by $\texttt{SEQ}_\texttt{b}$. The only difference is that $\texttt{SEQ}_\texttt{b}$ does not integrate over interaction word features.

4.5.3 Setup for Second-Level Ranker

All `SET` and `SEQ` methods generate candidate sequences, and the top 20 of those sequences are used in the second-level ranker. As explained in Section 4.4.3, each candidate sequence is represented using nine features. Again, both pairwise and listwise learning-to-rank algorithms are used. For the pairwise, all possible pairs are generated on the 20 candidate sequences and an SVM classifier using the LIBLINEAR implementation is learned, the C parameter being fixed by 5-fold cross-validation using the set mentioned in Table 4.5. The resulting models using LIBLINEAR are denoted as `MODEL-LL`, where `MODEL` is the version of the `SET` or `SEQ` used to generate the candidate sequences (*e.g.* `SET-LM-LL` denotes that the pairwise second-level ranker is used on the candidate sequences generated by `SET-LM`). For the listwise approach, we use the LambdaMART implementation of RankLib package where the number of trees is fixed by 5-fold cross validation using the value set mentioned in Table 4.5. Similarly, the resulting models using LambdaMART are denoted as `MODEL-LM`.

4.5.4 Evaluation

We evaluate the different models on the basis of how similar the predicted queries are to the true ones, as well as on the basis of the retrieval performance of the last

predicted query. We use the Jaccard similarity and the BLEU score to compute the similarity between the queries obtained by the different models and the original ones. These scores, reported in Section 4.6, are averaged over 5 folds.

In addition to evaluate the top predicted query, we further investigate the relevance of the top K ranked candidates. In earlier studies (*e.g.*, in [178]), the mean reciprocal rank (mRR) is utilized to asses the list of candidate queries. MRR, *i.e.* the mean inverse rank of the true query among the list of candidate queries, however fails to account for candidate queries close to the true one, yet not identical to it. Indeed, a candidate list containing queries close to the true one is more interesting than a candidate list in which all queries are far away for the true one. We thus propose to modify the standard MRR measure by taking into account, at each rank, the similarity between the candidate and the true queries. This leads to a measure we refer to as *mean Jaccard Reciprocal Rank, mJRR*, given by:

$$\texttt{mJRR} = \frac{1}{|I|} \sum_{i=1}^{|I|} \sum_{r=1}^{K} \frac{\text{Jaccard}(\hat{q}_r^i, q^i)}{r} \tag{4.5}$$

where $|I|$ is the number of interactions and $\text{Jaccard}(\hat{q}_r^i, q^i)$ the Jaccard similarity of the candidate query at rank r ($r \in [1, \ldots, K]$) and the original user query. In our experiments (see Section 4.6), K is set to 20.

Furthermore, to evaluate the effect of the session history length in the query prediction performance, we divided the queries into three categories in terms of their session history length: short, medium, and long. The next query prediction after observing first and second interactions corresponds to *short*, after observing the third and the forth *medium* and after observing at least 5 interactions *long*.

Additionally, the performance of the top-ranked candidate sequences (predicted queries) are compared to that of the original queries in terms of mean average precision (`MAP`), normalized discounted cumulative gain (`nDCG`), and normalized discounted cumulative gain @10 (`nDCG@10`) through the retrieval performed on the ClueWeb12 collection. Indri's default language modeling approach is used for performing retrieval. For calculation of `MAP`, `nDCG`, and `nDCG@10` we rely on the TREC-eval implementa-

tion[11]. Due to the size of the ClueWeb12 collection, performing retrieval is very time consuming, thus for retrieval performance-based evaluation we compared the MAP, nDCG, and nDCG@10 with the final queries of the sessions in the dataset.

4.6 Results

We begin our investigation through word similarity based evaluation in which we assess the quality of the predicted next query based on the original next query, by comparing different SET and SEQ models with the sequence view baseline $\texttt{SEQ}_\texttt{b}$ (studied in [178]) mentioned in Sections 4.5.1 and 4.5.2. We utilize all the sources of information in our experiments, namely words in the past queries, in the titles and snippets of the past retrieved documents, and the words from the retrieved and clicked documents. In this section, we simply use BLEU score to denote sentence-level BLEU score. One should note that higher values of Jaccard similarity, BLEU and mJRR scores signify a better performance. In the tables appearing hereafter, the best performance among all methods is shown in bold and other methods are marked with one asterisk (*) if they are significantly worse than the best method according to a paired t-test with 5% significance level with Bonferroni correction and with two asterisks (**) if they are significantly worse than the best method according to a paired t-test with 1% significance level with Bonferroni correction.

We start our experiments with the comparison of predicted next queries with the original next queries obtained from the data in a search session using Jaccard and BLEU scores. Table 4.6 shows the similarties obtained by different sequence-based models. Here, the proposed sequence model RNN architecture (SEQ) outperforms the baseline RNN architecture ($\texttt{SEQ}_\texttt{b}$). We further investigate multiple beam-search algorithms: standard beam-search (STD), diverse beam-search (DS) and diverse beam-search @1 (DS@1) to generate candidate queries using SEQ models. As one can note from Table 4.6, increasing diversity in the beam-search does not improve over the standard (STD) beam search in terms of the similarity scores between predicted and

[11]https://trec.nist.gov/trec_eval/

Table 4.6: Evaluation of different SEQ models based on the average Jaccard similarity, BLEU score between the predicted queries and original queries. Different beam search algorithms are compared: standard beam search (STD), diverse beam search (DS) and diverse beam search @1 (DS@1). The second score in parenthesis corresponds to standard deviation over 5 folds. The best measures (maximum) are in bold. "*" indicate model is significantly worse than the best method according to a paired t-test using Bonferroni correction with 5% and "**" with 1%.

Model	Jaccard	BLEU
SEQ-STD-LL	**0.288(0.023)**	**0.295(0.015)**
SEQ-STD-LM	0.265(0.019)**	0.29(0.015)**
SEQ-DS@1-LL	0.201(0.011)**	0.202(0.012)**
SEQ-DS@1-LM	0.188(0.01)**	0.201(0.009)**
SEQ-DS-LL	0.076(0.01)**	0.093(0.013)**
SEQ-DS-LM	0.060(0.009)**	0.075(0.01)**
SEQ_b-STD-LL	0.264(0.027)**	0.287(0.02)**
SEQ_b-STD-LM	0.246(0.025)**	0.28(0.018)**
SEQ_b-DS@1-LL	0.214(0.022)**	0.221(0.015)**
SEQ_b-DS@1-LM	0.200(0.021)**	0.218(0.017)*
SEQ_b-DS-LL	0.077(0.02)**	0.094(0.022)**
SEQ_b-DS-LM	0.061(0.019)**	0.076(0.022)**

