Methods for optimizing customer prospecting in automated display advertising with Real-Time Bidding

Yang Qiu

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAX075

Thèse de doctorat

ÉCOLE POLYTECHNIQUE

IP PARIS

Methods for optimizing customer prospecting in automated display advertising with Real-Time Bidding

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École Polytechnique

École doctorale n°626 : l'École Doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 13/10/2021, par

YANG QIU

Composition du Jury :

Jean-Marc STEYAERT Professor, École Polytechnique (LIX)	Président
Nikos PARAGIOS Professor, Paris-Saclay University (CentraleSupélec, Department of Mathematics)	Rapporteur
Dimitrios GUNOPULOS Professor, National and Kapodistrian University of Athens (Department of Informatics and Telecommunications)	Rapporteur
Oana BALALAU Researcher (ISFP), Inria (Inria Saclay)	Examinatrice
Themis PALPANAS Professor, University of Paris (LIPADE)	Examinateur
Ioannis TSAMARDINOS Professor, University of Crete (Department of Computer Science)	Examinateur
Michalis VAZIRGIANNIS Professor, École Polytechnique (LIX)	Directeur de thèse
Nikolaos TZIORTZIOTIS Senior Data Scientist, Jellyfish (R&D Department)	Co-encadrant de thèse
Martial HUE Senior R&D Manager, Jellyfish (R&D Department)	Invité

ABSTRACT

Online display advertising has become more and more popular in recent years thanks to the automation of the ad buying process. Specifically, Real-Time Bidding (RTB) allows the automated trading of ad impressions between advertisers and publishers through real-time auctions, at per-user level. The primary goal of RTB campaigns is to help advertisers target the right person, in the right context, with the right ad, and at the right time. Therefore, the accurate identification of the 'value' of a user for the advertiser is of high importance. Under this context, we examine two challenging display advertising problems: the conversion prediction and the audience expansion. In both tasks, we consider only the browsing history of the user as features, collected from real logged data.

In the first part of this dissertation, we examine the conversion prediction problem, where our objective is to predict whether a user will convert (visit website, buy product, etc.) or not to a given advertiser. Inspired by natural language processing, we introduce three self-supervised URL embedding models in order to compute semantically meaningful URL representations. Then, we have examined three different mapping functions to represent users that take as input the already learned URL representations. The first one returns the average of the URLs embedding vectors presented on the user's browsing history. The second one considers also the dependencies among the features of the embedding vector returned by the average mapping function. The third one uses the well-known Long Short Term Memory network (LSTM) that is suitable to process variable-length sequences. The main advantage of the third proposed user representation function is consideration of the chronological order in which the URLs appeared in the sequence. Finally, having computed users' representations, we are using the standard logistic regression model to predict conversion probabilities. To demonstrate the effectiveness of the different proposed conversion prediction models, we have conducted experiments on real logged events collected from an advertising platform. Experiments show that our proposed URL embedding models are able to produce meaningful URL representations by grouping together URLs of the same category. Apart from that, our empirical analysis indicates that the user browsing history provides useful information for predicting users' visit on the advertiser's website, while the consideration of the chronological order of the visited URLs significantly improves the model's performance.

In the second part, we investigate the audience expansion problem. Audience expansion, also known as audience look-alike targeting, is

one of the major display advertising techniques that helps advertisers to discover audiences with similar attributes to a target audience (seed users) interested in advertisers' products or services. In this direction, we propose different (similarity-based) audience expansion schemes able to identify users with similar browsing interests to those of the seed users provided by the advertiser. The proposed schemes are mainly based on different self-supervised representation models that are able to capture the interests of the users according to their browsing history. After, based on the computed meaningful user representations, we compute the affinity score between users by using any standard similarity metric. In the end, we rank the users based on their affinity scores to their closest user in the seed set and we select the top-ranked users as the target audience. Our experiments show that the schemes that are based on the learning of compact user representation outperform the schemes that represent users just as a set of URLs and use standard or weighted Jaccard similarity metrics. Last but not least, our analysis points out that the seed users filtering (we only consider the users where their sequences of URLs are not dominated by a single URL) plays a significant role on the performance of the similarity-based audience expansion schemes as it alleviates the effect of the websites' automatic refreshments.

RÉSUMÉ

L'affichage publicitaire en ligne est devenu de plus en plus populaire ces dernières années grâce à l'automatisation du processus d'achat des inventaires. Spécifiquement, les enchères en temps réel (Real-time bidding en anglais, ou RTB) permettent l'échange automatisé d'impressions publicitaires entre les annonceurs et les éditeurs via des enchères en temps réel, au niveau de l'utilisateur. L'objectif principal des campagnes RTB est d'aider les annonceurs à cibler la bonne personne, dans le bon contexte, avec la bonne publicité et au bon moment. Par conséquent, l'identification précise de la 'valeur' d'un utilisateur pour l'annonceur est très importante. Dans ce contexte, nous examinons deux problèmes complexes de l'affichage publicitaire: la prédiction de conversion et l'extension d'audience. Dans les deux tâches, nous considérons uniquement l'historique de navigation de l'utilisateur comme caractéristiques, collectées à partir de données réelles.

Dans la première partie de cette thèse, nous examinons le problème de la prédiction de conversion, où notre objectif est de prédire si un utilisateur se convertira (c'est-à-dire, s'il visitera le site web de l'annonceur, s'il achètera son produit, etc.) ou non vers un annonceur donné. Inspirés par le traitement du langage naturel, nous introduisons trois modèles auto-supervisés de plongement d'URL afin de produire des représentations d'URL sémantiquement significatives. Ensuite, nous avons examiné trois différentes fonctions de projection pour représenter les utilisateurs qui prennent en entrée les représentations d'URL déjà apprises. La première renvoie la moyenne des vecteurs de plongement d'URL présentés dans l'historique de navigation de l'utilisateur. La seconde considère également les dépendances entre les caractéristiques du vecteur de plongement renvoyé par la fonction de projection moyenne. La troisième utilise le réseau de neurones récurrents à mémoire court et long terme (LSTM) bien connu qui est adapté au traitement des séquences de longueur variable. L'avantage principal de cette fonction est d'apprendre la représentation de l'utilisateur avec une prise en compte de l'ordre chronologique des URLs dans la séquence. Enfin, après avoir calculé les représentations des utilisateurs, nous utilisons le modèle de régression logistique pour prédire les probabilités de conversion. Pour démontrer l'efficacité des différents modèles de prédiction de conversion proposés, nous avons mené des expérimentations sur des événements réels collectés à partir d'une plateforme publicitaire. Les expériences montrent que nos modèles de plongement des URLs proposés sont capables de produire des représentations d'URL significatives en regroupant les

URLs de la même catégorie. Par ailleurs, notre analyse empirique indique que l'historique de navigation de l'utilisateur fournit des informations utiles pour prédire la visite des utilisateurs sur le site web de l'annonceur, et aussi que la prise en compte de l'ordre chronologique des URLs visitées améliore considérablement la performance du modèle.

Dans la deuxième partie, nous étudions le problème de l'extension d'audience. L'extension d'audience, également connu sous le nom de ciblage 'look-alike', est l'une des principales techniques de l'affichage publicitaire qui aide les annonceurs à découvrir des audiences présentant des attributs similaires à un public cible (seed users) intéressé par les produits ou services des annonceurs. Dans cette direction, nous proposons différents schémas de l'extension d'audience basés sur la similarité (similarity-based), qui sont capables d'identifier les utilisateurs ayant des intérêts de navigation similaires à ceux des utilisateurs de référence fournis par l'annonceur. Les schémas proposés sont principalement basés sur différents modèles de représentation auto-supervisés qui sont capables de capter les intérêts des utilisateurs selon leur historique de navigation. Ensuite, basé sur des représentations d'utilisateur significatives calculées, nous calculons le score d'affinité entre les utilisateurs en utilisant n'importe quelle métrique de similarité standard. Au final, nous classons les utilisateurs en fonction de leurs scores d'affinité avec l'utilisateur le plus proche dans l'ensemble des utilisateurs de référence et sélectionnons les utilisateurs les mieux classés comme public cible. Nos expériences montrent que les schémas basés sur l'apprentissage de la représentation compacte des utilisateurs surpassent les schémas qui représentent les utilisateurs simplement comme un ensemble d'URLs et utilisent des métriques de la similarité de Jaccard, standard ou pondérée. Enfin, notre analyse souligne que le filtrage des utilisateurs de référence (nous ignorons les utilisateurs dont les séquences d'URLs sont dominées par une seule URL) joue un rôle important sur les performances des méthodes d'extension d'audience basés sur la similarité, car il atténue l'effet des rafraîchissements automatiques des sites Web.

PUBLICATIONS

The following publications are included in parts or in an extended version in this thesis:

Qiu, Yang, Nikolaos Tziortziotis, Martial Hue, and Michalis Vazirgiannis (2020). « Predicting conversions in display advertising based on URL embeddings. » In: *AdKDD'20*.

Tziortziotis, Nikolaos, Yang Qiu, Martial Hue, and Michalis Vazirgiannis (2021). « Audience expansion based on user browsing history. » In: *International Joint Conference on Neural Networks (IJCNN) 2021*.

ACKNOWLEDGMENTS

First of all, I wish to express my sincere appreciation to my supervisor Prof. **Michalis Vazirgiannis** at École Polytechnique and Mr. **Vincent Mady** at Jellyfish France, without whom this dissertation would not have been possible. I'm so grateful that they provided me with this precious opportunity to tackle real-world problems using cutting-edge machine learning models, which allowed me to gain valuable experience on both the academic and industrial sides. Prof. Vazirgiannis is a fantastic advisor. Throughout my Ph.D. studies, he has consistently led, encouraged, and supported me, both academically and personally. During the toughest period of my Ph.D., he continued to believe in me and gave me the freedom to find my way to get out of the situation. I really appreciate it. I also thank Mr. Mady for having faith in me and having taken great effort to initialize and establish this Ph.D. project. In addition, I gratefully acknowledge Jellyfish France who funded my work.

Secondly, I would like to thank my supervisors in the company, also my close collaborators: Dr. **Martial Hue** and Dr. **Nikolaos Tziortziotis**, for keeping a close eye on the progress of my Ph.D. and their tireless support in the preparation of my presentations, papers, and dissertation. Due to the company's personnel changes, Martial took me under his responsibility in the middle of the project. He swiftly deciphered the clues and pointed me in a workable direction. I learned a lot from his clear mind and problem-solving skills. Nikos joined us later and gave me tremendous help. I'm always impressed by and indebted to his expertise, efficiency, and rigorous attitude. I can hardly express how fortunate I have been to have had him by my side, working out all the puzzles and issues together.

Furthermore, I want to express my great gratitude to the distinguished researchers: Prof. **Nikos Paragios**, Prof. **Dimitrios Gunopulos**, Assistant Prof. **Oana Balalau**, Prof. **Themis Palpanas**, Prof. **Ioannis Tsamardinos**, Prof. **Jean-Marc Steyaert** for agreeing to be part of the committee members of my Ph.D. defense. I highly appreciate their valuable feedback on my work and thought-provoking questions during the defense. Particularly, I would like to thank Prof. Nikos Paragios and Prof. Dimitrios Gunopulos for taking their precious time to review my dissertation and providing insightful comments.

In addition, I would like to thank all my former and current colleagues, who I had the pleasure of interacting with and learning from as well. In particular,

- from École Polytechnique: Guokan Shang, Changmin Wu, Sammy Khalife, Olivier Pallanca, Stratis Limnios, Christos Xypolopoulos,

Paul Boniol, Guillaume Salha, Giannis Nikolentzos, George Dasoulas, George Panagopoulos, Johannes Lutzeyer, Jesse Read, Maria Rossi, Konstantinos Skianis. A special thanks to my 'roommate' Guokan for his warm support and inspiring discussions.

- from Jellyfish France: Rafik Khereddine, Zied Yakoubi, Marouane Azlaf, Dia Al Jrab, Charles Monzani, Matthieu Brito Antunes, Mehdi Erraki, Pierre Mary, Hicham Akaoka, Lucas Merlette, Gaylord Cherencey, Sidi Mohamed Boukhary. A special thanks to Rafik Khereddine and Zied Yakoubi, for taking charge of me at the early stage of my Ph.D..

Last but not least, I would like to thank my family, friends, and loved ones, for their presence and support along the way. I'm especially grateful to my parents, Liping Zhu and Xiaoda Qiu, for their endless love and unwavering support. I owe them so much.

A big thanks again to everyone who has accompanied me on this wonderful adventure, whether my memory permits me to add them here or not.

Yang Qiu
Paris, November 2021

NOTATIONS

This section provides a brief reference describing the notations used through out this dissertation. Specific notations are given inside each chapter.

Notation	Description
x	A scalar
$\boldsymbol{x}$	A vector
$\boldsymbol{X}$	A matrix
$\boldsymbol{X}^\top$	Transpose of matrix $\boldsymbol{X}$
X_{ij} or $X_{i,j}$	The entry of i-th row and j-th column of matrix $\boldsymbol{X}$
x_+ or x^+	Non-negative part of x, i. e. , $max(0, x)$
$\{x_1, x_2\}$	A set containing x_1 and x_2
(x_1, x_2) or $[x_1, x_2]$	A sequence constructed by x_1 and x_2
$\mathbb{R}$	Set of real numbers
$\mathbb{Z}$	Set of integers
$\|\boldsymbol{x}\|_F$	Frobenius norm of $\boldsymbol{x}$
$\|\boldsymbol{x}\|_p$	L_p norm of $\boldsymbol{x}$
$\boldsymbol{X} \circ \boldsymbol{X'}$	Hadamard product (element-wise product) of $\boldsymbol{X}$ and $\boldsymbol{X'}$
$\nabla_{\boldsymbol{x}} f$	Gradient of f with respect to $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Integral of f over $\boldsymbol{x}$
$\exp(x)$	Exponentiation of x, i. e. , e^x
σ	Sigmoid function
tanh	Hyperbolic tangent function

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Advertising is the business of trying to promote products, services or ideas to people. Due to its effectiveness (Pergelova, Prior, and Josep, 2010), nowadays ads appear everywhere with different formats in our daily life. You see them on posters on the street, on pages of newspapers, or on the web-pages you visit. You hear ads on the radio or between the songs you play via an application on the telephone. You also watch ads on TV, Youtube, etc. In the beginning, advertising was mainly used to persuade customers to buy products. With the gaining power to do precise targeting (more data, more powerful algorithms, more calculation resources), its aspect of idea influence started to get more attention. For example, Gerber et al., 2011 claim the strong but short-lived effects of televised campaign ads on US president voting preference. And as a fact, the US political advertising campaign spend has reached around 6 billion dollars [1], doubled from 2014. On the other side, people have noticed the inevitable influence of advertising (Pollay, 1986) and they have started to be aware, or even afraid of it (a steady 30% of the internet users use an ad-blocker [2]). The governments have also forced regulation such as General Data Protection Regulation (GDPR) in Europe, to protect users' privacy. The present challenge for advertising is how to keep the personalization power while preserving users' privacy.

EVOLUTION OF ADVERTISING: FROM TRADITIONAL TO ONLINE

The history of advertising has only been a few hundreds of years. Nevertheless, it evolves very fast along with the development of the society. It has become more apparent during the last few decades with the invention and growth of the internet and trend of digitization.

The early sign of advertisement can be traced back to a long time ago when Egyptians used papyrus to pass sales messages. The first printed advertisement was introduced by William Caxton, who printed advertisements for his book in the 1470s in England [3]. After that, in 1704, the first newspaper advertisement was published by the Boston

1. Political Ad Spend to Reach $6 Billion for 2020 Election: https://www.emarketer.com/content/political-ad-spend-to-reach-6-billion-for-2020-election
2. Ad Blocking Growth Is Slowing Down, but Not Going Away: https://www.emarketer.com/content/ad-blocking-growth-is-slowing-down-but-not-going-away
3. 10 Clever Tips to Inspire Your Next Print Advertisement Campaign: https://blog.bannersnack.com/print-advertisement/

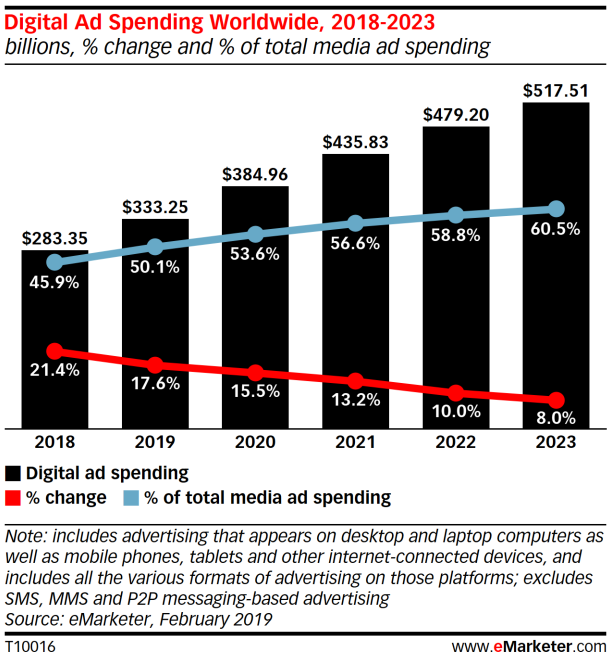


Figure 1.1 – Worldwide digital ad spending from 2018, with forecast to 2023. Figure taken from eMarketer [5].

Newsletter in the U.S. Then in 1922, radio advertising [4] came out as radio stations started to get business licenses and looked for profit. In 1941, the first TV advertisement was shown on WNBT for a Bulova watches company for 10 seconds.

The aforementioned traditional advertising channels (newspapers, magazines, radio, television) work quite well in general (Vries, Gensler, and Leeflang, 2017) and are still widely used nowadays. However, the degree of personalization stays limited at the group-level. For example, people who read the same newspaper receive exactly the same advertising message. The possibility of fine-grained personalization to person-level arrived when *online advertising* rose as a modern advertising method. On the internet, people express their intention by searching answers in search engines, posting status on social media, buying products on e-commerce websites, etc. By collecting richer data from the users and the online media that they interact with, advertisers can better understand the users' interest and need for both short term and long term, and provide specific and personalized advertisements accordingly. As a result, online (digital) advertising keeps growing rapidly with its spend reaching almost half of the global ad spending in 2018 and is forecasted to be more than half for the year 2020 (see Figure 1.1).

4. History of Radio Advertising: https://study.com/academy/lesson/history-of-radio-advertising.html

5. Global Digital Ad Spending 2019: https://www.emarketer.com/content/global-digital-ad-spending-2019

EVOLUTION OF ONLINE DISPLAY ADVERTISING ECOSYSTEM

At the beginning, advertisers were negotiating and making deals directly with publishers (i. e. , the owners of websites), to buy a certain amount of ad spaces at a fixed price, known as *Direct Deals*. The price was usually proposed based on a thousand ad displays (*impressions*), called *cost-per-mille* (*CPM*). Then, publishers were responsible for ensuring their delivery. Nevertheless, advertisers had little control on when, where and to whom the ads were going to be shown. After, due to the booming development of the internet, the number of websites and webpages exploded, so did the number of inventories. Thus, a lot of inventories were left unsold after Direct Deals. *Ad Network* (*AdN*) came as an intermediate broker, gathering remnant inventories from multiple publishers, grouping them (e. g. , based on the contextual information of the websites pages (Shuai Yuan, Jun Wang, and X. Zhao, 2013)), and selling them in bulk to advertisers according to their targeting rules. A common practice for advertisers was to connect to multiple AdNs in order to find enough high-quality cheap inventories. However, when more and more AdNs came into the market, it becomes hard for the advertisers to choose which AdN to work with because they often apply different strategies to group inventories. Furthermore, it is difficult to split the advertising budget among many AdNs efficiently. To overcome these issues, *Ad Exchange* (*AdX*) has been created. Similar to a stock exchange, AdX matches the ad transaction needs between advertisers and publishers in a programmatic way. This makes the whole process transparent as the advertiser knows exactly when, on which website and to which user his ad is going to be shown. Meanwhile, the focus of the advertiser starts to shift from the content of webpage to the user.