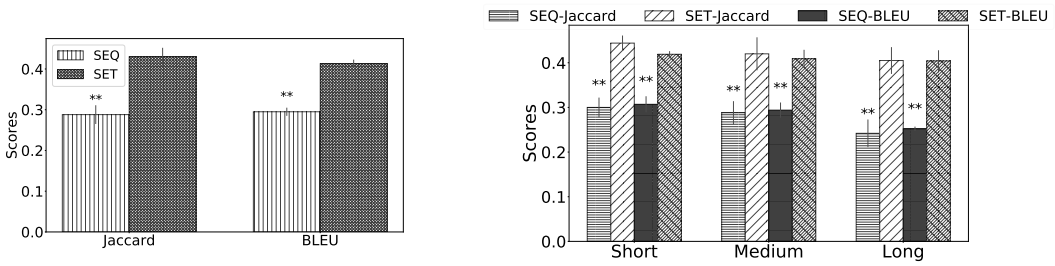
original queries. Moreover, the best performing sequence model is the SEQ model using standard beam search with LL as second-level ranker, SEQ-STD-LL. It leads to the best average Jaccard score, 0.288 (with standard deviation of 0.023 over 5 folds of cross validation), which is significantly better than the rest of the sequence models with 1% significance level with Bonferroni correction. Furthermore, SEQ-STD-LL leads to the best average BLEU score, 0.295 (with standard deviation of 0.015), which is significantly better than the rest of the sequence models with 1% significance level, except for SEQ_b-DS@1-LM. SEQ-STD-LL outperforms SEQ_b-DS@1-LM with 5% significance level. Here, models with LL (LIBLINEAR) as a second-level ranker consistently outperforms LM (LambdaMART). Hence, for further analysis, we retain LL as second level-ranker for sequence view models. In the following it is denoted as SEQ-BS instead of SEQ-BS-LL (BS corresponds to the beam search algorithm used).

Table 4.7: Evaluation of different SET models based on the average Jaccard similarity and BLEU score between the predicted queries and original queries. The second score in parenthesis corresponds to standard deviation over 5 folds. The best measures (maximum) are in bold. " * " indicate model is significantly worse than the best method according to a paired t-test using Bonferroni correction with 5% and " ** " with 1%.

Model	**Jaccard**	**BLEU**
SET-LL-150-LL	0.376(0.016)**	0.395(0.01)**
SET-LL-150-LM	**0.43(0.022)**	0.412(0.012)
SET-LL-100-LL	0.376(0.017)**	0.395(0.011)**
SET-LL-100-LM	0.429(0.022)	**0.413(0.01)**
SET-LL-75-LL	0.377(0.015)**	0.396(0.011)**
SET-LL-75-LM	0.428(0.025)	0.412(0.013)
SET-LL-50-LL	0.377(0.016)**	0.397(0.012)**
SET-LL-50-LM	0.429(0.022)	0.412(0.011)
SET-LM-LL	0.286(0.037)**	0.32(0.036)**
SET-LM-LM	0.315(0.069)**	0.327(0.049)**

Table 4.7 illustrates the Jaccard similarities and BLEU scores using set view models. Here, SET-LL-150-LM leads to the highest Jaccard similarity score 0.43 (with standard deviation of 0.022) which is significantly better than set view models with LM first-level ranker, and with LL second-level ranker with 1% significance level with Bonferroni correction. Moreover, SET-LL-100-LM is the best performing model in terms of BLEU score and significantly outperforms set view models with LM first-level ranker, and with LL second-level ranker with 1% significance level. Here we observe that SET-LL-X-LM models (X corresponds to the different number of pairs) results are comparable in terms of Jaccard similarity and BLEU scores. From the above observations, one can conclude that the set view models with LL as first-level ranker perform better than LM, whereas the set view models with LM as second-level ranker perform better than LL. We thus retain LL as first-level ranker and LM as second-level ranker for further analysis, simplifying our notations to *i.e* SET-X instead of SET-LL-X-LM.

Figure 4.4a depicts the comparison of the best performing SEQ and SET models in terms of Jaccard similarity and BLEU scores. In Figure 4.4a, the best SET model sig-



(a) Overall (b) Different session lengths

Figure 4.4: Comparison of best SET and SEQ models in terms of session lengths. The error bar corresponds to standard deviation over 5 folds. "**" indicate model is significantly worse than the best method according to a paired t-test using Bonferroni correction with 1%.

nificantly outperforms the best SEQ model with 1% significance level with Bonferroni correction in terms of both Jaccard similarity and BLEU scores.

Figure 4.4b illustrates the Jaccard similarity and BLEU score between the original query and predicted query for short, medium, and long session histories. It shows that for both SEQ and SET models the Jaccard similarity and BLEU score slightly drops as the session history increases.

Analysis of Jaccard similarity and BLEU scores between the predicted queries and original queries showed that the proposed sequence view model SEQ and set view model SET outperforms the baseline model $\texttt{SEQ}_\texttt{b}$ in terms of the similarity between predicted and original user query.

We now analyze the top candidate queries in terms of mJRR scores (see Section 4.5.4). Table 4.8 illustrates the mJRR scores for the sequence view models. As we observed in the earlier analysis, here the STD beam search leads to higher mJRR score for both sequence view models SEQ and $\texttt{SEQ}_\texttt{b}$. Furthermore, the best mJRR score, *i.e.*, 0.037 (with standard deviation of 0.002), is obtained by the proposed sequence view model with LL second level ranker using STD beam search. It significantly outperforms the rest of the sequence view models with 1% significance level with Bonferroni correction, except for $\texttt{SEQ}_\texttt{b}$-DS@1. Moreover, Table 4.9 presents the mJRR scores using set view models with LM as the first-level ranker, and with LL as the second-level ranker.

Table 4.8: Evaluation of different SEQ models (with LL second level ranker) based on the mJRR on the top ranked 20 candidate queries. Different beam search algorithms are compared: standard beam search (STD), diverse beam search (DS) and diverse beam search @1 (DS@1). The best measures (maximum) are in bold. "*" indicate model is significantly worse than the best method according to a paired t-test using Bonferroni correction with 5% and "**" with 1%.

Model	mJRR
SEQ-STD	**0.037(0.002)**
SEQ-DS@1	0.021(0.001)**
SEQ-DS	0.002(0.0)**
SEQ_b-STD	0.036(0.003)**
SEQ_b-DS@1	0.022(0.004)
SEQ_b-DS	0.002(0.001)**

As one can note, SET-X models obtain the same mJRR score, which can be explained by the fact that SET-X models lead to very similar Jaccard similarity between the predicted queries and the original queries.

We compare the best sequence and set view models in terms of mJRR score in Figure 4.5. The best SET model significantly outperforms the best SEQ model with 1% significance level with Bonferroni correction in all settings. In terms of different session lengths the SET model achieve similar mJRR score and we observe a slight drop in mJRR score of the SEQ model as the session length increases.

We further analyze the predicted queries in terms of their retrieval performance. For this we calculate the nDCG and nDCG@10 on the final queries of the search sessions. Table 4.10 provides the MAP, nDCG, and nDCG@10 on the ClueWeb12 collection using SET and SEQ models. Here, the best MAP, nDCG, and nDCG@10 scores are obtained by the set view model with LL first-level ranker with 50-pairs and LM second-level ranker, SET-50. The best MAP score, *i.e.*, 0.074 (with standard deviation of 0.009), is significantly better than SEQ_b-DS and SEQ-DS@1 with 5% significance level, and SEQ-DS with 1% significance level. Furthermore, in terms of MAP we do not observe a significant difference than the rest of the models. The resulting best nDCG score, *i.e.*,

Table 4.9: Evaluation of different SET models based on the mJRR of the top 20 ranked candidate queries. The second score in parenthesis corresponds to standard deviation over 5 folds. The best measures (maximum) are in bold."*" indicate model is significantly worse than the best method according to a paired t-test using Bonferroni correction with 5% and "**" with 1%.