In the following five sections (Sec. 1.1 to Sec. 1.5), we navigate different aspects of the Real-time Bidding (RTB) ecosystem. In Section 1.1, we first introduce major programmatic media buying types that applied in online display advertising, then describe the ad delivery flow of RTB along with its main participants. In Section 1.2, we introduce two main auction mechanisms (first price auction and second price auction) applied in RTB. Section 1.3 describes the performance measures used in RTB campaigns. In Section 1.4, we explain how the user is identified across RTB participants using cookie, as well as the privacy concerns arise with. Finally in Section 1.5, we discuss RTB data quality issues (data noise and ad fraud). For the rest parts of this chapter, we give an overview of our contributions [6] realized in this dissertation in Section 1.6 and state the thesis organization in Section 1.7.

6. For the background required to follow this part, one may need to refer to Chapter 2.

1.1 DISPLAY ADVERTISING & REAL-TIME BIDDING

Online advertising ecosystem involves three major participants:

- *Publishers*: Ad slots (ad inventories) owners. Their main objective is the increase of their revenue by selling their ad spaces at a high price.
- *Advertisers*: Ad slots buyers. In general, they are companies who want to promote their products or services. With an advertising budget, they buy ad inventories on media channels to display their ads, in order to fulfill their advertising strategies (e. g. , increase brand awareness, acquire new customers, sell products).
- *Users*: Ads audiences. With their own intent (e. g. , searching for answers to their question, checking news, buying products), they browse on the media channels of publishers. In the meantime, they watch ads and react accordingly (e. g. , positive feedback: click ads, visit advertiser's website, buy advertiser's products; negative feedback: do nothing after seeing an ad).

For instance, when I see an ad of brand `A` on a website `B`: I am the user, brand `A` is the advertiser, and website `B` is the publisher. The whole advertising value chain is established around these three participants. Intermediate players run their business by helping them optimize their objectives (e. g. , increasing publisher's revenue, enhancing advertiser's brand awareness, improving user's browsing experience).

In practice, various types of advertising methods are used to reach users through different channels, such as

- *Display advertising*: Advertisers use visual ads forms (text, images, videos, etc.) on internet channels (websites, apps, social media, etc.) to deliver their ad messages.
- *Paid search*: It is also known as *Search Engine Marketing* (*SEM*). Advertisers pay search engines (e. g. , Google, Bing) to show their ads (usually in text forms) at the top of the search results.
- *Email marketing*: Advertisers send advertising messages through emails, usually to a specific group of users.

Among them, display advertising is the most popular now, mainly due to the reason that it can influence the users across the whole *conversion funnel* (i. e. , awareness, interest, desire, action, loyalty)[7]. The other advertising types focus mainly on part of it. For example, email marketing focuses mainly on increasing brand awareness. As a fact, display advertising is estimated to get more than half of the digital ad spending of the US in 2019 (Fig. 1.2).

It is worth mentioning that for the definition of display advertising, in a narrower but common extent, the internet channels are restricted

7. The consumer decision journey: https://www.mckinsey.com/business-functions/marketing-and-sales/our-insights/the-consumer-decision-journey

Digital Ad Spending in the US, by Format, 2019-2023
billions

	2019	2020	2021	2022	2023
Display	**$70.06**	**$81.38**	**$92.53**	**$100.71**	**$107.91**
—Video	$36.01	$42.58	$49.02	$53.99	$58.39
—Banners and other*	$25.94	$30.20	$34.23	$37.12	$39.70
—Rich media	$5.28	$5.44	$5.85	$6.00	$6.10
—Sponsorships	$2.84	$3.16	$3.44	$3.60	$3.72
Search	**$53.73**	**$63.90**	**$73.31**	**$80.43**	**$87.15**
Lead generation	**$2.59**	**$2.84**	**$3.05**	**$3.15**	**$3.24**
Classifieds and directories	**$2.19**	**$2.34**	**$2.49**	**$2.53**	**$2.56**
Email	**$0.49**	**$0.55**	**$0.61**	**$0.65**	**$0.68**
Mobile messaging	**$0.28**	**$0.28**	**$0.29**	**$0.29**	**$0.29**
Total	**$129.34**	**$151.29**	**$172.29**	**$187.77**	**$201.83**

*Note: includes advertising that appears on desktop and laptop computers as well as mobile phones, tablets and other internet-connected devices on all formats mentioned; *includes ads such as Facebook's News Feed Ads and Twitter's Promoted Tweets*
Source: eMarketer, February 2019

T10056 www.eMarketer.com

Figure 1.2 – Online ad spending in the US, by format. Figure taken from eMarketer.

to the websites. In this dissertation, **we use this common extent unless special instruction**.

1.1.1 *Programmatic media buying*

Traditional media buying is accomplished through direct deals between advertiser and publisher which involve a lot of manual work. Human intervention is needed typically in processes such as requests for proposals, price negotiation, campaigns setting and optimization, which make the dealing procedure slow and inefficient. Additionally, ads are purchased in bulk where buyers have little control over the condition under which the ads are going to be shown. *Programmatic media buying* utilizes programs (and machine learning algorithms) to automate and optimize this process which dramatically reduces the operation cost and boosts the performance of ad campaigns. As a fact, more than 80% of the digital display ad budget is spent programmatically nowadays [8].

At first, programmatic media buying is executed in a sequential manner, based on the priority that the publisher set to each buyer, known as *waterfall* (or *daisy-chaining*) model. Under this model, programmatic advertising can be categorized to four types with priority from high to low at the publisher side:

1. *Programmatic Guaranteed*,
2. *Preferred Deals*,
3. *Private Marketplace* (*PMP*),

8. US Programmatic Digital Display Advertising Outlook 2021: https://www.emarketer.com/content/us-programmatic-digital-display-advertising-outlook-2021

Properties \ Types	Programmatic Guaranteed	Preferred Deals	PMP	RTB
Auction	No	No	Yes	Yes
Buyer	One	One	Multiple	Multiple
Ad Server Priority	Highest	Above PMP	Above RTB	Lowest
Impressions Guaranteed	Yes	No	No	No
Negotiation	Yes	May or May not	Minimum	No

Table 1.1 – A summary of programmatic buying types[9].

4. *Real-Time Bidding* (*RTB*).

These types are summarized in Table 1.1, depending mainly on whether an auction is involved and whether the publisher guarantees the amount of inventory delivery.

In Programmatic Guaranteed and Preferred Deals, no auction takes place and the publisher faces only one buyer. For the rest two types with lower priorities, the inventories are sold in auction where multiple buyers are involved.

PROGRAMMATIC GUARANTEED Also known as *Programmatic direct*, *Programmatic reserved*. After negotiation between the publisher and the advertiser (or its representative), the publisher promises to deliver a guaranteed number of impressions at a fixed price (e. g. , fixed CPM) to the advertiser, following its desired targeting setting (audiences list, ad positions, etc.). This process is similar to Direct Deals but in an programmatic (automated) way. It's beneficial for both sides: From the publisher side, it may be willing to sell premium inventories (e. g. , homepage takeovers) here, because it can have a guaranteed revenue and strengthens the long-term relationship with the advertiser. From the advertiser side, as it knows on which websites the ads are going to be shown, it reduces his brand safety concerns. The advertiser is willing to pay a generally high price for these premium inventories.

PREFERRED DEALS Also known as *Private access*, *Spot buying*. Compared to Programmatic Guaranteed, in Preferred Deals, the price of inventory is still fixed but the amount is not guaranteed. For the advertiser, he still has the privilege to check the inventory before it goes to auction. For the publisher, the risk is that as the amount of impressions is not guaranteed, when the fill rate is low (i. e. , the advertiser doesn't buy enough), a lot of inventories go to auction directly.

9. Source: The Four Types of Programmatic Deals https://www.adpushup.com/blog/explainer-the-four-types-of-programmatic-deals/

PRIVATE MARKETPLACE (PMP) Also known as *Private Auction*, *Invite-only Auction*. Inside PMP, the auction is private for invite-only buyers. This makes it more safe compared to open auctions as publishers can invite only the buyers that they trust (e. g. , to prevent ad fraud). Still, it is possible that some potential advertisers who are willing to pay a high price are not invited due to lack of information on the publisher side.

REAL-TIME BIDDING (RTB) Also known as *Open Auction*, *Open Marketplace*. RTB is the most popular type of programmatic advertising. It is an open auction that any connected advertiser and publisher can participate in, which results large volumes of available ad inventories. As regard of the good aspects, it's an efficient way for publishers to sell their remnant 'tail' inventories to gain potential high revenue through auctions, without worrying about the fill rate. Also the set up is easier compared to the other programmatic advertising types which may still need deal negotiations. On the other hand, due to large supply of ad inventories, the average price of impressions is relatively low. And as it is an open auction, ad frauds (e. g. , malicious creatives) happen more than other types where the buy side and sell side know each other generally well. Finally, in this open auction mechanism, accurate estimation of the value of each impression opportunity is crucial to the final revenue of each side, because no delivery guarantee exists and the number of competitors are quite high. Therefore, a too optimistic bidding may waste money while a too pessimistic bidding may loss winning opportunity.

As we have already seen, all these programmatic ad buying methods have their own pros and cons. Therefore, advertisers and publishers often use a combination of them to optimize their advertising strategies. Two popular combination models exist nowadays: *Waterfall* and *Header bidding*. The main difference between these two is that in the waterfall model, aforementioned ad buying methods are combined in a sequential manner while in header bidding, they are executed simultaneously.

Roughly speaking, in the waterfall model, the impression opportunity is first given to the buyers at higher priority. If they decline, it is passed to the ones standing at lower priority, with a lower *floor price* (the minimum price that the publisher is willing to sell the impression opportunity). This process repeats until someone clears the floor price, otherwise the publisher runs some in-house ads. Despite its simplicity, it has two main drawbacks: The first one is the latency issue. This one-by-one decision schema makes the execution to be potentially slow. The second is that the publisher may not get the highest price for his inventories. It happens because it's highly possible that the inventory is sold to the prior buyer even when the lower-ranked non-prior buyer offers a better price.

To overcome these drawbacks, Header bidding is introduced and gets popular. Under Header Bidding, the price offered from different participants are considered together at the same time, with the same priority. The simple rule is the offer with the highest price (higher than floor price) wins. In this way, the publisher generally increases its revenue by always selecting the highest price. Moreover, the decision is made once only.

However, Header bidding is still not perfect. The latency issue is not entirely solved because in the original format of Header bidding, ad requests are treated simultaneously at the user's browser which causes long loading time. The existing solution is to move this process to a specific server and the server just sends back the final decision. Also as a large part of inventories are exposed to any buyer, it's much easier for fraudsters to collect the customers' data of the publisher without his permission. A practical way to deal with this data leakage is to limit the information provided in bid requests to programmatic buyers and let it be adjustable according to the buyers' reliability and priority level. Compared to its downside, the advantages of Header Bidding are more attractive. As a fact, in 2019, nearly 80% of the 1000 most popular US websites (based on Alexa ranking) that sell ads in a programmatic way used Header Bidding [10].

1.1.2 *Real-Time Bidding*

Real-Time Bidding (RTB) is considered as a game-changer for digital advertising mainly due to the fact that the ad trading is done at per inventory (impression) basis, which makes the trading process transparent and targeting a specific user, at specific website, at a specific time to be possible.

Also, the adoption of Header Bidding gives advertisers the possibility to buy 'premium' inventories through RTB. While in the waterfall model, RTB mainly gets tail inventories as it stands at the lowest priority.

In Figure 1.3, we present an overview of the ad delivery flow in RTB, where each rectangle represents a key role (participant) in the RTB process:

- Demand-Side Platform (DSP): DSPs are tools (platforms) used by advertisers, to help them optimize their advertising strategies. They typically use algorithms to determine the 'value' of each impression based on their knowledge about the user (e. g. , behavior, geo-location, etc.) and offer a bid price wisely. They also provide advertisers with functionalities to set up and track their ad campaigns easily.

10. Five Charts: The State of Header Bidding: https://www.emarketer.com/content/five-charts-the-state-of-header-bidding

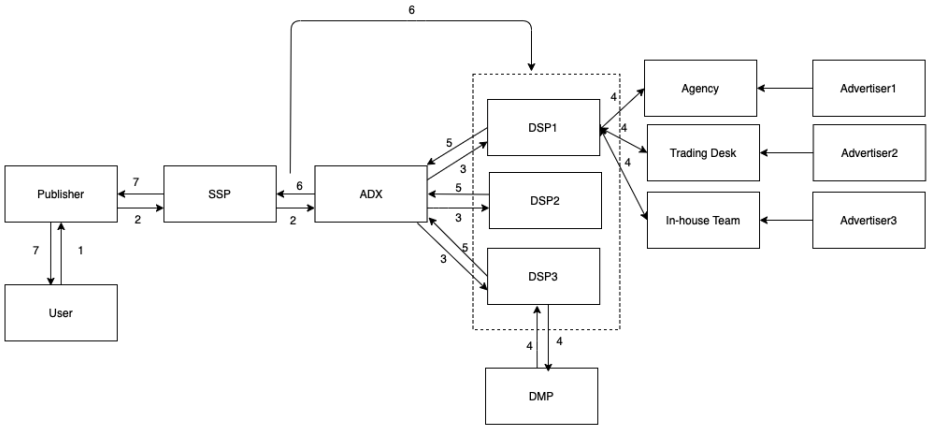


Figure 1.3 – An overview of the ad delivery flow in RTB.

- Supply-Side Platform (SSP): SSPs are tools (platforms) used by publishers, to help them manage and sell their ad inventories wisely in order to increase the revenue. Their functionalities include: floor prices setting, allocating the inventories to different partners, etc.
- Ad Exchange (AdX): It is an intermediate platform that connects DSPs and SSPs. It matches the ad transaction needs between them automatically, impression by impression. Typically it is done by assigning the impression to the DSP that offers the highest price.
- Data Management Platform (DMP): DMP is a platform that collects, cleans and reorganizes user data from multiple resources. It provides other participants (mainly DSPs) with extra user information that they can use to understand and model better the user.
- Agency or Trading Desk: They delegate the advertiser to connect to DSPs. Advertisers may not have their own in-house team to operate ad campaigns for different reasons (e. g. , lack of time, insufficient budget), instead, they hire agencies or trading desks to help them manage their advertising budget and optimize their advertising campaigns.

Actually, a typical ad delivery flow (see Fig. 1.3) can be described as follows:

1. An internet user visits the webpage of a publisher.
2. For each available ad slot on the webpage, the publisher sends a bid request to the connected SSP and the SSP sends it to the AdX, together with the floor price. The bid request mainly contains the information about the website (e. g. , URL, publisher ID) and the user (e. g. , user ID, geo-location, browser type) and the ad inventory (e. g. , position, size).

3. After the AdX receives the bid request, it transfers the bid request to multiple DSPs at the same time.
4. Each DSP decides if it will submit or not a bid response for this impression opportunity, based on its information about the user (extra user information may be provided by DMP), the website, the ad to put on, also the ad campaign setting of the advertiser (or their representative: agency, trading desk).
5. Each DSP sends the bid response (with the bid price) to the AdX, under a constrained time.
6. The AdX determines the highest bid sent by the DSPs. If the highest bid price is higher than the publisher's floor price, the impression opportunity is sold to the corresponding advertiser that generally pays the second highest bid price (*second price auction*, Vickrey, 1961). Otherwise, this inventory is considered unsold. In the end, AdX announces the auction result to the SSP and all the DSPs.
7. The SSP helps display the ad of the winner (advertiser) on the corresponding ad slot of the publisher.

In fact, the whole process will be finished in around 100 milliseconds, that's why it's called 'real-time'.

In this dissertation, we focus on the banner case of display advertising under the RTB process.

1.2 AUCTION MECHANISM

As aforementioned, in Real-Time Bidding (RTB) an open auction is launched inside the Ad Exchange (AdX). Each bidder (typically the DSP who bids on behalf of the advertiser) proposes his bid price, and the AdX platform assigns the impression opportunity to the bidder offering the highest price. The winner either pays the highest bid price (*first price auction*) or the second highest bid price (*second price auction*) depending on the type of auction. As the cost of a single impression is relatively low in practice, the bid price is counted per thousand impressions, referred as *cost-per-mille* (CPM) or *cost-per-thousand* (CPT).

For example, suppose that we have three bidders A, B and C, who offer bid price at 1.6$ CPM, 3$ CPM, 0.5$ CPM respectively. Then B wins the bid as he offers the highest price. He will pay 3$ CPM in the first price auction and 1.6$ CPM [11] in the second price auction.

Meanwhile, the publisher may set a *reserve price* (also known as *floor price*) for his inventory (Myerson, 1981) which roughly represents the minimum price that his is willing to sell this inventory. If the highest bid is below this price, the inventory is unsold. Otherwise, the auction process is fulfilled normally by including the reserve price as

11. Usually the AdX charges extra 0.01$ CPM in second price auction (Edelman, Ostrovsky, and Schwarz, 2007).

the bid price of a 'virtual' bidder. Considering the previous example with the three bidders, if the reserve price is set to 2$ CPM, B still wins the auction as his bid is above the reserve price. He pays 3$ CPM in the first price auction and 2$ CPM in the second price auction (2$ is the corresponding second highest price). On the other hand, if the reserve price is set higher than 3$ CPM, the inventory is unsold. Setting the reserve price encourages the bidder to bid higher because if the bidder bids always below the reserve price, it cannot win any auction. However, if the reserve price is set too high (too aggressive), the publisher may not able to sell enough inventories. The discovery of the optimal reserve price is a challenging task that has been studied recently (Shuai Yuan, Jun Wang, B. Chen, et al., 2014). Moreover, for the same inventory, it is possible that the publisher sets a personalized reserve price for each bidder (Paes Leme, Pal, and Vassilvitskii, 2016).

In fact, the aforementioned reserve price is known as *hard floor* price. There exists also *soft floor* price (Zeithammer, 2019), where the situation is slightly more complicated. In a nutshell, the soft floor price is a bid level set above the hard floor, and acts as a switch between the first and second price auction model. When the highest bid is above the soft floor, a second price auction takes place. Otherwise, the first price auction takes place. For example, we keep the hard floor at 2$ CPM. If we set the soft floor at 5$ CPM, then B clears the auction at 3$ CPM because we are running the first price auction. If we set the soft floor at 2.5$ CPM, then B clears the auction at 2.5$ CPM because we are running a second price auction. Consequently, by properly setting the floor prices, the publisher can drive the maximum profit in the price range between the hard floor and soft floor (as it's running the first price auction) and keep the behavior of bidders with high bid price unchanged (as it's running the second price auction). Thus, by tuning the floor price of each inventory wisely, publishers are able to increase their revenue by selling their ad slots at higher prices (Shuai Yuan, Jun Wang, B. Chen, et al., 2014; J. Li et al., 2017).

In practice, the AdX platform typically employs the second price auction. In this case, the dominant bidding strategies for each bidder is to *bid truthfully*. It means that the bidder should bid exactly how much he values the impression (Matthews, 1995), no matter what the other bidders do (Milgrom, 2004). It has been shown that this strategy maximizes the *utility* (value of the impression - price paid) of the bidder. Whereas for the first price auction, there is no dominant strategy, each bidder needs to adapt his bid according to the other bidders' behavior. However, due to the nature design of the second price auction and the floor prices that publishers applied, buyers have little information about how much they will pay in the end. In the meantime, first price auction starts to grow in favor (e. g. , Google is

moving to first-price auction [12]) due to the transparency in the final price paid and the popularity of Header Bidding.

1.3 PERFORMANCE METRICS OF ADVERTISING CAMPAIGNS

In Real-time bidding, advertiser and publisher typically use CPM as the pricing model for impressions transactions. The publisher wants to sell impressions at high CPM while the advertiser wants to buy impressions (through DSP) at low CPM. However, based on the ad campaign's objective, the advertiser's ultimate goal can be different from that.

The objective that the advertiser wants to optimize is often referred to as *Key Performance Indicator* (KPI). Given a fixed advertising budget, the advertiser typically wants to generate a maximum number of **ad impressions**, **ad clicks**, and **conversions**. The 'conversion' here means certain valuable actions predefined by the advertiser such as visiting the advertiser's website, register as advertiser's customer, buy advertiser's product. Thus, advertisers often use normalized KPIs such as *cost-per-mille* (*CPM*), *cost-per-click* (*CPC*) and *cost-per-action* (*CPA*) as the maximum price they are willing to pay in average for each generated impression, click, conversion, respectively. DSP uses these KPIs target to determine the value of the impressions and guide his bid prices.

Taking CPA as KPI for example, we can roughly decompose the expected CPM value of one specific impression as:

$$\begin{aligned} CPM &= \frac{\text{ad spend}}{\text{number of impressions}} \times 1000 \\ &= \frac{\text{ad spend}}{\text{number of conversions}} \times \frac{\text{number of conversions}}{\text{number of impressions}} \times 1000 \\ &= CPA \times CVR \times 1000 \end{aligned} \tag{1.1}$$

where *CVR* indicates the *conversion rate*, i. e. , the probability that an impression will bring a conversion. Thus, given the CPA target of an advertiser, an accurate estimation of the CVR provides a reasonable value for the present impression opportunity. This value is commonly used by the DSP to offer his bid accordingly (e. g. , in second price auction, the optimal strategy is to bid truthfully at a price equal to the value of the impression, Sec. 1.2).

Correspondingly, the actual performance of DSP is judged by the *effective* KPI that he realized. For example, *effective CPM* (*eCPM*) is calculated by dividing the actual ad spend by the realized number of impressions. The main objective of DSP is to have the eCPM smaller than the CPM target provided by the advertiser, and keep it as low as possible.

12. Google Switches To First-Price Auction: https://www.adexchanger.com/online-advertising/google-switches-to-first-price-auction/.

1.4 COOKIE-BASED USER IDENTIFICATION & PRIVACY

In order to identify and possibly track users, websites generally use *cookies* (Kristol, 2001). Cookie is a small text file sent by the website and stored in the user's web browser, when the user first visits the website. It stores essentially the user's unique identification (*user ID*) associated with this website, but it can also contain other information about the user such as language and theme preferences, browsing activities. Each time the user revisits the website, this cookie will be sent back to the website, allowing it to identify the user and generate personalized recommendations, contents, etc. It should be noticed that each cookie is website-specific (domain-specific), which means that website `A` can not read the cookie of website `B` and vice versa. For example, the same user is known as `USER_123` to the website `A` but as user `USER_456` to the website `B`.

In the context of RTB ecosystem, each platform (DSP, SSP, DMP, etc.) acts as a 'website' which has his own user identification system. Taking DSP for example, with the permission of his clients, he places a specific code on the clients' (advertisers') websites. Each time when the user visits the advertiser's website, this code makes him requests simultaneously the DSP's domain. As a result, the DSP can create his own cookie to identify users across his clients' websites. For instance, if website `A` and `B` are both the DSP's client, then the DSP can match the user id `USER_123` and `USER_456` with his own user id `USER_ABC`. Actually, the type of cookie on DSP side is referred as *third-party* cookie while the cookie created on the website that the user actually visits is referred as *first-party* cookie.

COOKIE MATCHING The aforementioned mechanism does not allow the user information to be synchronized between two different platforms. However, as we have seen previously, the user needs to be identified across different RTB platforms (DSP, SSP, DMP, etc.) to realize efficient ad bidding and user targeting. *Cookie matching* technique, also known as *cookie mapping* or *cookie syncing*, is commonly used as a solution to this problem. In fact, this matching is done along with the ad request and commonly implemented through exchanging a URL of invisible pixel image between two platforms. To be more precise, let's consider the user matching between SSP and DSP for instance. When a user visits a website `A` which has an ad slot, website's SSP partner can drop its cookie and create the corresponding user ID due to the specific code placed on website `A`. In the meantime, the SSP also creates a URL corresponding to an invisible image (dimension 1 pixel $\times$ 1 pixel, known as *pixel tags*) served from its DSP partner. The URL of this image includes the user ID of the SSP side (`USER_ABC`). Loading this URL (served from DSP) on the user's web browser allows the DSP to drop his cookie and creates the user ID of the DSP side (`USER_XYZ`).

As DSP can extract the user ID of the SSP side from the given URL, he creates a *matching-table* that maps the user ID of both sides. Also, the DSP can do the same reversely (puts the user ID of the DSP side on a URL and sends it back to the SSP) to let SSP have his own matching table. Thus, using this trick, we can synchronize the user ID between platforms.

The main downside of cookie-based user identification mechanism is that it relies totally on the existence and consistency of the cookie. For instance, if the user clears the cookies of his browser or changes the browser, then this user is treated completely as a new user. Alternative choices such as *device fingerprint* (Laperdrix et al., 2020) uses features of device's hardware and software to provide unique identification of the user using an *fingerprinting* algorithm (Rabin, 1981, Broder, 1993). The fingerprinting algorithm can be seen as a high-performance hash function. Device fingerprint can be used solely or combined with the usage of cookies to enhance the identification ability. Eckersley, 2009 states that in the best case, only 1 in 286,777 browsers will share its fingerprint with the others.

PRIVACY Using cookie (matching) enables identifying users and sharing data across different RTB players, offering better ad targeting. For instance, a user may visit an advertiser's website, browse some products and leave. When the same user arrives at another website, the advertiser can still target him (*retargeting*) by showing the products that he has looked or other recommended similar products, in order to bring him back to the advertiser's website to finish the purchase.

However, this also makes the user feel being tracked continuously and raises the concern of user privacy, even though certain people consider this as a trade-off for convenience and personalization (Kokolakis, 2017). As a fact, according to a survey of Pew Research Center [13], the majority of Americans (72%) feel all most of what they do online is being tracked by advertiser and (79%) concern about the usage of their data by the companies. For this reason, 42% of internet users report using an ad blocker [14]. People also use *virtual private network* (*VPN*) to mask their true IP addresses and *browser in private* to prevent activities tracking. In the meantime, regulations are established to protect user's data and privacy. For example, *General Data Protection Regulation* [15] *(GDPR)* has come into force across the European Union since 2018. Its most important idea is the *consent*. It demands that the organizations who want to collect personal data from users, must ask

13. Americans and Privacy: Concerned, Confused and Feeling Lack of Control Over Their Personal Information: https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-personal-information/

14. Ad Blocker Usage and Demographic Statistics in 2021: https://backlinko.com/ad-blockers-users/

15. Complete guide to GDPR compliance: https://gdpr.eu/

for the users' consent in a clear way. Actually, the user needs to have the option to choose whether he accepts the cookie or not.

POST THIRD-PARTY COOKIE ERA Due to the complexity of complying strictly with GDPR and rising concerns about user privacy, many browsers (Google [16], Firefox [17], Safari [18]) have started to block third-party cookies. Thus, the original targeting strategies which rely heavily on user tracking through third-party cookies need to be adapted to the new situation. There are several possible solutions being studied recently:

- Back to contextual advertising
  As we mentioned previously, RTB shifts the ad targeting focus from the websites' context to the users. Under the disappearance of third-party cookies, it seems that returning back to contextual advertising (Anagnostopoulos et al., 2007; Niu, J. Ma, and D. Zhang, 2009) could be a workaround. The main benefit is that it's not impacted by user privacy regulations. Also, as it's relevant to what you are looking at, and it usually does not interrupt your browsing experience. This is the core idea of *Native advertising* (Manic, 2015) which is emerging in news feeds and social media domains.
- Federated Learning
  Federated Learning of Cohorts (*FLoC*, Google, 2020) is a type of web tracking method based on *Federated Learning* (Konečný et al., 2016). It is proposed by Google in its *Privacy Sandbox* project (Google, 2021). The main idea is to keep the user data (e. g. , online activity) locally on the device (e. g. , browser) and use it to deduce the interest of the user. After, we group users who share similar interests using SimHash (Charikar, 2002) algorithm. For a given user, we can send his cohort ID (group ID) to the AdX to realize interest-based targeting. Thus, we preserve user's privacy by storing user data locally and weaken the user identification concerns by providing only his cohort ID.
- Rethink the usage of the first-party data
  As third-party cookies are banned, publishers and advertisers should rely more on their first-party data, even though that the number of users can be targeted are less than before. This situation encourages them to focus more on improving their knowledge about their customers and their user experience. For

16. Chrome tracking protection: https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html
17. Firefox tracking protection: https://blog.mozilla.org/en/products/firefox/todays-firefox-blocks-third-party-tracking-cookies-and-cryptomining-by-default/
18. Safari tracking protection: https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/

example, advertisers may design proper quizzes, interactive content or educational surveys to know their customer better, and use this knowledge to deliver personalized and relevant advertising messages that fit the needs and preferences of the users. In this way, customers gain trust on advertisers (publishers) and they are willing to reveal more personal information to them, in order to get more precise and personalized services, which help both parts in the long run.

1.5 DATA QUALITY

Garbage in, garbage out (*GIGO*) is a well-known sentence in the area of computer science, which means that one needs to ensure the quality of data before feeding it into the algorithm. In any other case, the learned algorithm will perform poorly or even be misleading.

In the context of RTB, considering DSP (ad buyer) as an example, we can collect log level data from the ad requests and user feedback. The log level data mainly come from five different sources:

- **Bid request:** data that describes a bid request.
- **Impression:** data returned if we won an auction.
- **Conversion:** data returned if the user converted and the conversion is attributed to our impression.
- **Click:** data returned if the user clicked on our impression.
- **Segmentation:** segment that the user belongs to.

where impression, conversion and click logs are used the most often.

Actually, each log (event) is an interaction result among user, publisher and advertiser, with features described as follows:

- **User side features:**
  - USER_ID: user ID.
  - RIP: user IP address (truncated).
  - ZIP: zip code of the user's address.
  - CITY: city name of the user's address.
  - UA: user agent. We can deduce the device type, browser type, operation system from it.
  - SEG_ID: segment ID. The segment that the user belongs to.
- **Publisher side features:**
  - PUB_ID: publisher ID.
  - URL: URL of the page.
- **Advertiser side features:**
  - ADV_ID: advertiser ID.
  - CP_ID: campaign ID.

  - CRE_ID: ad ID.
  - PL_ID: placement ID of the ad. It's can be roughly seen as an ID that represents the position of of the ad on the website.
  - SZ: size of the ad.

- **General features:**
  - AU_ID: auction ID. The unique id to identify a bid request. It is used to connect an impression to the winning bid request and to connect a conversion (or click) to the impression that generated it.
  - TM: timestamp of the log.
  - TY: a string that specifies the type of the event.

Additionally, each specific log has its own other features. For instance, in bid request data, it also has the value of the bid as a feature. In impression data, the view time of the impression is also noted. In click data, we generally know which ad the user has clicked.

The data quality in RTB is impacted from two aspects: i) *data noise* and ii) *ad fraud*. The noise of data rises mainly due to the nature of RTB data. It is easier to be detected with the help of posterior analysis, compared to the ad fraud which is mostly generated by non-human traffic.

DATA NOISE Due to the nature of RTB data, noise appears at different sources. Next, we give some examples of commonly seen data noise that comes from user ID, bid request, impression and click. At the same time, we show practical solutions to prevent or reduce the noise.

- **User ID noise:** As we mentioned previously in Section 1.4, the user is identified based on the cookie stored in his web browser. If the user clears his cookie or browses in private, he is considered as a totally new user. Thus, one single user may be identified with different user IDs in the system. To truly follow the user's entire browsing activities, additional features or detection mechanism are needed.
- **Bid request noise:** As we have seen previously, a bid request is created each time a user visits a website which has a biddable ad slot. An interesting case is that the website may apply an *auto refresh* mechanism. According to Google [19], websites can basically define different ways that an auto refresh of ads is triggered. For example, ads can be refreshed if there is no user activity on a predefined time interval (*time-based refresh*). That is

19. Google Real-time Bidding Proto: https://developers.google.com/authorized-buyers/rtb/realtime-bidding-guide#autorefreshsettings-object

to say, if you leave several windows of your browser open (it happens very often), those webpages keep generating bid requests without actual user visits. Also, ads can be refreshed due to an event-driven content change of the website, such as match scores change (*event-driven refresh*). Actually, this auto-refresh mechanism can create noise and harm the advertiser's ad performance. First, in the case of time-based refresh, the advertiser may spend extra money for non-viewable impressions as the user has actually left his screen. Second, one may want to reconstruct the browsing history of one user, by grouping visited URLs using bid request data. Nevertheless the constructed browsing history is not complete as not all websites have available and biddable ad slots. Additionally, due to the website auto refresh, the browsing history contains false visit traces. Therefore, if a model is directly built on the collected browsing history, it may be biased.

- **Impressions noise:** When you buy a banner impression, you know that it's loaded on the website. But you don't really know if the user has actually seen the ad. For example, for ads at the bottom of the webpage, it is possible that the user has closed the webpage before he scrolls down to their position. In practice, we typically use a feature related to "viewtime" of the ad to examine if it is actually seen by the user. For example, according to Google ads rules [20], a display ad is counted as viewable when at least 50% [21] of its area is visible on the screen for at least 1 second.
- **Clicks noise:** We typically treated the click of an ad as a positive feedback of the user. However, it's possible that the user just clicked it by accident. For example, when a user wants to close an ad in a mobile app, if the `close` button is too small or not well positioned, he can easily misclick it and close it quickly. To prevent advertisers from being charged for misclicks, some service providers like Yandex add an extra validation step for suspicious ad clicks where `Go to site` and `Cancel` buttons appear after the ad gets clicked [22]. Then, the advertiser is charged only when the user clicks the `Go to site` button. Apart from using specific web design, machine learning algorithms are also under study. Researchers try to combine other information in the data to define if a click is truly positive or not. For example, C. Wu et al., 2021 propose to separate the feedback in news feed into implicit weak feedback (click and skip), explicit feedback (share and dislike) and implicit strong feedback (finish reading or quick close). For this purpose, they use an attention network that uses

20. Understanding viewability and Active View reporting metrics: https://support.google.com/google-ads/answer/7029393?hl=en
21. For large display ads of 242,500 pixels or more, the percentage is set to 30%
22. Yandex confirming site clicks: https://yandex.com/support/partner2/technologies/antifraud.html

the representations of stronger feedback to distill positive and negative parts from implicit weak feedback such as clicks.

AD FRAUD Ad fraud, referred also as *invalid traffic*, is another cause of data quality issues in RTB. Ad fraud is generally harder to be detected than the data noise, because the counterpart (fraudsters) can use various tools and methods to fulfill the fraud, and also change their pattern regularly. Actually, in RTB, ad fraud generally comes from the publisher side and the advertiser side is the victim.

A common definition of ad fraud (Kotila and Dhar, 2016) is: "an activity where impressions, clicks, actions or data events are falsely reported to criminally earn revenue, or for other purposes of deception or malice". As explained in «IAB Europe's Guide to Ad Fraud» (IAB, 2020), there exist plenty ways of making ad frauds, generated not only by non-human traffic, but also by human traffic.

Taking the click frauds as an example, fraudsters may use sophisticated designed bots that click their ads automatically to generate revenue. But also they can hire a large group of low-paid workers to intentionally click their ads, known as *click farms*. Right after, we present three primary types of ad fraud: impression fraud, click fraud, conversion fraud, as follows.

- **Impression fraud:** This kind of fraud happens on impressions where fraudsters mainly want to cheat advertisers to buy ad slots that do not actually exist or of lower quality than expected.

  - Domain spoofing
    It's also known as *impression laundering*. The Fraudster (low-quality publisher) cheats advertisers by pretending to be a premium publisher, in order to make his ad slots more valuable and increase the demand. This is done through a number of complex redirects and nested ad calls, so that the advertiser sees premium publisher's URL instead of the fraud one. Domain spoofing not only wastes advertisers' money, but also produces brand safety concerns as advertisers' ads are actually displayed on low-quality websites.
  - Stacked and hidden ads
    Dishonest publishers may stack multiple ads on top of each other while only the topmost ad can be viewed, or simply make them all invisible. By doing this, they generate multiple impressions from a single page view which increases the ad traffic.
  - Ad injection
    Ad injection is a technique where fraudsters inject ads into a website without its knowledge or permission. The injected ads can be on top of the existing ads which makes them invisible (like ad stacking) or replace the existing ones or

even appear on pages that were not supposed to display ads at all. This is often caused by plugins installed on the user's web browser.

- **Clicks fraud:** In performance-based ad campaigns, advertisers are charged for each click or conversion generated. For click fraud, fraudsters generate false clicks in order to gain revenue from advertisers. We present several common click fraud types at following:

  - Bots
    Bots are computer programs that run automated tasks over the internet. In the area of advertising, the tasks can be view ads, click ads, watch videos, etc. To charge advertisers who use *payer-per-click* (*PPC*) pricing model (advertiser pays publisher every time the ad is clicked), fraudsters set up web pages and use bots to automatically click their ads. Simple bots are relatively easy to be blocked as they generally have static IP addresses, user agents, etc. Advertisers can simply move them to blacklist if an unusually high click-through-rate (CTR) is detected on those websites. Sophisticated bots, on the other hand, may change their user agent, IP address regularly, keep the click-through-rate normal, or even imitate human click behavior to surpass the detection.
  - Botnets
    Botnets are typically referred to computers or devices that have been controlled by fraudsters using malware or computer virus. The owner of the device usually is not aware that the device has been partially or fully manipulated. Fraudsters then use these botnets to click ads for them. It's hard to detect and block botnets, because except for the suspicious click behavior, all other characteristics correspond to real users.
  - Click Farms
    Fraudsters hire a large group of low-paid workers to intentionally click their ads and even fill forms, in order to generate clicks and conversions.

- **Conversions fraud:** For conversion fraud, fraudsters generate false conversions in order to gain revenue from advertisers.

  - Click fraud methods
    In general, aforementioned methods which generate click fraud can be also applied in conversion fraud. As advertisers can define different types of conversions such as visit websites, fill form, buy products, some click fraud may not be able to apply.For example, simple bots may be applied to generate website visits, but sophisticated designed bots or

human (click farms) are needed in order to fill forms. Some conversion types such as buying advertiser's products are generally hard to be achieved.

- Cookie stuffing
  In *affiliate marketing*, affiliates get paid for each customer he brings to the advertiser. By using cookie stuffing, fraudsters insert multiple (third-party) cookies on the user's browser, in order to claim the conversion attribution.

The actual amount of ad fraud cost is hard to count because ad fraud methods keep evolving so that not all of them are known. However, it has been estimated that around 20% to 40% of impressions are fraudulent [23].

There are several ways to prevent ad fraud. First, advertisers may want to adopt industry standards which reduce the possibility of involving illegal players. For example, the 'Ads.txt' file that was proposed by IAB tech lab [24] is a straightforward solution to avoid domain spoofing fraud. The idea is simply that publishers use this file as a whitelist to decide who can sell ads on their websites. Second, advertisers should look closely at the statistics of their ad campaigns performance. For example, ad campaigns who have a high number of impressions and a low click-though-rate may be affected by stacked and hidden ads. What's more, for non-trivial fraud behaviors (e. g. , making fraud using bot or botnets), it is possible to use data mining and machine learning methods to find the underlying pattern in order to detect them (R. Oentaryo et al., 2014; Blaise et al., 2020; Xing et al., 2021).

1.6 THESIS CONTRIBUTIONS AND RELATED WORK

This dissertation contributes pipelines, models, components, and new insights to problems that arise in the area of online display advertising.

In particular, we focus on developing novel approaches to address two challenging display advertising tasks: conversion prediction and audience expansion. For both of the tasks, we propose to consider only the browsing history (a sequence of URLs) of the user as features, collected from real logged data. Additionally,

- For conversion prediction task, we mainly introduced three self-supervised URL embedding models that can learn semantically meaningful URL representations based on the collected user browsing history.

23. Programmatic ad fraud rates: https://www.statista.com/statistics/778714/programmatic-ad-fraud-rates-worldwide-country/

24. Ads.txt – Authorized Digital Sellers: https://iabtechlab.com/ads-txt/

- For audience expansion task, we proposed various similarity-based audience expansion schemes, focusing on the learning a high-quality user embedding. A data-driven weight factor was also introduced to intentionally alleviate the frequency redundancy of URLs that affected by the website refreshment.

Next, we provide an overview of the contributions of the dissertation, together with related work.