Model	mJRR
SET-150	**0.048(0.002)**
SET-100	**0.048(0.002)**
SET-75	**0.048(0.002)**
SET-50	**0.048(0.002)**

0.191 (with standard deviation of 0.014), is significantly better than the best sequence view models with 1% significance level with Bonferroni correction. Here, SET-50 outperforms SEQ-STD with 5% significance level. Furthermore, the best nDCG@10 score (by SET-50), *i.e.*, 0.127 (with standard deviation of 0.015), is significantly better than sequence view models; SEQ-DS@1 and SEQ-DS with 1% significance level and for the rest with 5%. SET-50 is comparable with the rest of the set view models (SET-75, SET-100, SET-150) in terms of MAP, nDCG, and nDCG@10.

Figure 4.6 compares the retrieval performance of original user final queries with the predicted queries obtained by the best sequence and set view models. It shows that the best set view model can achieve as good retrieval performance as the user final queries. However, the sequence view model cannot achieve a comparable retrieval performance with original final queries, which can be explained by the limited dataset size.

4.7 Conclusion

We have studied in this chapter the problem of predicting next queries in search sessions. To do so, we have adopted two different views on queries, a bag-of-words and a sequence view, and introduced several features pertaining to different sources of evidence. The best prediction for next queries relies on the set view approach.

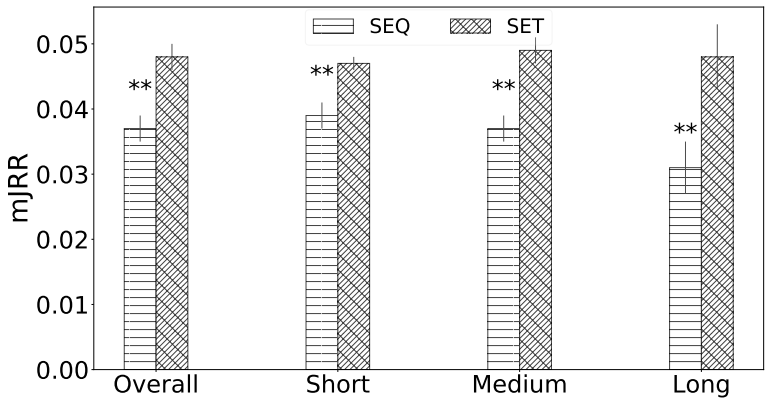


Figure 4.5: Comparison of best `SET` and `SEQ` models with original user queries in terms of `mJRR` performance. The error bars correspond to standard deviation over 5 folds. The next query prediction after observing first and second interactions corresponds to "Short", after observing third and forth interactions "Medium" and after observing at least 5 interactions "Long". "**" indicate model is significantly worse than the best method according to a paired t-test using Bonferroni correction with 1%.

Furthermore, we have shown that the final queries predicted yield retrieval results on a par with the final queries provided by users. The sequence view approach leads to a natural formulation of the problem; it however requires a large dataset to be trained, which is not usually available in the context of search sessions.

Table 4.10: Retrieval performance based evaluation of predicted final queries using different SET and SEQ models against the original final queries in terms of MAP, nDCG, and nDCG@10 of 5 fold cross-validation results on ClueWeb12 collection. The best measures (maximum) are in bold. "*" indicate model is significantly worse than the best method according to a paired t-test using Bonferroni correction with 5% and "**" with 1%.

Model	MAP	nDCG	nDCG@10
SET-150	0.072(0.012)	0.189(0.017)	0.124(0.02)
SET-100	0.073(0.011)	0.189(0.016)	0.126(0.019)
SET-75	0.071(0.011)	0.187(0.015)	0.124(0.017)
SET-50	**0.074(0.009)**	**0.191(0.014)**	**0.127(0.015)**
SEQ-STD	0.07(0.01)	0.178(0.014)*	0.112(0.008)*
SEQ-DS@1	0.062(0.008)*	0.164(0.011)**	0.098(0.012)**
SEQ-DS	0.062(0.008)**	0.162(0.008)**	0.099(0.01)**
SEQ_b-STD	0.065(0.01)	0.175(0.013)**	0.113(0.008)*
SEQ_b-DS@1	0.065(0.008)	0.172(0.008)**	0.113(0.011)*
SEQ_b-DS	0.064(0.008)*	0.173(0.008)**	0.114(0.008)*

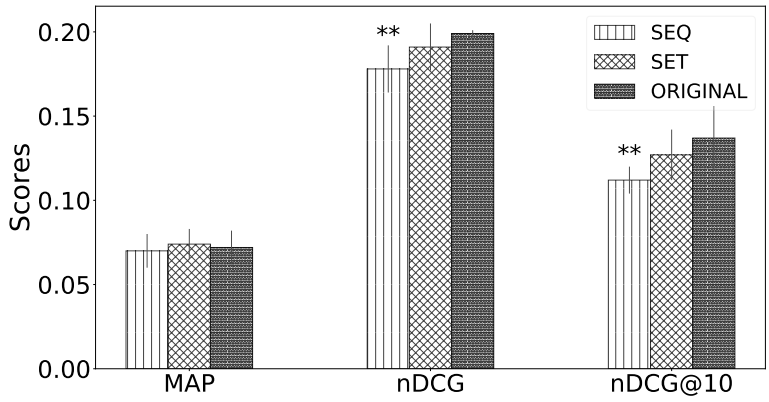


Figure 4.6: Comparison of best SET and SEQ models with original user final queries in terms of session search retrieval performance. The error bars correspond to standard deviation over 5 folds. "**" indicate model is significantly worse than the original query according to a paired t-test using Bonferroni correction with 1%.

Chapter 5

General Conclusions and Perspectives

In this thesis, we have developed sequence prediction approaches using recurrent neural networks. In particular we focus on two sequence prediction tasks: time series forecasting and information retrieval search session.

The following two sections will summarize the main points studied in this thesis and provide some perspectives on this work.

5.1 Time Series Forecasting

RNNs achieved state-of-the-art performance in many sequence learning tasks. Their sequence learning capabilities makes them an interesting approach for time series forecasting. Many time series exhibit periods due to the environmental factors such as seasonality or due to the patterns underlying the activities measured. In a forecasting scenario, periods correspond to the difference between the positions of the output being predicted and specific inputs. Thus, in Chapter 3, we first addressed the following question: *Can sequence-to-sequence RNNs model periods in time series?*

To capture periods, we needed a memory unit to reuse specific representations of input values to predict output values. A particular RNN architecture, sequence-to-sequence with content attention mechanism [8], is well suited to reuse the input

sequence to predict output. This architecture was initially proposed for the machine translation task. Hence, we first investigate the capability of this model to capture periods in time series. Here, the attention weight of an input value signifies how much this specific input value contributes in the output prediction or how much this specific input value aligns with the output element. Therefore, we used attention weights as an indicator of periods. We observed that the standard attention mechanism fails to capture periods. We augmented the input with position information to help capture the periods. Although this led to a more reasonable attention distribution, it was not fit to capture periods. Hence, we proposed to model periods with a vector which encodes the importance of the relative position between the output and input. This extended attention mechanism enabled us to learn how to put high weights for the input values corresponding to the periods with respect to the output position. This enabled us to modify the importance of the input hidden state. In addition to the aforementioned vector, we propose a matrix reweighing scheme which led to a finer control over hidden states.