1.6.1 *URL embedding*

Embedding techniques are generally referred to methods that project a term (e. g. , word, product, node, graph) to a vector, while the closeness of the terms are kept in the vector space. Technically speaking, in neural network models, the embeddings are learned through an embedding layer, jointly learned with other layers. The training of the embedding layer can be done in a supervised way or self-supervised way, depending on the task. After the learning process, each term has its own embedding vector which forms a lookup table (matrix) where the row corresponds to the term and the column corresponds to the embedding vector.

The term 'embedding' comes originally from the Word2vec model (Mikolov, K. Chen, et al., 2013; Mikolov, Sutskever, et al., 2013) where the authors try to produce a semantically meaningful vector for each word by using non labeled word sequences (document), based on assumption that the words used in similar contexts share similar meanings (Harris, 1954). The semantic closeness of words is kept in the embedding (vector) space where the word analogy can be deduced from simple mathematical operations, e. g. , : embedding("king") - embedding("queen") $\approx$ embedding("man") - embedding("woman"). A common practice is to use word embeddings that are trained on large unannotated corpus directly on downstream tasks (or slightly fine-tuned based on task-specific data). Results show that this generally improves the performance of the downstream tasks (Mikolov, K. Chen, et al., 2013; Devlin et al., 2018). Later, researchers attempted to apply the same idea to different terms such as documents (Q. Le and Mikolov, 2014), products (Barkan and Koenigstein, 2016; Vasile, Smirnova, and Conneau, 2016), nodes of graph (Grover and Leskovec, 2016), graphs (Narayanan et al., 2017).

URL (*Uniform Resource Locator*), i. e. , the address of a webpage, can be seen as a specific type of term. It is quite diverse, as a small change in the URL string results in a totally different URL. Also, its cardinality is very large which easily reaches billions, and keeps growing as new URLs (e. g. , the URLs of articles in the news website) are generated continuously. Thus, directly using one-hot encoding to represent each URL separately leads to a high-dimensional sparse vector, which can easily cause the curse of dimensionality issue (Bellman, 1957, 1961).

At the same time, different URLs may share the same part of the URL string (e. g. , all webpages of the same website share the same domain). This information is dismissed in a one-hot encoding vector. As a result, how to project each URL to a meaningful dense vector (URL embedding) has attracted researchers' attention recently.

Basically, a URL string can be seen as a sequence of terms where the term here can be either a *character*, a *token* or a *component*, with granularity from high to low.

- URL as a sequence of components
  Generally, one URL is delimited to three components [25]: protocol, domain (host), and path, from left to right. Taking the URL "http://www.exampledomain.net/index.html" for instance, the "http" is its protocol, the "www.exampledomain.net" is its domain and the "index.html" is its path. In the case of the protocol component, it could be either "http" (*Hypertext Transport Protocol*) or "https" (*Hypertext Transfer Protocol Secure*). In general, the domain component can be considered as the main page of the website. In the domain component, the rightmost token (".net" in the example) is the top level domain (*TLD*). The token just before it ("exampledomain" in the example) is the *domain name*. Path is what is left after the domain component, which corresponds to the relative directories of the webpages.
- URL as a sequence of tokens
  In another point of view, a URL can be also seen as a sequence of tokens. How to tokenize the URL depends on the experience and the methods applied. A simple choice could be the splitting of the URL every time where we encounter a special character (e. g. , ":", "/", "."). Thus the aforementioned URL can be tokenized as: ["http", "www", "exampledomain", "net", "index", "html"]. At the same time, as not all the tokens provide distinguishable and sufficient information, a preprocessing step is often applied before feeding them to the model. A common approach is to select tokens based on their frequency: tokens that appear too frequently (similar to stop words) or rarely (noise tokens) are removed. For example, "http", "www", "net" and "html" will be removed as they appear commonly on all URLs, providing little information. Other more sophisticated tokenization methods (Sennrich, Haddow, and Birch, 2016; Y. Wu et al., 2016; Kudo, 2018; Kudo and J. Richardson, 2018) that have been developed and used in the NLP area with the goal to extract informative subwords, may also be applied here.
- URL as a sequence of characters
  URL is considered as a string (a sequence of characters) without special treatment.

25. The components of a URL: https://www.ibm.com/docs/en/cics-ts/5.1?topic=concepts-components-url

Once the decomposition of a URL to terms is decided, the embedding of a URL can be obtained by aggregating the embeddings of its corresponding terms. Different methods that can be applied to compress a sequence of vectors into a single vector, such as: simple average, CNN with max-pooling (H. Le et al., 2018), RNN-based model (Cho et al., 2014; J. Cheng, Dong, and Lapata, 2016). The existing URL embedding methods in literature are generally a combination of the *URL-to-term decomposition* and *term embeddings aggregation*. For instance, H. Le et al., 2018 consider a URL as a sequence of characters and a sequence of tokens (referred to "words" in the article) at the same time. A CNN layer with max-pooling is applied on top of each sequence and concatenated to construct the final embedding of the URL. T. T. T. Pham, Hoang, and Ha, 2018 simply consider each URL as a sequence of characters and add a CNN or LSTM on top of it. H. Yuan et al., 2018 consider a URL as a sequence of five components (protocol, sub domain name, domain name, domain suffix, URL path). For each character, a skip-gram model is applied to learn their embeddings. Then, the embedding of each component is considered as the average of all the embedding vectors of its characters.

OUR CONTRIBUTION Although the aforementioned approaches have good performance in general, their applications and data are still limited in the domain of malicious URL detection while the URL embeddings are mostly learned in a supervised way. Inspired by Word2vec (Mikolov, K. Chen, et al., 2013) models, in our work (Qiu et al., 2020; Tziortziotis et al., 2021), we learn the URLs embeddings in a self-supervised way, using user browsing history data collected from RTB campaigns. Here, user browsing history is analogous to document and URL is analogous to word. As for the URL-to-term decomposition, we simply remove the protocol component ("http://" or "https://" part) of the URL and split the rest part by "/" to construct tokens. The embedding of URL is obtained by aggregating the embeddings of its tokens, proposed in three different ways. `Domain_only` uses the embedding of the domain token as URL embedding, while `Token_avg` (resp. `Token_concat`) averages (resp. concatenates) all the three token embeddings to construct URL embedding.

1.6.2 *User response prediction*

As we have seen previously, in performance-driven RTB campaigns, advertisers generally want to control their cost for each acquisition of positive user response (e. g. , click, conversion, etc.). For example, cost-per-action (resp. cost-per-click) is used as the maximum price where they are willing to pay in average for each generated conversion (resp. click). A representative of an advertiser (e. g. , DSP, agency, etc.) typically uses cost-per-action (CPA) or cost-per-click (CPC) to guide

his bid price of each impression, depending on the ad campaign's objective. Actually, the expected value of one specific impression ($CPM/1000$) can be decomposed to $CPA \times CVR$ (see Eq. 1.1, Sec. 1.3) or $CPC \times CTR$ where CVR (conversion rate) and CTR (click-through rate) represent the probability that this impression will lead to a conversion and a click, respectively. Thus, an accurate estimation of CVR or CTR is required to provide a reasonable value for the present impression opportunity. This value is important for the following bidding strategy because the bidder generally needs to know it in order to offer the bid accordingly [26] (e. g. , in second price auction, the optimal strategy is to bid truthfully at a price equal to the value of the impression). Too optimistic estimation of this value causes budget waste and reduces the number of impressions one can buy, while a too passive estimation may lead to no winning bids and miss ad impressions likely to lead to user actions.

In the following, we present a number of related works about the CTR and CVR estimation, respectively. Typically, supervised models are used here where each impression that leads to click/conversion is labeled as positive and the rest are labeled as negative. The main features used in the models are the features of impressions, as already presented in Sec. 1.5. Additional features can be also added. For example, one can consider the number of times that the user has seen the ad from this advertiser as a feature.

CTR ESTIMATION The goal of a CTR estimator is to estimate the probability that a user will click or not on the displayed ad of the advertiser. At the beginning, classical shallow models such as Logistic Regression (M. Richardson, Dominowska, and Ragno, 2007; Olivier Chapelle, Eren Manavoglu, and Romer Rosales, 2015), and Decision Tree models (J. H. Friedman, 2001; X. He et al., 2014) have been proposed. Early attempts of model improvement focus on two main issues: scalability and dataset imbalance.

- Scalability
  In RTB, the buyer needs to decide the bid price in less than 100 milliseconds. Thus the model needs to be highly scalable in order to give an estimation of the CTR in real time. Due to high cardinality of categorical features such as URL, user-agent, directly using one-hot encoding can cause curse of dimensionality issues (Bellman, 1957, 1961). Hashing trick (Weinberger et al., 2009) is commonly adopted (Olivier Chapelle, Eren Manavoglu, and Romer Rosales, 2015; Juan et al., 2016) as a simple and easy to implement solution in order to reduce the feature dimension.
- Dataset imbalance
  In practice, the CTR is generally less than 1%, which means that

26. In practice, for DSP, multiple ads are considered together. DSP often chooses the one with the highest value to bid on.

the data is highly imbalanced, with way less positive samples than negative ones. Researchers have developed many techniques to deal with class imbalance, where two most popular categories of solutions are the *data-level* and the *algorithm-level* approaches (Leevy et al., 2018). In the case of the data-level approaches, data sampling strategies are adopted where either the major class is down-sampled (Wilson, 1972; Laurikkala, 2001; J. Zhang and Mani, 2003) or the minor class is over-sampled (Chawla et al., 2002; Han, W.-Y. Wang, and Mao, 2005; H. He et al., 2008). For the algorithm-level approaches, cost-sensitive learning techniques (Elkan, 2001; X.-y. Liu and Z.-h. Zhou, 2006) are commonly adopted here, where we give different weights at each class during the learning. For instance, a simple but common practice is to give each class a weight that is reversed to its portion in the data, so that minor (resp. major) class samples get more (resp. less) importance.

Due to the aforementioned issues, linear models such as Logistic Regression (LR) are commonly used in practice (King and Zeng, 2001; Szwabe, Misiorek, and Ciesielczyk, 2017). Factorization machines (Rendle, 2010, FM) and its extensions (Juan et al., 2016; Pan et al., 2018) improve the LR model by taking also the interaction between features into account. Specifically, in the case of FM, each feature is assigned with a learnable embedding vector. The interaction between two features is modeled as the inner product of their corresponding feature embedding vectors.

As neural network (NN) models showed their effectiveness in various tasks, researchers started to investigate and use them in the CTR task. Instead of directly using sparse vectors to represent categorical features, NN models learn dense vector representations (embedding) of them. Then, these feature vectors are fed to Deep Neural Networks (DNN, e. g. , multi-layer perceptron) to model high-order feature interactions (H.-T. Cheng et al., 2016). Typically it is done by applying a *feature interaction layer* (Weinan Zhang et al., 2021). For example, the feature interactions in the FM model can be seen as a specific case of an Inner Product-based Neural Network (*IPNN*) or an Outer Product-based Neural Network (*OPNN*) (Yanru Qu et al., 2016). Convolution Neural Networks (Krizhevsky, Sutskever, and Geoffrey E Hinton, 2012; Q. Liu et al., 2015; Bin Liu et al., 2019) and Attention Mechanism (Vaswani et al., 2017; Xiao et al., 2017; G. Zhou et al., 2018) have been also used to model feature interactions.

However, stacking DNN directly on top of the feature interaction layer (refereed as *single tower* model (Weinan Zhang et al., 2021)) is possible to loss low-order feature interactions information. To overcome this weakness, *dual tower* models that put the feature interaction layer and the DNN in parallel have been used. *Wide & Deep network* (H.-T. Cheng et al., 2016) is one of the earliest attempts that add DNN

(referred as *deep* part) along with a standard LR model (referred as *wide* part). Guo et al., 2017 use a FM as the wide part, while the FM and DNN share the same feature embedding layer that is learned in an end-to-end way. Recently, Y. Cheng and Xue, 2021 propose the usage of a Discrete Choice Model (*DCM*, Train, 2009) to redefine CTR prediction problem, showing that most of the existing CTR prediction models can be included into a general architecture.

It is worth mentioning that click data is usually noisy and may even contain fraud (see Sec. 1.5). Therefore, it does not mean that a click represents positive user feedback for sure. The detection of this 'false' information has attracted the attention of research community recently (R. Oentaryo et al., 2014; Tolomei et al., 2018; C. Wu et al., 2021).

CVR ESTIMATION The main objective of a CVR estimator is to estimate the probability that the displayed impression of one advertiser leads to a conversion such as visiting the advertiser's website, buy advertiser's product, etc. To a certain degree, the CVR task is very similar to the CTR task. They share the same impression data, while only the label changes. Dataset imbalance issue is more severe in the CVR task as conversion is generally rarer than clicks. That's to say, the techniques and models that deal with CTR tasks can generally be adopted in CVR tasks. The major difference between CVR and CTR task is that **click feedback is instantaneous, but conversion feedback has a delay**. For example, a user may purchase an advertiser's product several days after he saw the advertiser's ads. To define whether a conversion is happening due to a displayed ad, a *conversion window* is set and used by advertiser. The length of the conversion window is variable and depends on the type of the conversion. Roughly speaking, a conversion window of 30 days means that starting from the time at which an impression is displayed, if a conversion happens during the next 30 days, it's possible that this conversion be attributed to this impression. Any conversion out of this window will be ignored.

A major difficulty caused by delayed feedback is that it causes "false negative" samples. A negative impression sample that has not led to a conversion yet may lead to a conversion later. Common solutions share a similar schema where the CVR model is decomposed to two models: i) a *conversion model* to predict whether a conversion will arrive or not, and ii) a *time delay model* to predict the conversion delay time. O. Chapelle, 2014 uses generalized linear models for both conversion and time delay models where an exponential distribution is used to model the delay time. Yoshikawa and Imai, 2018 argue that non-parametric models should be used to model the delay time since the delay time distribution may have various shapes depending on the context of the ad and the user. In order to better capture the information in the user feedback sequence, Y. Su et al., 2020 applie deep learning structures such as GRU (Cho et al., 2014), attention mechanism (Vaswani et al.,

2017) in both models. More details about CTR and CVR modeling can be found at the next survey papers: Gharibshah and Xingquan Zhu, 2021; Weinan Zhang et al., 2021.

OUR CONTRIBUTION In our work (Qiu et al., 2020), we investigate a conversion prediction problem where our objective is to predict if a user will convert or not the next day. We consider a user as converted if he has visited the advertiser's website. In contrast to other related works that use a combination of features related to the user profile (e. g. , ad information and website information), we consider only the **user's browsing history**. Specifically, user's browsing history is collected from bid request data, in a single day. Using user's browsing history is end-user friendly as it doesn't contain any other information (e. g. , user-agent, ip address, etc.) that could be used in order to identify the user. Another advantage of using a user's browsing history is that it typically reflects the interests of the user. Due to the high cardinality and diversity of URLs, the direct use of sparse transformations, such as the one-hot encoding, is not practical. Following the idea of Word2vec models (Mikolov, K. Chen, et al., 2013), we learn dense URL representations (URL embeddings, see details in Sec. 1.6.1). Having computed the URL embeddings, an aggregation layer is added on top of the URL embedding layer along with a dense layer that estimates the conversion probability of the user. The output of the aggregation layer can be considered as the user representation. Actually, we propose three different variants of the aggregation layer: average (averaging all the URL embeddings in the sequence), dense (a dense layer with same dimension of URL embedding), LSTM (a LSTM layer where the hidden state dimension is the same with URL embedding). In total, we have introduced ten different prediction conversion models. To evaluate the effectiveness of the proposed prediction conversion models, we have conducted large scale experiments on real log data collected from ad campaigns. The empirical results show that our representation models are able to group together URLs of the same category, and user browsing history alone is useful to predict users' visit on the advertiser's website.

1.6.3 *Audience expansion*

In online advertising, advertisers' goal is to deliver advertisements about their products or services to the right audiences (users), in order to increase their profit. The right audience here refers to the existing customers or the potential ones. *Audience expansion*, also known as *look-alike targeting*, is a popular strategy used to discover potential customers for the advertisers. By applying an audience expansion advertising strategy, the advertiser first provides a set of *seed users* who have already shown their interest in advertiser's products or services

(e. g. , converted users). Then, the agency (DSP) helps the advertiser to discover more users with similar interests (browsing behavior). The data used in the general audience expansion problem depends on what the agency (DSP) has available on his side. When no specific criteria is given by the advertiser, user behavior data are used as they are highly related to the user's ad responses. The main difficulties of an audience expansion task are:

- Seed users selection criteria
  Theoretically, advertisers are allowed to give any kind of seed users as they prefer, where the selection criteria of the seed users is not always clear and may even be unrevealed [27]. Consequently, the seed user set may contain a lot of noise, such as outlier users or multiple subgroups of users.
- Satisfaction of multiple objectives
  The expanded audiences often need to satisfy multiple objectives or constraints defined by the advertiser. For example, the agency or DSP needs to balance the number of expanded audiences (should be sufficiently large in order to have enough reach) and the budget constraint. Meanwhile, the expanded audiences are desired to have comparable performance (in terms of generated CTR, CVR, etc.) to the seed users.
- Scalability
  In online advertising ad campaigns, millions of users need to be examined, under a restricted runtime (minutes, or even seconds in real-time). Thus, the applied audience expansion methods need to be highly scalable in order to fulfill the needs of the continuously created ad campaigns.

The audience expansion problem can be treated in a supervised or unsupervised way. The general idea behind the supervised approaches (referred also as *regression-based*) is to consider the seed users as positive samples and the rest users as negative samples. Then, a classifier is trained using these labeled samples, trying to distinguish the positive from the negative users. However, it's worth mentioning that not all non-seed users in the user pool are negative users. How to carefully select true negative users from non-seed users is important for the performance of the learned classifier. This problem is known as positive-unlabeled learning (*PU learning*) problem (Bekker and Davis, 2020). For example, Jiang et al., 2019 examine multiple sampling strategies (Bing Liu et al., 2003; Mordelet and Vert, 2014) for selecting reliable negative samples from non-seed users.

In the case of unsupervised audience expansion approaches (known also as *similarity-based*), we focus on selecting a suitable metric to measure the similarity between users. Then, we can select the top-K closest users to the seed users as the expanded audience set. Method-

27. In practice, advertisers commonly provide converted users as seed users.

ologies of these two categories (*regression-based* and *similarity-based*) are often combined together (deWet and Ou, 2019) to achieve better performance.

Next, we briefly present existing **similarity-based** audience expansion methods applied in industry. For similarity-based models, without the need of specific treatment for the features, Locality-Sensitive Hashing (*LSH*, Slaney and Casey, 2008) technique is widely used in production (H. Liu et al., 2016; Q. Ma et al., 2016; Qiang Ma et al., 2016; deWet and Ou, 2019) to efficiently calculate the similarity between user. By using proper hashing functions, it puts similar inputs (users in our case) together (in same the 'bucket') with high probability. Due to its low computational cost, it can scale up to handle hundreds of millions of users (deWet and Ou, 2019).

At the same time, learning high quality embedding of feature helps to uplift the model's performance, as proven in the success of various embedding methods (Mikolov, K. Chen, et al., 2013; Pennington, Socher, and C. Manning, 2014; Devlin et al., 2018; Peters et al., 2018). A two-stage model structure is commonly adopted in practice. At the first stage, users embeddings are learned (usually offline) either in a supervised or self-supervised way. At the second stage, the model compute the similarity score of each user to its closest seed users and output the top-K users as expanded user set. For instance, deWet and Ou, 2019 propose to learn users embeddings based on the user-item (user-Pin) interaction data collected by Pinterest. Then, LSH is applied on the learned users embeddings to project similar users to the same region ('bucket'). This process is repeated for multiple times, where the density of seed users in each region is calculated each time. The affinity score of one candidate user to the seed user set is calculated as the average of the seed user density in his bucket.

Y. Liu et al., 2019 work on a news feed recommendation system for Wechat, where latest/hot/quality articles are considered as seed set (it is updated continuously). First, the Youtube DNN model (Covington, Adams, and Sargin, 2016) has been adopted to learn user embedding, which is a stack of an embedding, an average pooling (for features of the same field), a concatenation and a multiple layer perceptron (MLP) layer. The output of the MLP layer is considered as the user representation. In order to handle heterogeneous and multi-fields features, the concatenation layer has been replaced with an attention merge layer. Then, a supervised look-alike model based on attention and clustering algorithm (k-means, Hartigan and Wong, 1979) is used to predict a candidate user's label. The main reason for using k-means is for reducing the computational cost that is necessary for delivering real-time recommendations.

Meanwhile, as mentioned previously, the seed user set may contain a lot of noise. To tackle the issue, Zhuang et al., 2020 propose a two-stage audience expansion system (named *Hubble*) that uses the

well-designed *knowledge distillation* mechanism (G. Hinton, Vinyals, and Dean, 2015), to eliminate the *coverage bias* (the gap between the seeds and the actual audiences) introduced by the provided seed set. Specifically, at the first stage, user embeddings are learned based on the user-campaign bipartite graph by using an adaptive and disentangled graph neural network (*AD-GNN*). Next, besides the hard label that indicates if a user is a seed user or not, a softened label is created for each user with the help of the knowledge obtained by the AD-GNN (i. e. , through knowledge distillation). A classifier is then learned by considering both the hard and the softened label. In this way, the coverage bias of seed users is alleviated.

OUR CONTRIBUTION In our work (Tziortziotis et al., 2021), we have proposed and examined different similarity-based audience expansion schemes based on users browsing history data, where the seed users are the converted users. Thus, ideally, the expanded uses are users who share similar browsing interests with the seed users. Specifically, the proposed audience expansion models are generally two-stage models, where users representations are learned at the first stage and user similarity comparison is executed at the second one.

To learn users embeddings, we have adopted various NLP techniques where each **user** (represented by his browsing history) is analogous to a **document** (a sequence of words) and each visited **URL** is analogous to **word**. Actually, we have proposed *User2Vec* and *User2VecC* models which borrow the idea of document embedding techniques *Doc2Vec* (Q. Le and Mikolov, 2014) and *Doc2VecC* (M. Chen, 2017), respectively. These two techniques learn document embeddings in a self-supervised way. Then, user similarity comparison is done pair-wisely by using cosine or euclidean similarity metrics. Beside projecting each user to a single vector, we also examine methods that compare directly the users by considering them as sequences of URLs (Jaccard, 1912; S. Ioffe, 2010; Kusner et al., 2015). We have also proposed a data-driven weight for each URL which is inspired by the inverse document frequency (*IDF*, Sparck Jones, 1988), in order to alleviate the frequency redundancy of URLs that comes by website refreshment. Additionally, we have examined the effect of seed user filtering by removing users whose browsing history is dominated by few URLs. Our experiments on real world data have shown that the Continuous Bag of Words (CBOW) version of User2VecC model, combined with IDF transformation and euclidean similarity, is the best choice among the proposed audience expansion schemes. Last, we have seen that the seed user filtering is of high importance for achieving a good model performance.

1.7 THESIS ORGANIZATION

The remainder of the dissertation is organized as follows. Chapter 2 presents some preliminaries and introduces some general machine learning concepts, such as neural network models, popular word embedding techniques, etc. The next two chapters (Chapter 3 and 4) present in detail the methodologies proposed in this thesis that try to solve two challenging tasks in the area of display advertising. Specifically, Chapter 3 presents our work on the user conversion prediction task where we predict user conversion based on user browsing history data. One of the main novelties of this work is the learning of URL embeddings in a self-supervised way using only the user's browsing history. Chapter 4 presents our work on the audience expansion task using user browsing histories. In this work, we propose and examine different similarity-based audience expansion schemes, focusing on learning a high-quality user embedding. Finally, Chapter 5 concludes the dissertation and gives further promising research directions.

2

PRELIMINARIES AND BACKGROUND

In this chapter, we introduce necessary preliminary knowledge concerning this dissertation. We first introduce basic concepts and notations (Section 2.1) that we need in order to understand this chapter (algebra, optimization, etc). Then we briefly introduce the main concepts of machine learning (Section 2.1) including its categorization (Section 2.2.1), the training (Section 2.2.2) and the evaluation procedure (including evaluation protocol in Section 2.2.5 and evaluation metrics in Section 2.2.6), underfitting and overfitting issues (Section 2.2.3) and how to prevent overfitting through regularization (Section 2.2.4). After, we introduce a specific type of machine learning model, the neural networks (Section 2.3), which achieves state-of-the-art results in various domains such as computer vision, natural language processing, AI game playing where standard machine learning models fail to get close performance. Finally, we present models that can learn dense and meaningful representations of words (Section 2.4) and documents (Section 2.5) from unannotated corpus, with main interest in methods that use neural networks to learn those representations in a self-supervised way, because of their impressive performance in downstream tasks.

2.1 BASIC MATH

2.1.1 *Algebra*

VECTOR All the vectors in this dissertation are by default column vectors. A p-dimensional vector $\boldsymbol{v} \in \mathbb{R}^p$ is denoted as $\begin{bmatrix} v_1 \\ \vdots \\ v_p \end{bmatrix}$ or $[v_1, \cdots, v_p]^T$ where T is the transpose operator.

EIGENVALUE AND EIGENVECTOR A non-zero vector $\boldsymbol{v}$ of dimension n is an *eigenvector* of a square $n \times n$ matrix $\boldsymbol{A}$ if it satisfies the linear equation $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$ where λ is a scalar, referred as the *eigenvalue* corresponding to $\boldsymbol{v}$.

ORTHOGONAL MATRIX A $p \times p$ real square matrix $\boldsymbol{A}$ is an *orthogonal matrix* iff. $\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$ where $\boldsymbol{I}$ is identity matrix. Its columns and rows are orthogonal vectors.

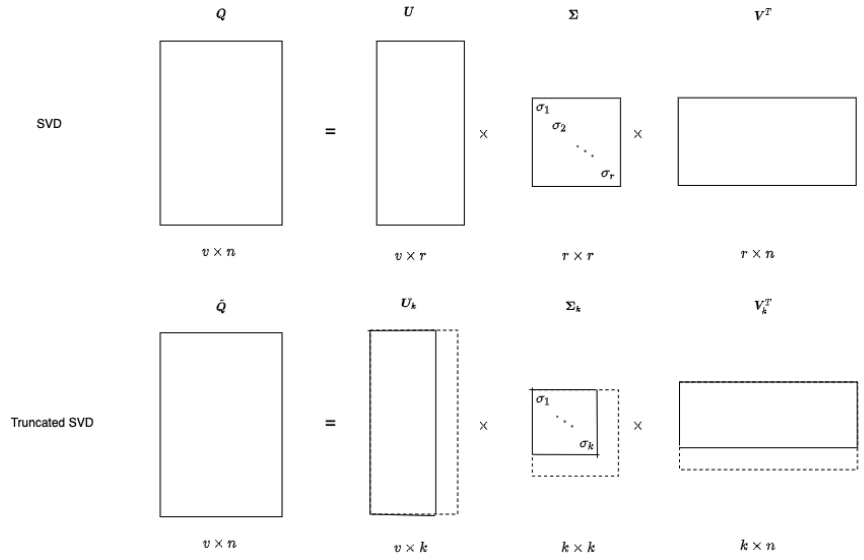


Figure 2.1 – Overview of SVD and Truncated SVD on term-document matrix Q of rank r. For Truncated SVD, the selected part are marked with solid lines.

POSITIVE-DEFINITE MATRIX A $p \times p$ symmetric real matrix $\boldsymbol{A}$ is said to be *positive-definite* iff. $\boldsymbol{v}^\top \boldsymbol{A}\boldsymbol{v} > 0$ for all non zero vector $\boldsymbol{v} \in \mathbb{R}^p$.

SVD Matrix factorization methods decompose a matrix as a product of matrices where new matrices are often with useful properties (e. g. , orthogonal, triangle). It facilitates future operations and latent factors discovery (e. g. , word-document co-occurrence matrix is decomposed to word-topic, topic-document matrices). *Singular Value Decomposition* (*SVD*, Golub and Reinsch, 1970) is the widest used matrix factorization method, it decomposes a given matrix $\boldsymbol{Q}$ of size $v \times n$ and rank r ($r \leqslant \min(v, , n)$) in the following way:

$$\boldsymbol{Q} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top, \tag{2.1}$$

where $\boldsymbol{U}$, $\boldsymbol{V}$ are orthogonal matrices of size $v \times r$, $n \times r$ respectively and $\boldsymbol{\Sigma}$ is a diagonal matrix of size $r \times r$. More specifically, $\boldsymbol{\Sigma} = diag(\sigma_1, \ldots, \sigma_r)$ where $\{\sigma_i\}$ are known as *singular values* of $\boldsymbol{Q}$, sorted in descending order ($\sigma_1 \geqslant \sigma_2 \cdots \geqslant \sigma_r$).

Some practical applications such as data compression, denoising, search for the best low-rank approximation $\tilde{\boldsymbol{Q}}$ of $\boldsymbol{Q}$, where the objective is to minimize the *Frobenius norm* between $\tilde{\boldsymbol{Q}}$ and $\boldsymbol{Q}$ (i. e. , $\|\tilde{\boldsymbol{Q}} - \boldsymbol{Q}\|_F = \sqrt{\sum_i \sum_j |\tilde{Q}_{i,j} - Q_{ij}|^2}$), under the constraint that $rank(\tilde{\boldsymbol{Q}}) = k < r$. Eckart–Young–Mirsky theorem (Eckart and Young, 1936) proves that this problem has an analytic solution of the form:

$$\tilde{\boldsymbol{Q}} = \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^T, \tag{2.2}$$

where $\boldsymbol{U}_k$,$\boldsymbol{\Sigma}_K$ and $\boldsymbol{V}_k$ are partitions of $\boldsymbol{U}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$ respectively, corresponding to $\boldsymbol{Q}'s$ k-largest singular values (See Figure 2.1). This

approach is known as *Truncated SVD*. In practice, k is chosen so that most information (often 90%) in $\boldsymbol{Q}$ is kept, i.e., $\sum_{i=1}^{k} \sigma_i \approx 0.9 \sum_{i=1}^{r} \sigma_i$. Truncated SVD is often served as an efficient dimension reduction method because $k \ll \min(n, v)$ when $\boldsymbol{Q}$ is sparse.

CONVOLUTION OPERATOR Convolution operator $*$ takes two functions g_1 and g_2 as input, and outputs a third function $g_1 * g_2$ which describes how the shape of g_1 is affected by g_2, formulated as follows:

$$(g_1 * g_2)(\boldsymbol{t}) = \int_{\boldsymbol{\tau}} g_1(\boldsymbol{\tau}) g_2(\boldsymbol{t} - \boldsymbol{\tau}) d\boldsymbol{\tau}, \boldsymbol{t} \in \mathbb{R}^p. \tag{2.3}$$

Its discrete version, where g_1, g_2 are both discrete functions defined on $\mathbb{Z}^d$, is formulated as:

$$(g_1 * g_2)(\boldsymbol{t}) = \sum_{\boldsymbol{\tau}} g_1(\boldsymbol{\tau}) g_2(\boldsymbol{t} - \boldsymbol{\tau}), \boldsymbol{t} \in \mathbb{Z}^p. \tag{2.4}$$

Convolution operator is widely used in signal processing tasks to examine frequency peaks of an input signal by taking the convolution of the original signal with sinusoidal functions of different frequencies. Additionally, by reflecting one of the two input functions on y-axis, we obtain a close related operator named *cross-correlation*, where its discrete version is defined as:

$$(g_1 \star g_2)(\boldsymbol{t}) = \sum_{\boldsymbol{\tau}} g_1(\boldsymbol{\tau}) g_2(\boldsymbol{t} + \boldsymbol{\tau}), \boldsymbol{t} \in \mathbb{Z}^p. \tag{2.5}$$

2.1.2 *Optimization*

PARTIAL DERIVATIVE Given a function $\mathcal{L} : \mathbb{R}^q \to \mathbb{R}$ of variable $\boldsymbol{w} = [w_1, \cdots, w_q]^\top$, its *partial derivative* with respect to w_i is defined as

$$\frac{\partial}{\partial w_i} \mathcal{L}(\boldsymbol{w}) = \lim_{h \to 0} \frac{\mathcal{L}(w_1, \ldots, w_i + h, \ldots, w_q) - \mathcal{L}(w_1, \ldots, w_i, \ldots, w_q)}{h} \tag{2.6}$$

which measures how $\mathcal{L}$ changes along the i-th dimension of its variable $\boldsymbol{w}$.

GRADIENT Given a function $\mathcal{L} : \mathbb{R}^q \to \mathbb{R}$ of variable $\boldsymbol{w}$, its *gradient* is defined as

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = \left[\frac{\partial}{\partial w_1} \mathcal{L}(\boldsymbol{w}), \cdots, \frac{\partial}{\partial w_p} \mathcal{L}(\boldsymbol{w}) \right]^\top \tag{2.7}$$

As a fact, $-\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ is the direction where $\mathcal{L}(\boldsymbol{w})$ decreases the fastest.

OPTIMIZATION AND OPTIMIZER Optimization is the process of using the algorithms called *optimizers* to either maximize or minimize an *objective function* $\mathcal{L} : \mathbb{R}^q \to \mathbb{R}$ on a feasible set $\mathcal{F} \subset \mathbb{R}^q$. As

maximizing $\mathcal{L}$ equals to minimizing $-\mathcal{L}$ and vice versa, we only present the minimization case in the following discussion.

We first introduce the unconstrained optimization case where $\mathcal{F} = \mathbb{R}^q$. In this case, we are interested to find $\hat{\boldsymbol{w}}$ that minimize $\mathcal{L}(\boldsymbol{w})$, i. e. , $\hat{\boldsymbol{w}} = \text{argmin}_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$. Supposing $\mathcal{L}$ is smooth enough so that its gradient exists everywhere, then for each $\boldsymbol{w}$ where $\mathcal{L}(\boldsymbol{w})$ reaches a local minimum, we have

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = 0. \tag{2.8}$$

Otherwise, we can move $\boldsymbol{w}$ along the opposite direction of the non-zero valued partial derivative, $\mathcal{L}(\boldsymbol{w})$ will still decrease. Let's consider a simple situation first where only one minimum exists, i. e. , local minimum equals to global minimum. In this case, if the analytic solution of $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = 0$ is possible, we can find the global minimum point easily by solving this equation. If analytic solution is not possible, which is often the case, *gradient descent-based* solutions are adopted. They are based on the fact that the opposite direction of the gradient $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})$ is where $\mathcal{L}$ decreases the fastest. In a nutshell, we randomly select a starting point $\boldsymbol{w}_0$, we then let it move along the opposite direction of its gradient with small step size, we repeat this process until we reach a point $\hat{\boldsymbol{w}}$ where the gradient is 0 (in practice, we only require the gradient to be small enough due to computational limits), i. e. , reaches the global minimum point.

For the more general constrained case where the feasible set $\mathcal{F} \subset \mathbb{R}^q$, a generic framework which includes the most popular *iterative gradient decent optimizers* is provided by Reddi, Kale, and Kumar, 2019. Within this framework, *the learning rate* applied on the optimizer is changed adaptively in each iteration, under an online optimization setting. Roughly speaking, following this family of gradient decent optimizers, we can generally approach to the minimum point of an objective function $\mathcal{L}(\boldsymbol{w})$ in a convergence speed that is linearly correlated with the total iteration rounds T. From now on, we follow the setting in their work to introduce this framework, with slight change of the notation. We denote $\boldsymbol{\mathcal{S}}_q^+$ as the set of all $q \times q$ positive definite matrices; $\mathcal{L}(\boldsymbol{w})$ as the empirical risk (see Section 2.2.2) of training samples with parameter $\boldsymbol{w}$, which represents the objective function to be minimized; $\mathcal{L}_t(\boldsymbol{w})$ as the empirical risk of the t-th mini-batch. We denote the projection of any point $\boldsymbol{w}'$ onto $\mathcal{F}$ as $\Pi_{\mathcal{F}}(\boldsymbol{w}') = \text{argmin}_{\boldsymbol{w} \in \mathcal{F}} \|\boldsymbol{w}' - \boldsymbol{w}\|_2$, the projection of $\boldsymbol{w}'$ onto $\mathcal{F}$ with a positive definite matrix $\boldsymbol{A}$ as $\Pi_{\mathcal{F},\boldsymbol{A}}(\boldsymbol{w}') = \text{argmin}_{\boldsymbol{w} \in \mathcal{F}} \|\boldsymbol{A}^{\frac{1}{2}}(\boldsymbol{w}' - \boldsymbol{w})\|_2$. The generic framework is described in Algorithm 1 (Reddi, Kale, and Kumar, 2019), where α_t is referred as the *step size* and $\alpha_t V_t^{-\frac{1}{2}}$ as the *learning rate* of the algorithm, $\phi_t : \mathcal{F}^t \to \mathbb{R}^q$ and $\psi_t : \mathcal{F}^t \to \boldsymbol{\mathcal{S}}_q^+$ are two aggregation functions. For each iteration t, we update the present point $\boldsymbol{w}_t$ following the direction of $-\nabla \mathcal{L}_t(\boldsymbol{w}_t)$, with a step size α_t decreasing with the iteration t.

Algorithm 1: Generic framework for most popular iterative gradient descent algorithms

Input: $\boldsymbol{w}_1 \in \mathcal{F}$, step size $\{\alpha_t > 0\}_{t=1}^{T}$, sequence of functions $\{\phi_t, \psi_t\}_{t=1}^{T}$;
for $t = 1$ *to* T **do**
$\quad \boldsymbol{g}_t = \nabla_{\boldsymbol{w}} \mathcal{L}_t(\boldsymbol{w}_t)$
$\quad \boldsymbol{m}_t = \phi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t)$
$\quad \boldsymbol{V}_t = \psi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t)$
$\quad \boldsymbol{w}'_{t+1} = \boldsymbol{w}_t - \alpha_t \boldsymbol{V}_t^{-\frac{1}{2}} \boldsymbol{m}_t$
$\quad \boldsymbol{w}_{t+1} = \Pi_{\mathcal{F}, \boldsymbol{V}_t^{\frac{1}{2}}}\left(\boldsymbol{w}'_{t+1}\right)$
end

The classical (mini-batch) *Stochastic Gradient Descent* (*SGD*, Robbins and Monro, 1951) algorithm belongs to the case where

$$\phi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t) = \boldsymbol{g}_t, \psi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t) = \boldsymbol{I}, \alpha_t = \alpha/\sqrt{t}. \tag{2.9}$$

2.1.3 *Activation functions*

Here, we present some basic activation functions (*sigmoid, hyperbolic tangent, softmax, ReLU*) used in the models presented in this dissertation.

SIGMOID The *sigmoid* activation function σ takes a scalar z as input, transforms it into a value between 0 and 1 (interpreted as probability), defined as follows:

$$\sigma(z) = \frac{1}{1+e^z}, z \in \mathbb{R} \tag{2.10}$$

It is often served as the last layer of a binary classification model (e. g. , Logistic Regression) where it outputs the predicted probability for the positive class.

HYPERBOLIC TANGENT FUNCTION *Hyperbolic tangent function* (*Tanh*) takes a scalar z as input, transforms it into a value between -1 and 1, defined as follows:

$$\tanh(z) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}, z \in \mathbb{R} \tag{2.11}$$

SOFTMAX The *softmax* function takes a vector $\boldsymbol{z} \in \mathbb{R}^k$ as input, normalizes it into a probability distribution $\boldsymbol{p}$ proportional to the exponentiation of $\boldsymbol{z}$, i. e. ,

$$p_i = softmax(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, \text{ for } i = 1, \ldots, k. \tag{2.12}$$

It's an extension of sigmoid activation function for k-class classification model where each position of the softmax's output is the predicted probabilities for each class.

RELU The *Rectified Linear Unit (ReLU)* activation function (Nair and Geoffrey E. Hinton, 2010) takes a scalar z as input, returns z if it is positive and 0 otherwise. i. e. ,

$$ReLU(z) = z^+ = max(0, z), z \in \mathbb{R} \tag{2.13}$$

The advantages of using ReLU are that the function and its gradient are extremely easy to compute, which reduces the computation time comparing to using sigmoid. Also, it introduces sparsity as negative input are squashed to 0.