Furthermore, lost records and/or mistakes during data entry or due to faulty sensors lead to missing values or gaps in time series. Standard approaches to deal with missing values, *e.g.* interpolation and data imputation, make strong assumptions on the structure of the data. In this respect, the following question is addressed in this thesis: *Do sequence-to-sequence RNNs cope well with missing values?*

Inferred missing values or gaps using numerical or deterministic approaches (*e.g.* interpolation or data imputation) are less reliable than other inputs. Especially in gaps, it is increasingly less reliable the further away the inferred value is from the observed values. Hence, we proposed two reweighing schemes as a modification of the attention mechanism. A first one, *a priori* adapted to padding, is based on a decaying function to penalize the missing values with respect to the distance from the last observed value. A second one penalizes the missing values according to where they lie in the gap: at the beginning, middle or end of the gap. This approach *a priori* adapted to interpolation methods which infer the values in the gap based on the values before and after the gap.

In the multivariate time series forecasting case, each variable can have its own period. As the underlying patterns of each time series can differ, the inferred missing values of each variable are not necessarily equally reliable. Thus, we proposed a direct extension to multivariate time series that has a separate encoder and attention mechanism for each temporal variable. Later, we combine the context vectors of each temporal variable by simple concatenation. This allows to consider potential dependencies between different variables to predict the output. We also proposed a second approach that has a single (global) attention model for all variable representations. In both cases, the variable representations were obtained with separate encoders.

The experimental results over several univariate and multivariate time series demonstrate the effectiveness of these extensions. Indeed, these extensions perform significantly better than the original attention model as well as state-of-the-art baseline methods based on ARIMA and random forests. The research directions following our time series forecasting contributions could be as follows:

- To avoid using a validation set that may be not large enough to properly select the best method, a formal criteria to select the best extension for both univariate and multivariate time series could be proposed.

- The proposed approaches provide point forecasts. A future direction is to provide prediction interval which enables to assess future uncertainty [32].

- Here, we adapted a unimodal approach, with the availability of other modalities that is related to the evaluation of a time series, *e.g.* some recent news in politics might affect the evaluation of a stock market. One possible approach would be to consider an attention mechanism over multiple modalities, *e.g.* time series, image and text (similar to [159]).

- The proposed RNN models were trained with a single loss function using back-propagation through time. Some recent studies [181] showed that utilizing a multi-task learning approach might help enrich the hidden representations. The most straight forward multi-task extension of our proposed multivariate models would be considering forecasting loss function of other variables along with

target variables. An example of such case is when one is interested in electricity load forecasting using a multivariate time series forecasting by taking into account the weather. Multivariate temporal variables in this case are electricity load and temperature. In a multi-task learning setting, one can have two loss functions based on electricity load and temperature forecasting performances, as opposed to having only one loss function based on electricity load forecasting performance.

- Our time series models assumes a fixed sampling rate, one future direction is relaxing this assumption and able to predict future values of an irregularly sampled time series. Predicting future condition of a patient is an irregularly sampled time series forecasting task in which irregularly sampled medical records of a patient used.

5.2 Next Query Prediction

The second challenge addressed in this thesis is predicting next queries in an IR search session. A relatively complex information need requires users to formulate multiple queries to cover all the aspects of their search. The next query prediction is a first step towards building systems that are able to simulate IR search sessions. With such systems, one could possibly expand the current IR datasets based on single queries to sessions, which leads to higher retrieval performance than single-query-based retrieval [30].

A search session inherits a dynamic nature, in which the information need of a user might change over a search session as a consequence of her interactions with the search engine. Furthermore, these interactions help a user to determine what to search for, *e.g.*, an information need on a specific topic requires specific terminology.

To assess the influence of search session interactions on the user's query reformulations, we conducted the following analysis: (i) the general statistics of search sessions (*e.g.* how the query length evolves through a session), (ii) the evolution of the topic through a search session, (iii) the sources of next query words, *i.e. where*

the next query words come from. Some of the findings of these analysis can be listed as follows:

- Most sessions are composed of four queries, few sessions have more than six queries;
- The number of terms (words) in a query is usually less than seven and most queries are of length two to four;
- Queries composed of one word are likely to evolve to multi-word queries. However, queries consisting of more than four words are more likely to be reformulated into shorter queries;
- Consecutive queries are rather cohesive and the topic of two consecutive queries does not change drastically;
- Query words are present in the previous interactions (87.83% of the query words);

The goal of this part was to predict next query, at any point in a session, given the previous interactions, namely previous queries, retrieved documents, snippets of retrieved documents, and clicked documents in this session. To do so, we proposed two methods to predict next queries in sessions: the set view and the sequence view approaches.

The set view approach consider the query words as a bag of words. In this case, the probability of a word to be in the next query only depends on the previous interactions. This probability was estimated by using a learning to rank approach, which results in a ranked list of words with corresponding scores. To efficiently compose candidate (next) queries, we built a tree of words.

The sequence view approach considers that a query is composed of a sequence of words and sequence of previous interactions. Here, we proposed an extension of an hierarchical encoder-decoder RNN approach of query prediction by taking into account the sources of the query words. Here, to incorporate the sources of the next

query words, we augmented an attention mechanism on top of the previous interaction words. Although the sequence view approach leads to a natural formulation of the problem, it usually require a large dataset to be trained, which is not usually available in the context of search sessions. The best prediction for next queries relies on the set view approach. Furthermore, we have shown that the final queries predicted yield retrieval results on a par with the final queries provided by users.

The research directions following our IR search session next query prediction contributions could be listed as follows:

- For the next query prediction task, we proposed two approaches: the set and the sequence views. One straightforward direction would be a model combining the two, *e.g.* using a weighted score to generate candidate queries, or make use of a consensus score to rank the candidate queries.

- In the TREC Session Search dataset, identifiers of the users are not present. With the availability of a future search session dataset including identifiers, one can extend the hierarchical network to encompass the user profiles to develop a personalized query prediction system.

- Here, to train the hierarchical encoder-decoder RNN, we used the cross-entropy loss function. Although the cross-entropy provides a fairly good training of the model, other more task-specific losses might be interesting to investigate, *e.g.* rank-based loss [89].

- Our extended hierarchical encoder-decoder RNN model consist of an attention mechanism between the previous query word and interaction words. Here, the attention calculation is built on the additive similarity evaluation of the attention input. Other attention calculation approaches could be adapted, *e.g.* multiplicative attention calculation.

Bibliography

[1] Eiffel tower view : the best places to admire the iron lady. `https://www.pariscityvision.com/en/paris/landmarks/eiffel-tower/best-viewpoints-eiffel-tower`. Accessed: August-2018.

[2] Wasi Ahmad, Kai-Wei Chang, and Hongning Wang. Multi-task learning for document ranking and query suggestion. In *ICLR*, 2018.

[3] Eitan Altman. *Applications of Markov Decision Processes in Communication Networks*, pages 489–536. Springer US, Boston, MA, 2002.

[4] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *CoRR*, abs/1412.7755, 2014.

[5] Ricardo Baeza-Yates. Graphs from search engine queries. In Jan van Leeuwen, Giuseppe F. Italiano, Wiebe van der Hoek, Christoph Meinel, Harald Sack, and František Plášil, editors, *SOFSEM 2007: Theory and Practice of Computer Science*, pages 1–8, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[6] Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *Proceedings of the 2004 International Conference on Current Trends in Database Technology*, EDBT'04, pages 588–596, Berlin, Heidelberg, 2004. Springer-Verlag.

[7] Augusto Cesar Espíndola Baffa and Angelo E. M. Ciarlini. Modeling pomdps for generating and simulating stock investment policies. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2394–2399, New York, NY, USA, 2010. ACM.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv e-prints*, abs/1409.0473, September 2014.

[9] P. Baldi, Y. Chauvin, T. Hunkapiller, and M. A. McClure. Hidden Markov models of biological primary sequence information. *Proc. Natl. Acad. Sci. U.S.A.*, 91(3):1059–1063, Feb 1994.

[10] Pierre Baldi and Søren Brunak. *Bioinformatics: The Machine Learning Approach*. MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[11] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.

[12] William R. Bell and Steven C. Hillmer. Issues involved with the seasonal adjustment of economic time series: Reply. *Journal of Business & Economic Statistics*, 2(4):343–349, 1984.

[13] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[14] Y Bengio and Paolo Frasconi. An input output hmm architecture. 7:427–434, 12 1995.

[15] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[16] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, Aristides Gionis, and Sebastiano Vigna. The query-flow graph: Model and applications. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 609–618, 2008.

[17] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, and Sebastiano Vigna. Query suggestions using query-flow graphs. In *Proceedings of the 2009*

Workshop on Web Search Click Data, WSCD '09, pages 56–63, New York, NY, USA, 2009. ACM.

[18] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307 – 327, 1986.

[19] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. Machine learning strategies for time series forecasting. In *Business Intelligence*, pages 62–77. Springer, 2013.

[20] Ilaria Bordino, Gianmarco De Francisci Morales, Ingmar Weber, and Francesco Bonchi. From machu_picchu to "rafting the urubamba river": Anticipating information needs via the entity-query graph. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 275–284, New York, NY, USA, 2013. ACM.

[21] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Inc., San Francisco, CA, USA, 1990.

[22] George EP Box and Gwilym M Jenkins. Some recent advances in forecasting and control. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 17(2):91–109, 1968.

[23] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[24] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451. Association for Computational Linguistics, 2017.

[25] Daniele Broccolo, Lorenzo Marcon, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. Generating suggestions for queries in the long tail with an inverted index. *Information Processing & Management*, 48(2):326–339, 2012.

[26] Peter J Brockwell and Richard A Davis. *Time Series: Theory and Methods*. Springer-Verlag, Berlin, Heidelberg, 1986.

[27] Chris J.C. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical report, June 2010.

[28] Nicole Buerle and Ulrich Rieder. *Markov Decision Processes with Applications to Finance*. Springer-Verlag Berlin Heidelberg, 1st edition, 01 2011.

[29] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking svm to document retrieval. In *Proceedings of the 29th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 186–193, 2006.

[30] Ben Carterette, Paul Clough, Mark Hall, Evangelos Kanoulas, and Mark Sanderson. Evaluating retrieval over sessions: The trec session track 2011-2014. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 685–688, 2016.

[31] Ben Carterette, Evangelos Kanoulas, Mark M. Hall, and Paul D. Clough. Overview of the TREC 2014 session track. In *Proceedings of The 23rd Text REtrieval Conference, TREC*, 2014.

[32] Chris Chatfield. *Prediction Intervals for Time-Series Forecasting*, pages 475–494. Springer US, Boston, MA, 2001.

[33] Chris Chatfield. *The analysis of time series: an introduction*. CRC press, 2016.

[34] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *CoRR*, abs/1606.01865, 2016.

[35] Haibin Cheng, Pang-Ning Tan, Jing Gao, and Jerry Scripps. Multistep-ahead time series prediction. In Wee-Keong Ng, Masaru Kitsuregawa, Jianzhong Li, and Kuiyu Chang, editors, *Advances in Knowledge Discovery and Data Mining*, pages 765–774, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[36] Kyunghyun Cho, Bart van Merriënboer, Çalar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[37] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F. Stewart, and Jimeng Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In Finale Doshi-Velez, Jim Fackler, David Kale, Byron Wallace, and Jenna Wiens, editors, *Proceedings of the 1st Machine Learning for Healthcare Conference*, volume 56 of *Proceedings of Machine Learning Research*, pages 301–318, Children's Hospital LA, Los Angeles, CA, USA, 18–19 Aug 2016. PMLR.

[38] Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. RETAIN: An interpretable predictive model for healthcare using reverse time attention mechanism. In *NIPS*, pages 3504–3512, 2016.

[39] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 577–585. Curran Associates, Inc., 2015.

[40] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[41] Robert B. Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition procedure based on loess (with discussion). *Journal of Official Statistics*, 6:3–73, 1990.

[42] Jerome Connor, Les E Atlas, and Douglas R Martin. Recurrent networks and NARMA modeling. In *NIPS*, pages 301–308, 1991.

[43] Paul S. P. Cowpertwait and Andrew V. Metcalfe. *Introductory Time Series with R*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[44] Germán Creamer and Yoav Freund. Predicting performance and quantifying corporate governance risk for latin american adrs and banks. In *Proceedings of the Second IASTED International Conference on Financial Engineering and Applications*, 11 2004.

[45] Sven F Crone, Michele Hibon, and Konstantinos Nikolopoulos. Advances in forecasting with neural networks? empirical evidence from the nn3 competition on time series prediction. *International Journal of Forecasting*, 27(3):635–660, 2011.

[46] Mostafa Dehghani, Sascha Rothe, Enrique Alfonseca, and Pascal Fleury. Learning to attend, copy, and generate for session-based query suggestion. In *Proceedings of the ACM on Conference on Information and Knowledge Management, CIKM*, pages 1747–1756, 2017.

[47] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.

[48] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[49] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179 – 211, 1990.

[50] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[51] Robert F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982.

[52] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[53] Yugang Fan, Ping Li, and Zhihuan Song. Dynamic least squares support vector machine. In *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, volume 1, pages 4886–4889. IEEE, 2006.

[54] Mikel L. Forcada and Ramón P. Ñeco. Recursive hetero-associative memories for translation. In *Proceedings of the International Work-Conference on Artificial and Natural Neural Networks: Biological and Artificial Computation: From Neuroscience to Technology*, IWANN '97, pages 453–462, Berlin, Heidelberg, 1997. Springer-Verlag.

[55] P.H. Franses and R. Paap. *Periodic Time Series Models*. Advanced texts in econometrics. Oxford University Press, 2004.