2.2 MACHINE LEARNING BASICS

Machine learning (ML) is often considered as a measure to realize *Artificial Intelligence (AI)*. It focus on designing computer *models* (*algorithms*) which automatically **learn** from **data** to solve specific tasks. Compared to rule-based models which require large amount of human experts' investigation for each encountered problem, learning-based models are typically adapted to a type of problems and require only enough data with moderate human supervision. ML is the core of various applications such as spam detection (Dada et al., 2019), image classification (Krizhevsky, Sutskever, and Geoffrey E Hinton, 2012; K. He et al., 2015), product recommendation (Linden, Smith, and York, 2003; Q. Chen et al., 2019), language translation (Bahdanau, Cho, and Yoshua Bengio, 2016; Johnson et al., 2016; Y. Wu et al., 2016), autonomous driving (Grigorescu et al., 2019). Taking the spam detection task as an example, the objective is to design an algorithm that automatically identify whether an email is a spam or not. The main questions to be answered here are: How the model learns from data? Can we confirm the model has learned something? What is the actual performance of the learned model? In the following sections, we introduce the main concepts in machine learning by trying to answer these questions.

2.2.1 *Machine learning categorization*

In machine learning domain, the data is typically presented in form of *dataset*, which is a collection of *samples (examples)*. As computers can only take numeric inputs, each sample is typically represented by a multi-dimensional tensor where each entry represents a qualitative or quantitative measure of its characteristic (*feature*). Without loss of generality, we suppose each input is a vector, denoted as $\boldsymbol{x} \in \mathbb{R}^p$ where p is the dimension of the feature space. A dataset with n

examples is denoted as a set $\{\boldsymbol{x}_i\}_{i=1}^n$ or a matrix $\boldsymbol{X} \in \mathbb{R}^{p \times n}$ as the feature dimension p is the same for all samples.

In the spam detection task, for example, we can represent an email using two features: the number of special characters in the email and the domain of the sender's email address. An email which contains 100 special characters and sent from *xxx@gmail.com* can be represented by a 2-dimensional vector $[100, 4]^\top$ where 100 is the number of special characters in the email and 4 is the index of *gmail.com* in the vocabulary of domains.

Depending on the types of data that machine learning algorithms are allowed to use, most of them can be included in two categories (Hastie, Robert Tibshirani, and J. Friedman, 2001; Bishop, 2006): *supervised learning* and *unsupervised learning*.

SUPERVISED LEARNING Given an annotated dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, the goal of supervised learning is to learn how to correctly predict y knowing $\boldsymbol{x}$. The y is referred as *target*, with a role to guide and supervise the learning process. Actually, depending on the type of the target y, we can generally separate supervised learning tasks to *regression* and *classification*. In regression tasks, y is a scalar, i. e. , $y \in \mathbb{R}$. For example, predicting the salary of an employee given his gender, years of experience and job title is a regression task. In classification tasks, y is a *class* (*category*), i. e. , $y \in \{0, \dots, k-1\}$, $k \geqslant 2$ where k is the number of predefined classes. In practice, classification can be further divided into three settings: 1) *binary* classification: each example can only be labeled with one of two classes ($k = 2$, e. g. , spam or non-spam), 2) *multi-class* classification: similar to the binary one, but with more possible classes ($k > 2$), 3) *multi-label* classification: each example can belong to multiple classes at the same time ($y \in \{0, 1\}^k$).

UNSUPERVISED LEARNING For unsupervised learning, we are given an unannotated dataset $\{\boldsymbol{x}_i\}_{i=1}^n$. Without the guidance of the labels $\{y_i\}_{i=1}^n$, the goal is to learn the underlying properties or structures of $\{\boldsymbol{x}_i\}_{i=1}^n$ (Pearson, 1901; Goodfellow et al., 2014; Kingma and Welling, 2014) or to classify them into different groups (MacQueen, 1967; Ester et al., 1996; Rokach and Maimon, 2005). Unsupervised models often involve comparing the similarity between two samples (e. g. , if two samples are close, they are clustered together) where the similarity measure has a heavy impact on the model's performance, which should be coincident with the nature of the data.

Although we are in the era of *Big Data* where huge amounts of data are produced, processed and stored at every second, annotated data's quantity is still limited, due to the fact that high quality data labeling requires large amount of careful human work. On the other side, unannotated data such as images, text corpus, videos, seem to be 'unlimited' and can be collected easily. To take profit of these unlimited resources,

self-supervised learning (sometimes considered as a specific type of unsupervised learning) attracts more and more attention (Arora et al., 2019; Xiao Liu et al., 2021), especially in *Natural Language Processing (NLP)* domain (Mikolov, K. Chen, et al., 2013; Devlin et al., 2018). The general idea is to create proper labels directly from the unannotated dataset, then run supervised learning models on top of it. Instead of focusing on the performance of the auxiliary supervised models, self-supervised learning focus on learning meaningful intermediate representations. The learned representations will be used directly or be fine-tuned to serve downstream tasks. For example, in NLP domain, a common practice is to mask (hold) a word from a sentence and let the model to predict the masked word (e. g. , masked language model in Section 2.4.3). By doing this, the model is forced to learn semantically meaningful representation of words which is used later in downstream task to improve the task's performance (Hendrycks et al., 2019; J. D. Lee et al., 2020).

Other categories of machine learning models also exist. *Semi-supervised learning* (Xiaojin Zhu, 2005) is a mixture of supervised learning and unsupervised learning where the algorithms are given a small amount of labeled data and much larger amount of unlabeled data. In *Reinforcement learning* (Sutton and Barto, 2018), the algorithm (referred as the 'agent') interacts with an environment instead of a fixed dataset and interactively gets feedback (*reward*) from the environment according to its actual state and next action, with the goal to maximize the total reward. These two categories will not be covered in this dissertation.

2.2.2 *Learning as optimization*

In this section, we introduce the learning (training) procedure of machine learning algorithms (models) focusing on supervised learning under a probabilistic setting.

Machine learning tasks typically involve three components: *dataset*, *model* and *loss function*. In supervised learning, we are given an annotated dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ consisting of n input-target pairs where each pair $(\boldsymbol{x}, y)$ is drawn independently from a fixed but unknown joint distribution P. A machine learning model (algorithm), considered as a family of functions $\{f|f \in \mathcal{F}\}$ that it can realize where each f corresponds to a specific setting of it, models the dependency of y on a given $\boldsymbol{x}$. $f(\boldsymbol{x})$ (often denoted as $\hat{y}$) is an estimator of y where f is referred as *predictor*. The discrepancy between the target y and its estimator $\hat{y}$ is measured by $\ell(\hat{y}, y)$ where ℓ is the loss function.

We denote the *expected risk* (also known as *expected loss, generalization error)* of a given predictor f as

$$\mathcal{L}(f) = \mathbb{E}[\ell(f(\boldsymbol{x}), y)] = \int \ell(f(\boldsymbol{x}), y) dP(\boldsymbol{x}, y). \tag{2.14}$$

It measures the general approximation quality of f under the data distribution P. Indeed, ideally we would like to learn (find) the *oracle best* predictor $f^\star$ which minimizes $\mathcal{L}(f)$ over all possible functions, i. e. , to learn

$$f^\star = \operatorname*{argmin}_f \mathcal{L}(f). \tag{2.15}$$

However, we have two main issues here: First, we don't have access to the underlying distribution P, we only have its samples in the dataset. Second, it's not possible to test all the functions. For the first issue, in practice, we use the *empirical risk* (also known as *empirical error, training error)* $\mathcal{L}_n(f)$, calculated based on the dataset, as a proxy of the expected risk $\mathcal{L}(f)$ where

$$\mathcal{L}_n(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\boldsymbol{x}_i), y_i). \tag{2.16}$$

For the second issue, we constrain the search of f on a set $\mathcal{F}$ (representing the model) that we think probably suits the task. Thereupon, the original goal of finding $f^\star$ (Eq. 2.15) is transferred to find $\hat{f} \in \mathcal{F}$ where

$$\hat{f} = \operatorname*{argmin}_{f\in\mathcal{F}} \mathcal{L}_n(f). \tag{2.17}$$

In most cases, $f \in \mathcal{F}$ is parameterized by a fixed number of parameters, denoted as a vector $\boldsymbol{w} \in \mathbb{R}^q$ (the number of parameters q can be different from the input's dimension p). By replacing f with $f_{\boldsymbol{w}}$, we rewrite the Equation 2.17 to

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}\in\mathcal{W}} \mathcal{L}_n(\boldsymbol{w}), \tag{2.18}$$

where

$$\mathcal{L}_n(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i) \tag{2.19}$$

with $\mathcal{W}$ being the candidate set of $\boldsymbol{w}$. Once the aforementioned three components (dataset, model, loss function) of the learning problem are defined, we typically use optimization algorithms such as the gradient descent based optimizer introduced in Algorithm 1, Section 2.1.2 to find $\hat{\boldsymbol{w}}$.

Taking again the spam detection (binary classification) task for example, one typical choice of the models is *logistic regression* (Cox, 1958) where we estimate the probability of an email being a spam in a log-linear way, formulated as follows:

$$\hat{y} = f_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x}), \tag{2.20}$$

where σ is the *sigmoid* function, $\boldsymbol{w} \in \mathbb{R}^p$. The corresponding loss function l is *log loss* (also known as *logistic loss* or *cross-entropy loss*), where its binary-class version is defined as follows,

$$\ell(\hat{y}, y) = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y})). \tag{2.21}$$

Actually, the choice of loss function ℓ is not unchangeable for a given model. In spam detection task, one intuitive loss function could be the *0-1 loss*:

$$\ell(g(\hat{y}), y) = \mathbb{1}_{g(\hat{y}) \neq y} = \begin{cases} 1, & \text{if } g(\hat{y}) \neq y \\ 0, & \text{if } g(\hat{y}) = y \end{cases} \tag{2.22}$$

where $g(\hat{y})$ is the corresponding label of $\hat{y}$ and $\mathbb{1}$ is the *indicator function*. To be more precise, $g(\hat{y}) = \mathbb{1}_{\hat{y} > thres}$ where *thres* represents a threshold, typically set to 0.5. However, as regard of the optimization process, convexity is a desirable property for the objective function to be minimized. It can guarantee that only one minimum exists (local minimum equals to the global minimum). So instead of using 0-1 loss, we typically use log loss (Eq. 2.21). Meanwhile, the choice of ℓ can be task-dependent and objective-dependent, as one can argue that it should be more critical to misclassify a non-spam email as spam because the receiver may miss important messages. In this scenario, a weighted version of log loss should be considered.

In addition, most algorithms specifically model a *bias* term by adding a dimension in the input feature vector $\boldsymbol{x}$ with value 1. Thus, instead of having $\boldsymbol{x} = [x_1, \ldots, x_p]$, we have

$$\boldsymbol{x} = [x_1, \ldots, x_p, 1]. \tag{2.23}$$

For the rest of this dissertation, we include the bias term by default while keeping the same notation unchanged.

2.2.3 *Error decomposition*

We expect the learned predictor $\hat{f}$ (Eq. 2.17) to have a good performance in practice, expressed by a low generalization error $\mathcal{L}(\hat{f})$. This error is closely related to the capacity (complexity) of the model $\mathcal{F}$ which is typically measured by *Vapnik–Chervonenkis dimension* (*VC-dimension*, Vapnik, 1995). To be more precise, a classification model with VC-dimension n means there exists a set of n points that can be shattered by it, and no set of $n+1$ points can be shattered. The 'shatter' here means that for any label assignment (with label 0 or 1) of each data point, their exists at least one function inside the function set, that separates the data points with no mistake. For example, the set of hyperplanes in 2-d space (i. e. , lines) is of VC-dimension 3 because any label assignment of 3 data points that are not colinear can be separated perfectly (i. e. , shattered) by it, and no set of 4 points can be shattered.

Denoting $f^\star_{\mathcal{F}}$ the predictor which minimizes the expected risk $\mathcal{L}(f)$ on the candidate function set $\mathcal{F}$, i. e. ,

$$f^\star_{\mathcal{F}} = \operatorname*{argmin}_{f \in \mathcal{F}} \mathcal{L}(f), \tag{2.24}$$

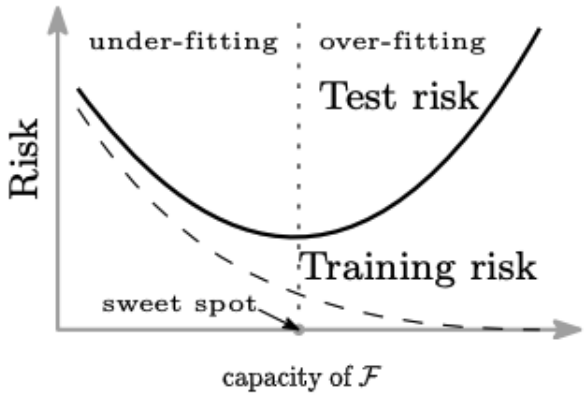


Figure 2.2 – A typical relationship between the capacity of the model $\mathcal{F}$, the training risk $\mathcal{L}_n(\hat{f})$ (training error, dashed line) and the test risk $\mathcal{L}(\hat{f})$ (generalization error, solid line) of the learned predictor $\hat{f} \in \mathcal{F}$. The training risk always goes down when we increase the capacity of $\mathcal{F}$, while the test risk first goes down, then goes up, forming a U-shaped curve. Figure taken from Belkin et al., 2019.

the expectation of the generalization error of the learned predictor $\hat{f}$ can be decomposed as sum of three error terms as follows (Bottou and Bousquet, 2008):

$$\begin{aligned}\mathbb{E}[\mathcal{L}(\hat{f})] &= \mathbb{E}[\mathcal{L}(\hat{f}) - \mathcal{L}(f^\star_\mathcal{F})] + \mathbb{E}[\mathcal{L}(f^\star_\mathcal{F}) - \mathcal{L}(f^\star)] + \mathbb{E}[\mathcal{L}(f^\star)] \\ &= \epsilon_{est} + \epsilon_{app} + \mathbb{E}[\mathcal{L}(f^\star)],\end{aligned} \tag{2.25}$$

where the expectation is taken on the random choice of the training data. The *estimation error* ϵ_{est} measures the impact of using empirical risk instead of expected risk when searching the best predictor. *Vapnik–Chervonenkis theory* (*VC theory*, Vapnik, 1995) shows that ϵ_{est} can be bounded by some term which increases with the VC-dimension of $\mathcal{F}$ and decreases with the number of training samples n. The *approximation error* ϵ_{app} measures how closely the optimal solution found in $\mathcal{F}$ (i. e. , $f^\star_\mathcal{F}$) approximates the oracle one (i. e. , $f^\star$), it decreases when we choose a larger set of $\mathcal{F}$. The final error term $\mathbb{E}[\mathcal{L}(f^\star)]$ is an *irreducible error*, corresponding to the generalization error of the oracle best solution $f^\star$, which is the smallest generalization error over all functions. When we fix the number of training samples n, $\mathbb{E}[\mathcal{L}(\hat{f})]$ only depends on the choice of the candidate function set $\mathcal{F}$.

In general, a U-shaped curve is formed for the generalization error $\mathcal{L}(\hat{f})$ when we tune $\mathcal{F}$'s (the model's) capacity (complexity) from low to high, see Figure 2.2. The increase of $\mathcal{F}$'s capacity is typically done by adding functions into it, i. e. , using larger $\mathcal{F}$, and vice versa. This U-shaped curve is closely related to two major problems in machine learning: *underfitting* and *overfitting*. At the left part of the curve, $\mathcal{F}$'s capacity is relatively low, we are at underfitting zone with high ϵ_{app}

and low ϵ_{est}. The training error and generalization error both go down along with the increase of $\mathcal{F}$'s capacity. At the right part of the curve, $\mathcal{F}$'s capacity is relatively high, we are at overfitting zone with low ϵ_{app} and high ϵ_{est}. The training error still goes down but the generalization error goes up with the increase of the $\mathcal{F}$'s capacity. Therefore, in order to have the lowest generalization error, one need to find the **optimal tradeoff** point of $\mathcal{F}$'s capacity which sits at the bottom of 'U' (marked as 'sweet spot' in the figure).

This tradeoff is also known as *bias-variance tradeoff* (Geman, Bienenstock, and Doursat, 1992), often analyzed and presented in the case where the loss function ℓ is the square loss, i.e., $\ell(\hat{y}, y) = (y - \hat{y})^2$. The approximation error ϵ_{app} represents the bias and the estimation error ϵ_{est} represents the variance.

In practice, as $\hat{f}$, the minimizer of $\mathcal{L}_n(f)$, needs to be found with limited time and calculation resources, we generally stop the optimization process at a point $\tilde{f}$ when we think we are close enough to $\hat{f}$. Taking the gradient descent optimizer for example, we stop it either when the maximum number of iterations is reached or when the change of loss is inside a predefined tolerance range. The expectation of the generalization error of the practical solution $\tilde{f}$ can be decomposed as follows (Bottou and Bousquet, 2008):

$$\begin{aligned}\mathbb{E}[\mathcal{L}(\tilde{f})] &= \mathbb{E}[\mathcal{L}(\tilde{f}) - \mathcal{L}(\hat{f})] + \mathbb{E}[\mathcal{L}(\hat{f}) - \mathcal{L}(f^\star_{\mathcal{F}})] + \\ &\quad \mathbb{E}[\mathcal{L}(f^\star_{\mathcal{F}}) - \mathcal{L}(f^\star)] + \mathbb{E}[\mathcal{L}(f^\star)] \\ &= \epsilon_{opt} + \epsilon_{est} + \epsilon_{app} + \mathbb{E}[\mathcal{L}(f^\star)].\end{aligned} \tag{2.26}$$

Therefore, in practice, the choice of optimizer and the number of iterations used in the optimization process also influence the final performance of the model.

2.2.4 *Regularization*

In order to prevent overfitting and ill-posed optimization problem during the learning procedure, *regularization* is needed. Different strategies exist, such as *early stopping* (Yao, Rosasco, and Caponnetto, 2007), *L_p-norm penalties* (A. N. Tikhonov, 1963), *bagging* (Breiman, 1996). Quite often, regularization could be expressed as adding an *penalty* term on the objective function which can be seen as a preference or prior imposed on the solution that the algorithm should find. The regularized version of the optimization problem, corresponding to the Equation 2.17, is:

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}}\, \mathcal{L}_n(f) + \lambda R(f), \tag{2.27}$$

where R is the penalty term on f and λ is a hyper-parameter which controls the degree of regularization. When f is parameterized by $\boldsymbol{w}$, we have the regularized estimator of $\boldsymbol{w}$ as:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w} \in \mathcal{W}}{\operatorname{argmin}} \mathcal{L}_n(\boldsymbol{w}) + \lambda R(\boldsymbol{w}). \tag{2.28}$$

For L_p-norm regularization, two common used terms are L_2 and L_1 regularization, where we penalize large weights of parameters. For the L_2 regularization (also known as *weight decay*, Hanson and Pratt, 1989), $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$. In the linear regression case, adding this term (*Ridge regression*, Hoerl and Kennard, 1970) resolves the ill-posed optimization problem when the features are highly linear correlated to each other. For L_1 regularization, $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$. In linear regression case, adding this term (*Lasso*, R. Tibshirani, 1996) shrinks the weights of some parameters to zero which implies implicitly a feature selection and introduces sparsity in the model.

2.2.5 *Evaluation protocol*

Hyper-parameters, such as the degree of regularization (λ of the Equation. 2.27), control the high-level behavior of the learning algorithms. They are set before the learning procedure starts. In order to choose the best one, we need to access the generalization error $\mathcal{L}(\hat{f})$ of the learned predictor $\hat{f}$ under each hyper-parameter. However, as the underlying distribution P is unknown, we use the empirical error $\mathcal{L}_n(\hat{f})$ on unseen dataset (*validation set*) as a proxy. Meanwhile, in order to have an idea about the performance of the final chosen predictor (learned under the best hyper-parameter), another unseen dataset (*test set*) is required. So in the end, during the whole machine learning process, we need three datasets: training set, validation set and test set. All of them should come from the same underlying distribution P and be independent from each other.

In practice, instead of three independently collected datasets, we often have one whole dataset at hand. To create aforementioned datasets, we randomly split the data into three disjoint partitions, usually with a train-validation-test cut equals to a%-b%-c% where $a > b \geqslant c > 0$. However, this one-shot split may potentially cause the model's performance to depend on a particular random choice of train-validation split and it reduces the amount of data that the training set can use. A *cross-validation* strategy (Stone, 1974) is adapted to solve these issues where the test set is still held out for final evaluation, but the training and validation set are created repeatedly by randomly taking parts of the rest data. In a common *k-fold cross-validation* approach, the data is randomly split to k equal-sized subsets (folds). Given a hyper-parameters setting, for each one of the k folds, we take the rest $k-1$ folds as training set to train the algorithm and this fold as evaluation

set to evaluate the algorithm. The performance of the given hyper-parameter setting is judged by the average evaluation performance of k rounds. We then take the best hyper-parameter setting, retrain the algorithm using the whole k folds data to get our final predictor $\hat{f}$. In the end, the actual performance of the final predictor $\hat{f}$ is evaluated on the test set.

2.2.6 *Evaluation metrics*

Evaluation metrics are the measures that the model is **eventually** evaluated with, which can be different from the objective function used in the learning procedure. Theoretically, we can chose any measure we want, but an ideal one should reflect the real world needs of the task on which we apply the model. The common reason that we do not directly optimize our model using the actual evaluation metrics is that quite often, they are non-convex or not even smooth, e. g. , the *error rate* (the average of 0-1 loss on all samples).

Taking the logistic regression for example, as we have seen previously, we use the log loss as loss function (Eq. 2.21). The actual evaluation metrics can be *accuracy*, *F1-score*, *Receiver operating characteristic (ROC) curve*, *Precision-Recall (PR) curve*, etc. , depending on the practical needs. In the following, we first introduce several basic measure components and then use them to present ROC curve and PR curve which we'll use in our experiments, focusing on the binary classification case.

2.2.6.1 *Basics concepts*

		Predicted class	
		positive	negative
Actual class	positive	TP	FN
	negative	FP	TN

Table 2.1 – Confusion matrix of binary classification.

For binary classification tasks, given a discrimination threshold *thres*, a sample is classified as positive (P) if the predicted probability p (that the sample belongs to the positive class) is greater or equal than the predefined threshold, i. e. , $p \geqslant thres$. On the other hand, if $p < thres$, it is classified as negative (N). In general, *thres* is set to 0.5. By comparing the predicted class of each sample with its actual class, we can tell if the predictions we made are true (T) or false (F). Therefore, we can separate the prediction outcomes to four types, summarized as a *confusion matrix* (also known as *error matrix*, Stehman, 1997) shown in Table 2.1, where TP (True Positives), FP (False Positives), TN (True Negatives), FN (False Negatives) represents

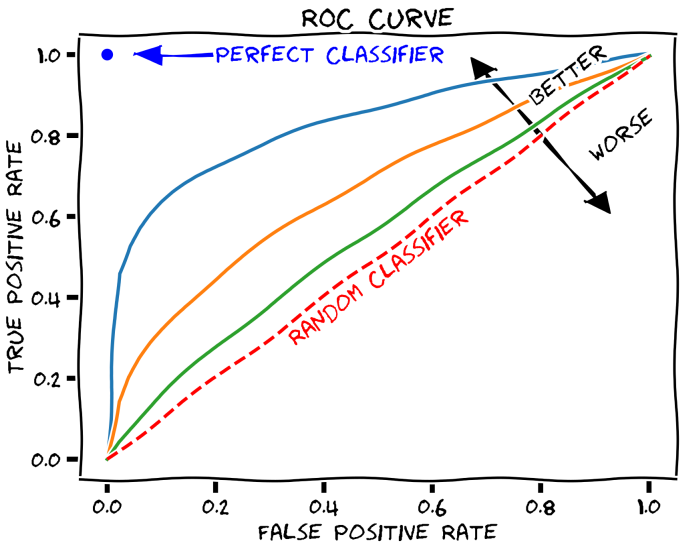


Figure 2.3 – Example of ROC curve on different classifiers where a better (resp. worse) ROC curve will be closer to the left upper corner (resp. right lower corner) of the coordinate. Figure taken from wikipedia [1].

the number of samples that correctly predicted as positive, incorrectly predicted as positive, correctly predicted as negative, incorrectly predicted as negative respectively.

2.2.6.2 *ROC curve & AUC score*

We denote *True positive rate (TPR)* and *False positive rate (FPR)* as follows:

$$\begin{aligned} TPR &= \frac{TP}{TP+FN}\,, \\ FPR &= \frac{FP}{FP+TN}\,, \end{aligned} \tag{2.29}$$

where TPR represents the ratio of samples correctly predicted as positive (TP) among all positive samples (TP+FN) and FPR represents the ratio of samples incorrectly predicted as positive (FP) among all negative samples (FP+TN).

ROC curve (Fawcett, 2006) is a graphical curve where each point on the curve specifies (FPR, TPR) tuple of certain discrimination threshold *thres*. Specifically, by decreasing *thres* gradually from 1 to 0, ROC curve starts at $(0, 0)$ where all samples are classified as negative and ends at $(1, 1)$ where all samples are classified as positive. Figure 2.3 presents an illustrated example of several classifiers with different performance. Generally speaking, the better (resp. worse) the classifier is, the closer it approaches the left upper corner (resp. right lower corner) of the coordinate. A perfect classifier (classifies all samples

1. https://en.wikipedia.org/wiki/Receiver_operating_characteristic

correctly) assigns probability $p = 1$ for ever positive sample and $p = 0$ for every negative sample, i. e. , $TPR = 1\ \forall FPR$ and $FPR = 0\ \forall TPR$. A random classifier, sitting on the diagonal of the coordinate, has $TPR = FPR$ all the time.

In practice, to compare different ROC curves in a straightforward way, *Area Under the Curve (AUC) score* is often used where we simply calculate the surface of the area that covered by the ROC curve. It is equal to the probability that a classifier will rank a randomly chosen positive sample higher than a randomly chosen negative one (Fawcett, 2006), which indicates a classifier's general performance.

Despite its wide usage, ROC curve may not be that informative (Davis and Goadrich, 2006) when applied on an highly imbalanced dataset with few minor class samples and many major class samples. Suppose that minor class is positive class, then due to the large number of negative samples (FP+TN), FPR will not be sensitive to the change of FP. Thus, two algorithms with important gap of FP may still be comparable on ROC curve. To overcome this, *Precision-Recall (PR curve)* is used as an alternative for highly imbalanced dataset with a focus on the positive (minor) class.

2.2.6.3 *PR Curve & AP score*

We denote *Precision* and *Recall* as follows:

$$\begin{aligned} Precision &= \frac{TP}{TP + FP}\ , \\ Recall &= \frac{TP}{TP + FN}\ , \end{aligned} \tag{2.30}$$

where Precision, Recall represent the ratio of samples correctly predicted as positive (TP) among all positive predictions (TP+FP) and among all positive samples (TP+FN), respectively. In other words, Precision shows how accurate the positive predictions are and Recall (exactly the True Positive Rate (TPR) in Equation 2.29) shows the coverage of actual positive samples in the predictions.

Same as ROC curve, PR curve can be obtained by varying the discrimination threshold (*thres*) of the classifier where each point on the PR curve specifies $(Recall, Precision)$ tuple of the corresponding threshold. Specifically, by decreasing *thres* gradually from 1 to 0, PR curve starts at $(0, 1)$ where are samples classified as negative and ends at $(1, r)$ where all samples are classified as positive, with r the ratio of positive samples in the dataset ($r = 0.5$ for a balanced dataset). The PR curve of a random classifier is represented by a horizontal line where the value on y-axis is r. A better (resp. worse) classifier will approach closer to the upper right corner (resp. lower left corner) of the coordinate.

To summarize PR curve in one single score, unlike ROC curve, we can't use its AUC score because linear interpolation of points is

typically involved in calculating the AUC score, it will give overly-optimistic estimate of the classifier's performance (Davis and Goadrich, 2006). Instead, we use the *Average precision (AP) score* which is a weighted sum of the Precisions achieved at each threshold *thres*, defined as follows:

$$AP = \sum_n (Recall_n - Recall_{n-1}) Precision_n, \tag{2.31}$$

where $Precision_n$ and $Recall_n$ are the Precision and Recall at n-th threshold respectively.

Comparing to ROC curve which looks at all outcomes equally, PR curve focus mainly on the prediction results corresponding to the positive class where True Negatives (TN) are totally ignored. Thus, PR curve suits well the case where the positive class is more important than the negative class, or the positive class is the minor class in an imbalanced dataset. Under other conditions, using ROC curve is preferred.

2.3 NEURAL NETWORKS

Artificial Neural Networks (ANNs), or *Neural Networks (NNs)* in short, is a specific kind of machine learning models which is originally inspired from neuroscience to imitate human brains' working mechanism. Nowadays, they go beyond that and are more sophisticatedly designed which achieve state-of-the-art results across various domains, such as computer vision (K. He et al., 2015; Tan and Q. V. Le, 2020), Natural Language Processing (NLP) (Devlin et al., 2018; Brown et al., 2020), game playing (Silver et al., 2016; Justesen et al., 2017), where standard machine learning algorithms fail to get close performance. In the following discussion, first, we briefly introduce biological neurons' working mechanism and how artificial neurons imitate them in a mathematical way. Then, we introduce the concept of *layer*, which is the basic building block of NNs and we provide several examples of commonly used types of layers. Finally, we discuss the reasons why neural networks are so successful nowadays and the challenges they potentially face.

2.3.1 *Basic concepts*

Human brains are able to deal with extremely complex tasks such as visioning, feeling, talking, reasoning, which involve cooperation of huge amount of different neurons as information carriers. Information, in form of impulses and chemical signals, is transmitted by neurons through different areas of the brain. A typical structure of neurons can be seen in Figure 2.4 where the dendrites receive the signals in a non-uniform manner, aggregate and pass them to the cell body,

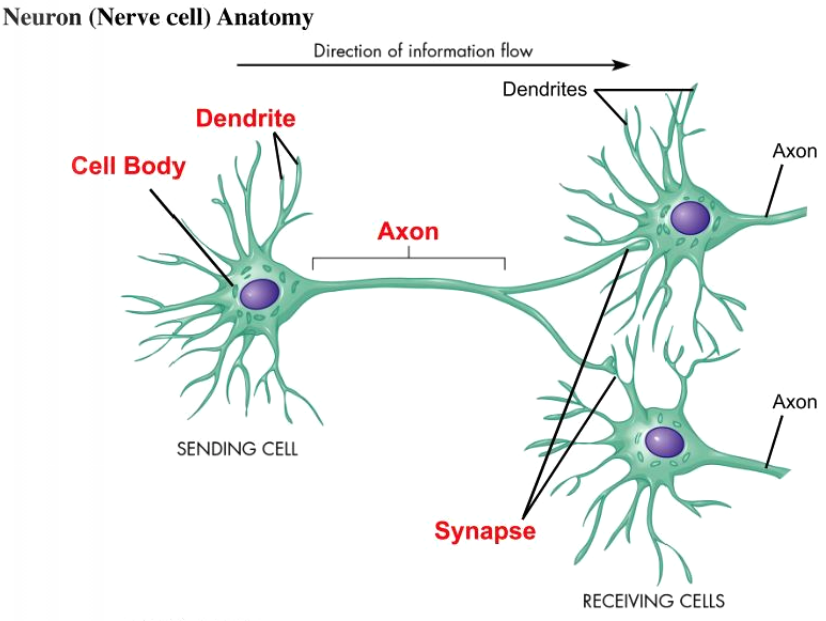


Figure 2.4 – A typical structure of biological neurons [2]. The dendrites receive the signals in a non-uniform manner, aggregate and pass them to the cell body, then the axon sends the aggregated signal to other connected neurons through synapses.

then the axon sends the aggregated signal to other connected neurons through synapses if the aggregated signal passes certain threshold (*activated*).

Artificial neurons are introduced as mathematical models that imitate the biological neurons' working mechanism in a simplified manner. For a basic artificial neuron, given an input signal $\boldsymbol{x} \in \mathbb{R}^p$, the output signal o is formulated as follows:

$$o = g(z) = g(\boldsymbol{w}^\top \boldsymbol{x}), \tag{2.32}$$

where $\boldsymbol{w} \in \mathbb{R}^p$ is the weight vector, g is typically a non-linear function known as *activation function* (Section 2.1.3), and $z \in \mathbb{R}$ is the aggregated signal.

Artificial neural networks (ANNs), consisting of artificial neurons, are introduced to imitate functionalities of our brain. They are generally presented as a stack of *layers* where each layer is a group of artificial neurons that operate together to provide certain functionality at a specific depth of NN. Each layer transforms its input to a slightly more composite, often more abstract representation. Generally speaking, for a neural network, the very first layer which just takes the input as it is is referred as *input layer* and the last layer which gives the output of the NN is referred as *output layer*. The rest layers in between are all denoted as *hidden layers*. We denote the *depth* of NN as the number of

2. source: https://www.yumpu.com/en/document/read/33756198/axon-dendrite-cell-body-neuron-nerve-cell-anatomy-synapse

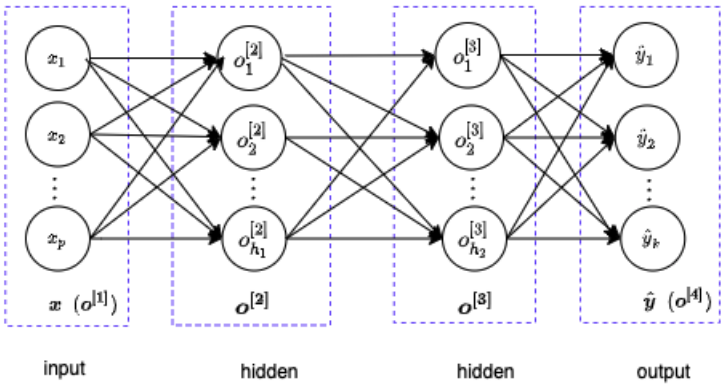


Figure 2.5 – A 4-layer feedforward neural network. Each node represents an artificial neuron with the directed link indicating the information flow and each blue dashed rectangle represents a layer. Only the output of each neuron is displayed on the node.

layers it has, **excluding** the input layer. We often call a neural network as *Deep Neural Network (DNN)* if its depth is profound.

2.3.2 *Notations*

We present a 4-layer *FeedForward Neural Network (FFNN)* (a basic type of NN) in Figure 2.5 to introduce some basic notations of NN. In the figure, each *node* (circled *unit*) represents an artificial neuron, each blue dashed rectangle represents a layer. The directed links, often referred as *edges*, indicate the flow of information where each node receives information from the nodes pointing to it and sends information to the nodes it points to. Thus, for this FFNN, the information travels only in the forward direction. It starts from the input layer, goes through two hidden layers and ends at the output layer. It's worth to mention that for each neuron, only its output ($\boldsymbol{o}$) is displayed on the node, the other information (g, z, $\boldsymbol{w}$ in Equation 2.32) is omitted.

In general, for a NN with L layers (depth $L-1$), we denote the l-th layer as a function $\Phi^{[l]}$ which takes $\boldsymbol{o}^{[\boldsymbol{l-1}]}$ (the output of layer $l-1$) as input and gives $\boldsymbol{o}^{[l]}$ as output, i. e. ,

$$\boldsymbol{o}^{[l]} = \Phi^{[l]}(\boldsymbol{o}^{[l-1]}) \quad \forall l \in [1, \ldots, L], \tag{2.33}$$

where $\boldsymbol{o}^{[0]} = \boldsymbol{x}$ and $\Phi^{[1]}(\boldsymbol{x}) = \boldsymbol{x}$. Specifically, we denote $\boldsymbol{o}^{[1]}$ as $\boldsymbol{x}$ for the input layer and $\boldsymbol{o}^{[L]}$ as $\hat{\boldsymbol{y}}$ for the output layer. Considering the whole L-layer NN as a function f, given the input $\boldsymbol{x}$, we get its output as

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}) = \Phi^{[L]}\left(\cdots \Phi^{[2]}\left(\Phi^{[1]}(\boldsymbol{x})\right)\cdots\right). \tag{2.34}$$

2.3.3 *Type of layers*

The aforementioned FFNN is just a simple example of NNs, one can design various formats of NNs by stacking different types of layers (Leijnen and Veen, 2020). Here we introduce several popular and commonly used layers. Without loss of generality, each layer is denoted simply as a function $\Phi_{\boldsymbol{W}}$ where $\boldsymbol{W}$ specifies its parameters.

FULLY CONNECTED LAYER *Fully connected (FC) layer* (also known as *Dense layer*) is the essential building block of FFNN where each node inside the layer is connected to all the nodes in the previous layer. For a FC layer with k nodes (i. e. , with k-dimensional output), we formulate it as

$$\Phi_{\boldsymbol{W}}(\boldsymbol{x}) = g(\boldsymbol{W}^\top \boldsymbol{x}) \in \mathbb{R}^k, \tag{2.35}$$

where $\boldsymbol{x} \in \mathbb{R}^p$ is the input vector, $\boldsymbol{W} \in \mathbb{R}^{p \times k}$ represents the parameters and g is an activation function.

The 'brute force' FC layer can be computationally expensive when $p \times k$ is large due to its full connectivity with previous layer ($p \times k$ parameters to learn). Typically, it is served as the last layer of NN in classification task with g being the softmax activation function. In this case, the input vector is transformed to a probability distribution as output where the output's dimension k is the number of classes.

CONVOLUTIONAL LAYER *Convolutional (CONV) layer* is the essential building block of *Convolutional Neural Network (CNN)* which has a big success in computer vision area for tasks such as image classification (Krizhevsky, Sutskever, and Geoffrey E Hinton, 2012; K. He et al., 2015; Simonyan and Zisserman, 2015), video analysis (Karpathy et al., 2014). It also shows good performance in other tasks such as sentence classification (Kalchbrenner, Grefenstette, and Blunsom, 2014; Kim, 2014), time series analysis (B. Zhao et al., 2017).

Traditionally, CONV layer is designed to take two-dimensional data (e. g. , an image) as input. For each operation, a matrix multiplication is performed between a part of the input (e. g. , a part of the image) and a weight matrix (referred as a *filter* or a *kernel*). This operation is applied repeatedly (with the same kernel) along different axis of the input, known as 1D CONV if applied along one single axis (Kim, 2014) and 2D CONV for two axis (Krizhevsky, Sutskever, and Geoffrey E Hinton, 2012).

Taking the 2D CONV layer for image classification task as an example, we denote the input image as a matrix $\boldsymbol{X} \in \mathbb{R}^{p_1 \times p_2}$ where p_1 and p_2 refer to its height and width respectively, the kernel as $\boldsymbol{W} \in \mathbb{R}^{k_1 \times k_2}$ ($k_1 \leqslant p_1$, $k_2 \leqslant p_2$). Then, the output of the 2D CONV layer is a matrix

$\Phi_{\boldsymbol{W}}(\boldsymbol{X}) \in \mathbb{R}^{(p_1-k_1+1)\times(p_2-k_2+1)}$. Its entry of i-th row and j-th column is formulated as:

$$\Phi_{\boldsymbol{W}}(\boldsymbol{X})_{i,j} = (\boldsymbol{W} \star \boldsymbol{X})(i,j) = \sum_{n=1}^{k_1}\sum_{m=1}^{k_2} W_{n,m} X_{i+n,j+m} \tag{2.36}$$

where $\star$ represents the *cross-correlation* operator (see convolution operator, Section 2.1.1 for details).

Comparing to FC layer, CONV layer has *local connectivity* and *parameter sharing* as properties. In FC layer, each node is connected to all the previous nodes (i. e. , dense connections), while in CONV layer, each node is only connected to a local region the previous nodes (i. e. , sparse connections). This local connectivity can be seen as an analogy to cells in animal's vision system where they only look at a preferred location of images (Hubel and Wiesel, 1959; Lindsay, 2020). Also, all the nodes in the CONV layer share the same parameters (same convolution kernel $\boldsymbol{W}$) which significantly reduces the number of parameters to learn.