[56] Philip Hans Franses and Richard Paap. Forecasting with periodic autoregressive time series models. Econometric Institute Research Papers EI 9927-/A, Erasmus University Rotterdam, Erasmus School of Economics (ESE), Econometric Institute, 1999.

[57] Joseph Futoma, Sanjay Hariharan, Katherine A. Heller, Mark Sendak, Nathan Brajer, Meredith Clement, Armando Bedoya, and Cara O'Brien. An improved multi-output gaussian process RNN with real-time validation for early sepsis detection. In Finale Doshi-Velez, Jim Fackler, David C. Kale, Rajesh Ranganath, Byron C. Wallace, and Jenna Wiens, editors, *Proceedings of the Machine Learning for Health Care, MLHC 2017, Boston, Massachusetts, USA, 18-19 August 2017*, volume 68 of *Proceedings of Machine Learning Research*, pages 243–254. PMLR, 2017.

[58] Susan Gauch and John B Smith. An expert system for automatic query reformation. *Journal of the American Society for Information Science*, 44(3):124, 1993.

[59] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. Convolutional sequence to sequence learning. In *ICML*, 2017.

[60] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying LSTM to time series predictable through time-window approaches. In *International Conference on Artificial Neural Networks*, pages 669–676. Springer, 2001.

[61] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

[62] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, October 2000.

[63] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with LSTM recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

[64] Tony Van Gestel, Johan A. K. Suykens, Dirk-Emma Baestaens, Annemie Lambrechts, Gert R. G. Lanckriet, Bruno Vandaele, Bart De Moor, and Joos Vandewalle. Financial time series prediction using least squares support vector machines within the evidence framework. *IEEE Trans. Neural Networks*, 12(4):809–821, 2001.

[65] Zoubin Ghahramani. Hidden markov models. chapter An Introduction to Hidden Markov Models and Bayesian Networks, pages 9–42. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2002.

[66] Marzyeh Ghassemi, Marco AF Pimentel, Tristan Naumann, Thomas Brennan, David A Clifton, Peter Szolovits, and Mengling Feng. A multivariate timeseries modeling approach to severity of illness assessment and forecasting in icu with sparse, heterogeneous clinical data. In *Proceedings of the... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, volume 2015, page 446. NIH Public Access, 2015.

[67] C Lee Giles, Steve Lawrence, and Ah Chung Tsoi. Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine learning*, 44(1-2):161–183, 2001.

[68] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Capacity management and demand prediction for next generation data centers. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 43–50, July 2007.

[69] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[70] Jan G. De Gooijer and Rob J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443 – 473, 2006. Twenty five years of forecasting.

[71] C. W. J. Granger and Roselyne Joyeux. An introduction to long-memory time series models and fractional differencing. *Journal of Time Series Analysis*, 1(1):15–29, 1980.

[72] A. Graves, M. Liwicki, S. Fernndez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, May 2009.

[73] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4, July 2005.

[74] Alex Graves. *Supervised sequence labelling with recurrent neural networks*. PhD thesis, Technical University Munich, 2008.

[75] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[76] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[77] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwinska, Sergio Gomez 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. *Nature*, 538(7626):471–476, 2016.

[78] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[79] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.

[80] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. DRAW: A recurrent neural network for image generation. *CoRR*, abs/1502.04623, 2015.

[81] D. M. Grether and M. Nerlove. Some properties of "optimal" seasonal adjustment. *Econometrica*, 38(5):682–703, 1970.

[82] Dongyi Guan, Hui Yang, and Nazli Goharian. Effective structured query formulation for session search. Technical report, DTIC Document, 2012.

[83] Dongyi Guan, Sicong Zhang, and Hui Yang. Utilizing query change for session search. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 453–462, 2013.

[84] Christophe Van Gysel, Evangelos Kanoulas, and Maarten de Rijke. Lexical query modeling in session search. *CoRR*, abs/1608.06656, 2016.

[85] Barbara Hammer. On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1):107 – 123, 2000.

[86] James W. Hansen. Realizing the potential benefits of climate prediction to agriculture: issues, approaches, challenges. *Agricultural Systems*, 74(3):309 – 330, 2002.

[87] M. R. Hassan and B. Nath. Stock market forecasting using hidden markov model: a new approach. In *5th International Conference on Intelligent Systems Design and Applications (ISDA'05)*, pages 192–196, Sept 2005.

[88] Qi He, Daxin Jiang, Zhen Liao, Steven C. H. Hoi, Kuiyu Chang, Ee-Peng Lim, and Hang Li. Web query recommendation via sequential query prediction. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 1443–1454, 2009.

[89] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939, 2015.

[90] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

[91] Sepp Hochreiter, Yoshua Bengio, and Paolo Frasconi. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In J. Kolen and S. Kremer, editors, *Field Guide to Dynamical Recurrent Networks*. IEEE Press, 2001.

[92] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[93] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[94] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[95] Tsung-Jung Hsieh, Hsiao-Fen Hsiao, and Wei-Chang Yeh. Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm. *Applied soft computing*, 11(2):2510–2525, 2011.

[96] Michael Jen-Chao Hu. *Application of the Adaline System to Weather Forecasting*. Department of Electrical Engineering, Stanford University., 1964.

[97] Chien-Kang Huang, Lee-Feng Chien, and Yen-Jen Oyang. Relevant term suggestion in interactive web search based on contextual information in query session logs. *J. Am. Soc. Inf. Sci. Technol.*, 54(7):638–649, May 2003.

[98] Rob Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software, Articles*, 27(3):1–22, 2008.

[99] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[100] Bernard J. Jansen, Danielle L. Booth, and Amanda Spink. Patterns of query reformulation during web searching. *Journal of the American Society for Information Science and Technology*, 60(7):1358–1371, 2009.

[101] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.

[102] Michael I. Jordan. Serial order: A parallel distributed processing approach. Technical Report 8604, Institute for Cognitive Science, University of California, 1986.

[103] Michael I. Jordan. Artificial neural networks. chapter Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, pages 112–127. IEEE Press, Piscataway, NJ, USA, 1990.

[104] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2342–2350. JMLR.org, 2015.

[105] B. H. Juang and L. R. Rabiner. Hidden markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991.

[106] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, May 1998.

[107] Michael J Kane, Natalie Price, Matthew Scotch, and Peter Rabinowitz. Comparison of arima and random forest time series models for prediction of avian influenza h5n1 outbreaks. *BMC bioinformatics*, 15(1):276, 2014.

[108] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks, May 2015.

[109] Andrej Karpathy and Fei-Fei Li. Deep visual-semantic alignments for generating image descriptions. *CoRR*, abs/1412.2306, 2014.

[110] Han-Gyu Kim, Gil-Jin Jang, Ho-Jin Choi, Minho Kim, Young-Won Kim, and Jaehun Choi. Recurrent neural networks with missing information imputation for medical examination data prediction. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 317–323, Feb 2017.

[111] Sung Kwon Kim. Optimal algorithms for finding the longest path with length and sum constraints in a tree. *IEICE TRANSACTIONS on Information and Systems*, 94(6):1325–1328, 2011.

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

[113] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[114] Khamsum Kinley, Dian Tjondronegoro, Helen Partridge, and Sylvia Edwards. Human-computer interaction: The impact of users' cognitive styles on query reformulation behaviour during web searching. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pages 299–307, 2012.

[115] Jan Koutník, Klaus Greff, Faustino J. Gomez, and Jürgen Schmidhuber. A clockwork RNN. *CoRR*, abs/1402.3511, 2014.

[116] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology. Applications to protein modeling. *J. Mol. Biol.*, 235(5):1501–1531, Feb 1994.

[117] Bjoern Krollner, Bruce J. Vanstone, and Gavin R. Finnie. Financial time series forecasting with machine learning techniques: a survey. In *ESANN*, 2010.

[118] Andrew Kusiak, Anoop Verma, and Xiupeng Wei. A data-mining approach to predict influent quality. *Environmental monitoring and assessment*, 185(3):2197–2210, 2013.

[119] Dominique Ladiray and Benoît Quenneville. *Outline of the X-11 Method*, pages 13–22. Springer New York, New York, NY, 2001.

[120] A Laepes and R Farben. Nonlinear signal processing using neural networks: prediction and system modelling. Technical report, Technical report, Los Alamos National Laboratory, Los Alamos, NM, 1987.

[121] K. J. Lang and G. E. Hinton. The development of the time-delay neural network architecture for speech recognition. *Tech. report CMU-CS-88-152, Carnegie-Mellon University, Pittsburgh, PA*, 1988.

[122] Kevin J. Lang, Alex H. Waibel, and Geoffrey E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3(1):23 – 43, 1990.

[123] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[124] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1243–1251. Curran Associates, Inc., 2010.

[125] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861 – 867, 1993.

[126] Jiwei Li, Will Monroe, and Dan Jurafsky. A simple, fast diverse decoding algorithm for neural generation. *CoRR*, abs/1611.08562, 2016.

[127] Lei Li, Chieh-Jan Mike Liang, Jie Liu, Suman Nath, Andreas Terzis, and Christos Faloutsos. Thermocast: A cyber-physical forecasting model for datacenters. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1370–1378, New York, NY, USA, 2011. ACM.

[128] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130, 2017.

[129] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with LSTM recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

[130] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[131] Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang. Learning natural language inference using bidirectional LSTM model and inner-attention. *CoRR*, abs/1605.09090, 2016.

[132] Yiqun Liu, Junwei Miao, Min Zhang, Shaoping Ma, and Liyun Ru. How do users describe their information need: Query recommendation based on snippet click model. *Expert Systems with Applications*, 38(11):13847–13856, 2011.

[133] Daniel P Loucks, Eelco Van Beek, Jery R Stedinger, Jozef PM Dijkman, and Monique T Villars. *Water resources systems planning and management: an introduction to methods, models and applications*. Paris: Unesco, 2005.

[134] Jiyun Luo, Xuchu Dong, and Hui Yang. Session search by direct policy learning. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, pages 261–270, 2015.

[135] Jiyun Luo, Sicong Zhang, and Hui Yang. Win-win search: Dual-agent stochastic game in session search. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 587–596, 2014.

[136] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[137] David M Kreindler and Charles J Lumsden. Effects of the irregular sample and missing data in time series analysis. 10:187–214, 05 2006.

[138] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[139] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[140] Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 469–478, New York, NY, USA, 2008. ACM.

[141] Marina Meila and Michael I. Jordan. Learning fine motion by markov mixtures of experts. In *Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, USA, November 27-30, 1995*, pages 1003–1009. MIT Press, 1995.

[142] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[143] Volodymyr Mnih, Nicolas Heess, Alex Graves, and koray kavukcuoglu. Recurrent models of visual attention. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2204–2212. Curran Associates, Inc., 2014.

[144] A. Mohamed and G. Hinton. Phone recognition using restricted boltzmann machines. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4354–4357, March 2010.

[145] Debashis Mondal and Donald B. Percival. Wavelet variance analysis for gappy time series. *Annals of the Institute of Statistical Mathematics*, 62(5):943–966, Oct 2010.

[146] A. Morales-Esteban, F. Martnez-lvarez, A. Troncoso, J.L. Justo, and C. Rubio-Escudero. Pattern recognition to forecast seismic time series. *Expert Systems with Applications*, 37(12):8333 – 8342, 2010.

[147] K-R Müller, Alexander J Smola, Gunnar Rätsch, Bernhard Schölkopf, Jens Kohlmorgen, and Vladimir Vapnik. Predicting time series with support vector machines. In *International Conference on Artificial Neural Networks*, pages 999–1004. Springer, 1997.

[148] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3882–3890. Curran Associates, Inc., 2016.

[149] Kezban Dilek Onal, Ye Zhang, Ismail Sengor Altingovde, Md Mustafizur Rahman, Pinar Karagoz, Alex Braylan, Brandon Dang, Heng-Lu Chang, Henna Kim, Quinten McNamara, Aaron Angert, Edward Banner, Vivek Khetan, Tyler McDonnell, An Thanh Nguyen, Dan Xu, Byron C. Wallace, Maarten de Rijke, and Matthew Lease. Neural information retrieval: at the end of the early years. *Information Retrieval Journal*, 21(2):111–182, Jun 2018.

[150] Umut Ozertem, Olivier Chapelle, Pinar Donmez, and Emre Velipasaoglu. Learning to suggest: A machine learning framework for ranking query suggestions. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 25–34, New York, NY, USA, 2012. ACM.

[151] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[152] Qiao Qian, Minlie Huang, Jinhao Lei, and Xiaoyan Zhu. Linguistically regularized lstm for sentiment classification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1679–1689. Association for Computational Linguistics, 2017.

[153] L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, Jan 1986.

[154] Filip Radlinski and Thorsten Joachims. Query chains: Learning to rank from implicit feedback. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 239–248, 2005.

[155] T Raicharoen, C Lursinsap, and P Sanguanbhoki. Application of critical support vector machine to time series prediction. circuits and systems. iscas03. In *Proceedings of the 2003 International Symposium on*, volume 5, pages 25–28, 2003.

[156] MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.

[157] K. Rehfeld, N. Marwan, J. Heitzig, and J. Kurths. Comparison of correlation analysis techniques for irregularly sampled time series. *Nonlinear Processes in Geophysics*, 18(3):389–404, 2011.

[158] Soo Young Rieh et al. Analysis of multiple query reformulations on the web: The interactive information retrieval context. *Information Processing & Management*, 42(3):751–768, 2006.

[159] Matthew Riemer, Aditya Vempaty, Flavio P Calmon, Fenno F Heath III, Richard Hull, and Elham Khabiri. Correcting forecasts with multifactor neural attention. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 3010–3019, 2016.

[160] Enders A. Robinson. Predictive decomposition of time series with application to seismic exploration. *GEOPHYSICS*, 32(3):418–484, 1967.

[161] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[162] F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962.

[163] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986.

[164] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[165] Rodrygo L. T. Santos, Craig Macdonald, and Iadh Ounis. Learning to rank query suggestions for adhoc and diversity search. *Information Retrieval*, 16(4):429–451, 2013.

[166] Nicholas I. Sapankevych and Ravi Sankar. Time series prediction using support vector machines: A survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, 2009.

[167] Joseph L. Schafer and John W. Graham. Missing data: Our view of the state of the art. *Psychological Methods*, 7(2):147–177, 2002.

[168] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997.

[169] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[170] Ramesh Sharda and R Patil. Neural networks as forecasting experts: an empirical test. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 491–494. IEEE, 1990.

[171] Hava Siegelmann. Computation beyond the turing limit. 268:545–8, 05 1995.

[172] Hava T. Siegelmann and Eduardo D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77 – 80, 1991.

[173] H.T. Siegelmann and E.D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132 – 150, 1995.

[174] Fabrizio Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 4(1-2):1–174, 2010.

[175] Christopher A. Sims. Seasonality in regression. *Journal of the American Statistical Association*, 69(347):618–626, 1974.

[176] Marc Sloan, Hui Yang, and Jun Wang. A term-based methodology for query reformulation understanding. *Information Retrieval Journal*, 18(2):145–165, 2015.

[177] Eugen Slutzky. The summation of random causes as the source of cyclic processes. *Econometrica: Journal of the Econometric Society*, pages 105–146, 1937.

[178] Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 553–562. ACM, 2015.

[179] Shaler Stidham and Richard Weber. A survey of markov decision models for control of networks of queues. *Queueing Systems*, 13(1):291–314, Mar 1993.

[180] R. Stratonovich. Conditional markov processes. *Theory of Probability & Its Applications*, 5(2):156–178, 1960.

[181] Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J Pal. Learning general purpose distributed sentence representations via large scale multi-task learning. In *International Conference on Learning Representations*, 2018.

[182] R. Sun and C. L. Giles. Sequence learning: from recognition and prediction to sequential decision making. *IEEE Intelligent Systems*, 16(4):67–70, July 2001.

[183] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D.

Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[184] Richard S. Sutton. On the significance of markov decision processes. In *ICANN*, volume 1327 of *Lecture Notes in Computer Science*, pages 273–282. Springer, 1997.

[185] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[186] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[187] Idan Szpektor, Aristides Gionis, and Yoelle Maarek. Improving recommendation for long-tail queries via templates. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 47–56, New York, NY, USA, 2011. ACM.

[188] Marina Theodosiou. Forecasting monthly and quarterly time series using stl decomposition. *International Journal of Forecasting*, 27(4):1178 – 1195, 2011.

[189] George C Tiao and George EP Box. Modeling multiple time series with applications. *Journal of the American Statistical Association*, 76(376):802–816, 1981.

[190] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[191] K. Tokuda, Y. Nankaku, T. Toda, H. Zen, J. Yamagishi, and K. Oura. Speech synthesis based on hidden markov models. *Proceedings of the IEEE*, 101(5):1234–1252, May 2013.

[192] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, volume 3, pages 1315–1318 vol.3, June 2000.

[193] S. E. Tsai, Y. S. Chen, C. Y. Tsai, and S. W. Tu. Improving query suggestion by utilizing user intent. In *2010 IEEE International Conference on Information Reuse Integration*, pages 25–30, Aug 2010.

[194] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[195] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[196] A. V. Vecchia. Periodic autoregressive-moving average (parma) modeling with applications to water resources1. *JAWRA Journal of the American Water Resources Association*, 21(5):721–730, 1985.

[197] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *CoRR*, abs/1511.06391, 2015.

[198] Oriol Vinyals, Charles Blundell, Tim Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3630–3638. Curran Associates, Inc., 2016.

[199] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015.

[200] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164, 2015.

[201] Dianne Waddell and Amrik S. Sohal. Forecasting: The key to managerial decision making. *Management Decision*, 32(1):41–49, 1994.

[202] Gilbert Walker. On periodicity in series of related terms. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 131(818):518–532, 1931.

[203] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 515–524, New York, NY, USA, 2017. ACM.

[204] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[205] Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339 – 356, 1988.

[206] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards AI-Complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015.

[207] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014.

[208] D. J. White. A survey of applications of markov decision processes. *The Journal of the Operational Research Society*, 44(11):1073–1096, 1993.

[209] D. J. White and J. M. Norman. Control of cash reserves. *Journal of the Operational Research Society*, 16(3):309–328, Sep 1965.

[210] I. R. White, P. Royston, and A. M. Wood. Multiple imputation using chained equations: Issues and guidance for practice. *Stat Med*, 30(4):377–399, Feb 2011.

[211] P. Whitle. *Hypothesis Testing in Time Series Analysis*. Statistics. Almqvist & Wiksells, 1951.

[212] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, May 1992.

[213] Ronald J. Williams and David Zipser. Backpropagation. chapter Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity, pages 433–486. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.

[214] Bin Wu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. Query suggestion with feedback memory network. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 1563–1571, Republic and Canton of Geneva, Switzerland, 2018. International World Wide Web Conferences Steering Committee.

[215] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[216] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason

Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[217] SHI Xingjian, 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*, pages 802–810, 2015.

[218] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, volume 14, pages 77–81, 2015.

[219] Sidney Yakowitz. Dynamic programming applications in water resources. *Water Resources Research*, 18(4):673–696, 1982.

[220] Grace Hui Yang, Xuchu Dong, Jiyun Luo, and Sicong Zhang. Session search modeling by partially observable markov decision process. *Inf. Retr.*, 21(1):56–80, February 2018.

[221] Hui Yang, Dongyi Guan, and Sicong Zhang. The query change model: Modeling session search as a markov decision process. *ACM Trans. Inf. Syst.*, 33(4):20:1–20:33, May 2015.

[222] Zichao Yang, Zhiting Hu, Yuntian Deng, Chris Dyer, and Alexander J. Smola. Neural machine translation with recurrent attention modeling. *CoRR*, abs/1607.05108, 2016.

[223] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated LSTM. *CoRR*, abs/1508.03790, 2015.

[224] Li Yao, Atousa Torabi, Kyunghyun Cho, Nicolas Ballas, Christopher Pal, Hugo Larochelle, and Aaron Courville. Describing videos by exploiting temporal structure. In *Computer Vision (ICCV), 2015 IEEE International Conference on*. IEEE, 2015.

[225] Takayoshi Yoshimura, Keiichi Tokuda, Takashi Masuko, Takao Kobayashi, and Tadashi Kitamura. Simultaneous modeling of spectrum, pitch and duration in hmm-based speech synthesis. In *Sixth European Conference on Speech Communication and Technology*, 1999.

[226] G Udny Yule. On a method of investigating periodicities in disturbed series, with special reference to wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 226:267–298, 1927.

[227] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[228] Guoqiang Peter Zhang. An investigation of neural networks for linear time-series forecasting. *Computers & Operations Research*, 28(12):1183–1202, 2001.

[229] Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 35–45. Association for Computational Linguistics, 2017.

[230] Zhiyong Zhang and Olfa Nasraoui. Mining search engine query logs for query recommendation. In *Proceedings of the 15th International Conference on World Wide Web*, pages 1039–1040, 2006.

[231] Yin Zheng, Richard S. Zemel, Yu-Jin Zhang, and Hugo Larochelle. A neural autoregressive approach to attention-based recognition. *Int. J. Comput. Vision*, 113(1):67–79, May 2015.