RECURRENT LAYER *Recurrent layer (RNN layer)* is the essential building block of *Recurrent Neural Network (RNN)* which is commonly used in sequential tasks, especially in NLP area, such as language translation (Cho et al., 2014; Sutskever, Vinyals, and Q. V. Le, 2014), speech recognition (Alex Graves, Mohamed, and G. Hinton, 2013). RNN layer takes a sequence of vectors $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ as input, and output a sequence of vectors $(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T)$ of the same size, where T is the length of the sequence, often referring to 'time'. Thus, the RNN layer can be generally expressed as:

$$\Phi_{\boldsymbol{W}}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = (\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T), \tag{2.37}$$

where $\boldsymbol{W}$ groups all the parameters in the RNN layer. To be more precise, for each time step t, an output vector $\boldsymbol{o}_t$ as well as a **hidden** vector $\boldsymbol{h}_t$ is produced in a recursive manner. In the standard RNN layer, $\boldsymbol{h}_t$ and $\boldsymbol{o}_t$ are defined as follows:

$$\begin{aligned} \boldsymbol{h}_t &= g_h(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t), \\ \boldsymbol{o}_t &= g_o(\boldsymbol{W}_o \boldsymbol{h}_t), \end{aligned} \tag{2.38}$$

where g_h and g_o are the activation functions, $\boldsymbol{W}_h, \boldsymbol{W}_x, \boldsymbol{W}_o$ are the weight matrices, and the initial hidden state $\boldsymbol{h_0}$ is often set to $\mathbf{0}$. It's worth to mention that if we set $\boldsymbol{W}_h = 0$, then we'll have no information flowing between each time step. The RNN layer turns to a simple two-layer FFNN at each time step, running in parallel.

A well-known issue for the standard RNN layer is that it can't learn long term dependency in the sequence due to *vanishing gradient* and *exploding gradient* problems (Hochreiter, 1998; Pascanu, Mikolov, and Yoshua Bengio, 2013). Variants such as *Long Short-Term Memory* (*LSTM*,

Hochreiter and Schmidhuber, 1997), *Gated Recurrent Unit* (*GRU*, Cho et al., 2014) are introduced to solve this issue by adding 'gate' units to control information flow in the layer. In our dissertation, we mainly use LSTM as variant of RNN, where its output $\boldsymbol{o}_t$ at time step t is calculated as follows (Hochreiter and Schmidhuber, 1997):

$$\begin{aligned}
\boldsymbol{F}_t &= \sigma(\boldsymbol{W}_{fh}\boldsymbol{o}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t) \\
\boldsymbol{I}_t &= \sigma(\boldsymbol{W}_{ih}\boldsymbol{o}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t) \\
\boldsymbol{O}_t &= \sigma(\boldsymbol{W}_{oh}\boldsymbol{o}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t) \\
\tilde{\boldsymbol{c}}_t &= \sigma(\boldsymbol{W}_{ch}\boldsymbol{o}_{t-1} + \boldsymbol{W}_{cx}\boldsymbol{x}_t) \\
\boldsymbol{c}_t &= \boldsymbol{F}_t \circ c_{t-1} + \boldsymbol{I}_t \circ \tilde{c}_t \\
\boldsymbol{o}_t &= \boldsymbol{O}_t \circ \tanh(\boldsymbol{c}_t).
\end{aligned} \tag{2.39}$$

$\boldsymbol{F}_t$, $\boldsymbol{I}_t$, $\boldsymbol{O}_t$, $\boldsymbol{c}_t$ refer to the *forget gate, input gate, output gate, cell state,* respectively. Basically, at each time step t, the present cell state $\boldsymbol{c}_t$ (corresponding to the hidden state $\boldsymbol{h}_t$ in standard RNN layer) is updated as a sum of previous cell state $\boldsymbol{c}_{t-1}$ through forget gate $\boldsymbol{F}_t$ and additional cell state $\tilde{\boldsymbol{c}}_t$ through input gate $\boldsymbol{I}_t$. The output $\boldsymbol{o}_t$ is then obtained as a *tanh*-transformed cell state through the output gate $\boldsymbol{O}_t$.

2.3.4 *Optimization*

BACKPROPAGATION As regard of training the neural networks, we can put it inside the framework of Equation 2.18 as an optimization problem, using the gradient decent optimizer (introduced in Section 2.1.2) to find the optimal weight parameters $\boldsymbol{w}$ that minimize the neural network model's training error $\mathcal{L}_n(\boldsymbol{w})$. In order to do that, we need to calculate the gradient $\nabla_{\boldsymbol{w}} f$ where f is the function that represents the NN. However, as NNs are generally a stack of sophisticated layers with non-linear functions, the form of f could be extremely complex which makes it hard to calculate $\nabla_{\boldsymbol{w}} f$ directly. *Backpropagation* (*BP*, D. E. Rumelhart, G. E. Hinton, and R. J. Williams, 1988) algorithm is widely adopted due to its efficiency on calculating the gradient. In fact, based on the *chain rule*, the gradient of the f with respect to any parameter w_i^l at layer l, can be decomposed as a multiplication of other derivatives which can be calculated in a layer-wise manner. To prevent redundant calculation, the calculating is proceeded in a **backward** way that starts from the very last layer (the output layer). As for training RNNs, a similar version called *Backpropagation Through Time* (*BPTT*, Werbos, 1990) is used by unfolding the calculation through time.

GLOBAL MINIMUM? For convex optimization problem, as we have already discussed previously in Section 2.1.2, (first-order) gradient descent based algorithms such as Algorithm 1, guarantee to find the global minimum point. Unfortunately, in non-convex optimization problem, there's no such guarantee. In fact, they'll find *critical points*

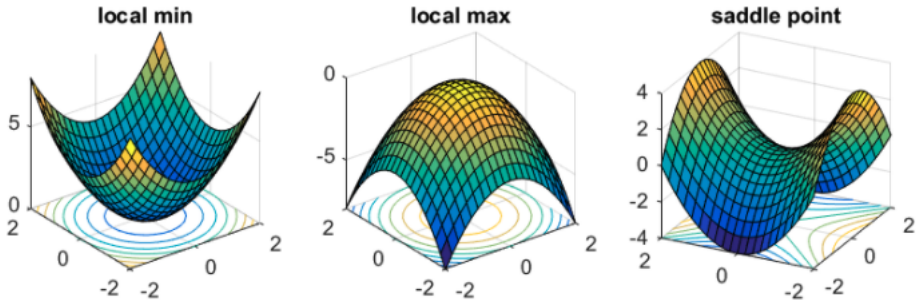


Figure 2.6 – An example of critical points where the gradient equals to $\mathbf{0}$. From left to right, we present local minimum, local maximum and saddle point respectively. Figure taken from blog of Rong Ge. [3]

(the points where the gradient goes to $\mathbf{0}$) which can be either *local minimum*, *global minimum* or *saddle point*. An example of local minimum and saddle point is shown in Figure 2.6.

Actually, the function f which represents the neural networks is typically non-convex. Taking the FFNN in Figure 2.5 as an example: one can easily permute the position of two nodes in a specific layer along with their associated edges and weights to $\boldsymbol{x}$, this operation does not change the value of the objective function to be optimized which violates the definition of convexity. In practice, both saddle points and local minima seem not a big issue for deep neural networks, optimized using adaptive optimizers. For local minima issues, Choromanska et al., 2015 show that for large size NN, local minima are close to the global minimum and local minima points are of high quality (measured by test error). For saddle points issues, Dauphin et al., 2014 argue that they are more critical than local minima issues, especially in high dimension space. The authors propose a second-order saddle-free Newton method which is able to escape the saddle points rapidly. As regard of first-order gradient descent algorithms, Staib et al., 2020 state that adaptive optimizers can efficiently escape from saddle points. They provide a convergence proof under certain conditions, by considering adaptive optimizers as preconditioned SGD in an online manner.

2.3.5 *Success reasons*

Deep Neural Networks (DNNs) seem to be extremely powerful that they achieve state-of-the-art results for various tasks in many domains [4], such as computer vision (K. He et al., 2015; Tan and Q. V. Le, 2020), NLP (Devlin et al., 2018; Brown et al., 2020), recommendation

3. Escaping from Saddle Points: https://www.offconvex.org/2016/03/22/saddlepoints/

4. Browse State-of-the-Art: https://paperswithcode.com/sota

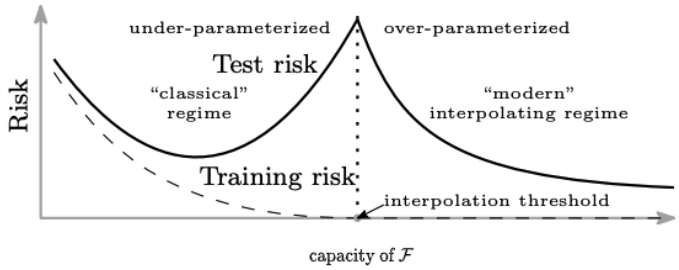


Figure 2.7 – Double descent phenomena for deep neural networks. The typical U-shape behavior of the generalization error (solid line) is kept before the capacity of DNN (denoted as $\mathcal{F}$) reaches the interpolation threshold, where the training error reaches zero. After the interpolation threshold, the generalization error goes down again when we increase the capacity. Figure taken from Belkin et al., 2019.

system (Hidasi et al., 2016; Berg, Kipf, and Welling, 2017; Kang and McAuley, 2018; Sun et al., 2019). At the same time, they are sophisticated designed by stacking various types of layers, which makes it not easy to interprete the exact reasons of their success. Nevertheless, several insights have been provided by researchers.

STRONG EXPRESSIVENESS According to the *Universal Approximation Theorem* (Cybenkot, 2006), a simple 2-layer FFNN with arbitrary nodes in hidden layer (bounded depth, unbounded width) and a sufficiently smooth activation function can generally approximate any continuous function closely. Kidger and Lyons, 2020 further show the dual case where *Deep Narrow Networks* (unbounded depth, relative small width) have the same approximation power. Therefore, for any oracle best solution $f^\star$, we may find one NN structure close enough to it, i. e. , having very small approximation error ϵ_{app} (Eq. 2.25).

GENERALIZATION POWER As mentioned previously in Section 2.2.3, strong expressiveness generally means complex models, which may lead to overfitting (right part of the U-shaped curve in Fig. 2.2). For NNs, regularization techniques such as *batch normalization* (Sergey Ioffe and Szegedy, 2015), *layer normalization* (Ba, Kiros, and Geoffrey E. Hinton, 2016), *dropout* (Srivastava et al., 2014), *early stopping* (Yao, Rosasco, and Caponnetto, 2007) are commonly applied to prevent it.

Interestingly, recent work of Belkin et al., 2019 show that over-parameterized DNNs, generalize surprisingly well instead of overfitting. The generalization error first goes down then goes up just like traditional U-shaped curve, when we increase the capacity of DNN (denoted as $\mathcal{F}$). However, it goes down again once the *interpolation threshold* of DNN's capacity is reached where the training error equals to zero, see Figure 2.7. This phenomena is known as *double descent*, of

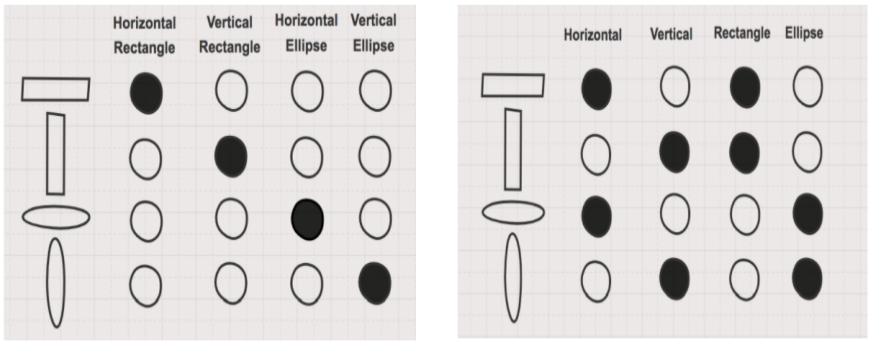


Figure 2.8 – An example of non-distributed representation (left) vs distributed representation (right). Figure taken from Garrett Hoffman's blog [5].

which exact causes are still under investigation (C. Zhang et al., 2017; C. Li et al., 2018; Zitong Yang et al., 2020).

DISTRIBUTED REPRESENTATION *Distributed representation* is a core idea in DNNs where we assume that the data is generated as a composition of attributes or features, and usually in a hierarchy way (e. g. , stack of layers), which have already proven its reasonableness in computer vision area (Zeiler and Fergus, 2013).

Let's take a look at the differences between a distributed representation and a non-distributed one. Supposing that we are asked to describe four different graphic shapes, shown in each row of Figure 2.8. For non-distributed representation, such as one-hot encoding (Section 2.4.1), each graphic shape is represented by a specific attribute (entry) in the feature space, independent from each other (see the left figure of Figure 2.8). Classical machine learning algorithms, such as *decision trees* (Loh, 2011), *k-Nearest Neighbours* (*k-NN*, Cunningham and Delany, 2020), take this representation, split the input feature space into disjoint regions, and use the information of label y in each region to learn the predictor. For distributed representation, each graphic shape is represented by a composition of attributes, shared with each other (see the right figure of Figure 2.8). In practice, those attributes may not be explicitly defined and their values are learned either in a supervised way or in a self-supervised way (e. g. , *word2vec*, Mikolov, K. Chen, et al., 2013) depending on the data and the task.

Compared to non-distributed representation, distributed representation has several advantages. First, as the number of regions that it can represent is exponential with the dimension of feature (2^4 for this example), we can significantly reduce the dimension of input

5. How neural networks learn distributed representations: https://www.oreilly.com/content/how-neural-networks-learn-distributed-representations/

feature space, so that prevent the curse of dimensionality (Bellman, 1957, 1961). Secondly, unlike in non-distributed representation where the feature values has equal distance with each other, for distributed feature representations, their distances are not the same and can be deduced by comparing the corresponding feature vectors.

2.4 WORD REPRESENTATION

People expect intelligent machines to act in a way close to how humans perceive and process information in its natural form. Unfortunately, unlike humans, machines can only understand and treat data in numerical format. That is to say, non-numerical data, like text, must be transformed to numerical format in order to be treated by machines.

Text data is hierarchical and sequential in nature. Its smallest atom is the *character*. A sequence of characters forms a *word*, a sequence of words form a *sentence*, a sequence of sentences form a *document* and a set of documents form a *collection*. Depending on the nature of the task, the view of people and the methods used, text data can be considered and treated in different ways. For example, one can simply consider the word as the smallest atom of text instead of the character, or consider a document as a set of sentences instead of a sequence of sentences.

From now on, we denote character, word, document, collection as c, w, d, D, respectively. All the related notations are given in Table 2.2 which will serve the rest part of this dissertation.

In the following subsections, we mainly introduce different ways to transform words to vectors. We first introduce the basic *one-hot encoding* that transforms a word (more generally a categorical feature) to a sparse vector. Then we focus on *word embedding* methods that learn dense representations of words using unannotated corpus.

2.4.1 *One-hot encoding*

Given a vocabulary of words V of size v where each word w is identified by its index i, the *one-hot encoding* of w is an extremely sparse vector $\boldsymbol{o}_w \in \mathbb{Z}^v_{\{0,1\}}$ with only one non-zero entry on position i.

It is commonly used due to its simplicity where no special treatment is required. But it has three main limitations: sparsity, scalability, and information lost. 1) The sparsity is trivial as the one-hot encoded vector is full of zeros except at one position, which causes the sample-feature matrix (e. g. , the word-document matrix) to be extremely sparse. To overcome that, special storage formats are proposed to reduce the storage cost[6]. Also, classical algorithms (which originally

6. Sparse matrices formats used in SciPy library: https://docs.scipy.org/doc/scipy/reference/sparse.html

Table 2.2 – Notations used in this chapter.

Notation	Description
c	Character.
w	Word.
w_i	Word of position i in a word sequence.
w^i	Word with index i in word vocabulary.
d	Sentence/Paragraph/Document. A sequence of words $(w_1, \ldots, w_T)$ where T is the length of the sequence.
D	Collection. A set of sentences/paragraphs/documents.
n	Number of documents in collection D.
V	The vocabulary of all words appearing in D. All words are indexed.
v	Size of the vocabulary V.
$\boldsymbol{C} \in \mathbb{Z}_+^{v \times n}$	Word-Document occurrence matrix.
$\boldsymbol{Q} \in \mathbb{Z}_+^{v \times v}$	Word-Word co-occurrence matrix.
$\boldsymbol{o} \in \mathbb{Z}_{\{0,1\}}^{v}$	One-hot encoding vector of word.
$\boldsymbol{b} \in \mathbb{Z}_+^{v}$	Bag-of-Words (BoW) representation vector of document.
$\boldsymbol{t} \in \mathbb{Z}_+^{n}$	Document frequency representation vector of word.
$\boldsymbol{x} \in \mathbb{R}^{p}$	embedding vector of a term (word, n-gram, etc.)

take a dense vector as input) are modified to take only the non-zero values of a sparse input vector, in order to reduce computational cost (B. McMahan, 2011; H. B. McMahan et al., 2013). 2) Scalability issues appear when we want to use higher order features, especially when the vocabulary size v gets large. For example, n-gram method takes n consecutive words as features, the dimension of the feature space grows rapidly with respect to n. 3) The semantic similarity between words get ignored as the distance between any two one-hot encoded vectors is equivalent. However, words like "cat" and "dog" are similar at some degree (both are animal, pet), they should be closer in the vector space than the other non-relevant words with the distance reflecting their semantic closeness.

2.4.2 *Word embedding*

The distributional hypothesis in linguistics (Harris, 1954) suppose that **words which are used and occur in similar contexts tend to have similar meanings**. Based on this hypothesis, *word embedding* models try to learn a short, dense vector representation of words (often referred as *word vector* or *word embedding*) where the similarity between words is encoded in the vector space, compared to a long, sparse one-hot encoded one. In the following, we'll introduce several most famous and popular word embedding models: *LSA, GloVe, Word2vec, fastText, ELMo, BERT*, which can be generally separated to *Count-based* and *Prediction-based* (Z. Zhang, 2019).

2.4.2.1 *Count-based*

Count-based word embedding methods learn word vectors generally from analyzing a matrix which is related to word and its context's co-occurrence statistics. They share a common framework with 5 steps (Z. Zhang, 2019):

1. Preprocess the documents with tokenization, annotation, tagging, parsing, etc.
2. Construct a count-based information matrix with format of row $\times$ column as word $\times$ document, word $\times$ word, etc.
3. Transform and re-weight the matrix using *Term Frequency–Inverse Document Frequency* (*TF-IDF*, Sparck Jones, 1988), *Pairwise Mutual Information* (*PMI*, Church and Hanks, 2002), etc.
4. Factorize the transformed matrix using *Singular Value Decomposition* (*SVD*, Golub and Reinsch, 1970), *Non-negative Matrix Factorization* (*NMF*, D. Lee and Seung, 1999), etc.
5. Compare the produced vectors using distance measures such as euclidean, cosine.

In the following, we adopt and follow this framework to explain count-based word embedding methods, with step 1 ignored.

LSA *Latent Semantic Analysis* (*LSA*, Dumais, 2004), or *Latent Semantic Indexing* (*LSI*, Deerwester et al., 1990) in the domain of information retrieval, tries to project words and documents to a 'topic' space, by factorizing word-document occurrence matrix using Trucated SVD.

In step 2: We construct a word-document occurrence matrix $\boldsymbol{C} \in \mathbb{Z}_+^{v \times n}$ (v is the size of word vocabulary V, and n is the size of document collection D) where C_{ij} represents the times that the word w_i appears in document d_j.

In step 3: We keep the matrix $\boldsymbol{C}$ as it is in this step.

In step 4: We use Truncated SVD (Section 2.1.1) as dimension reduction technique. $\boldsymbol{C}$ is approximated by $\tilde{\boldsymbol{C}}$ with $\tilde{\boldsymbol{C}} = \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^\top$ where $\boldsymbol{U}_k \in \mathbb{R}^{v \times k}$ is the word-topic matrix, $\boldsymbol{V}_k \in \mathbb{R}^{n \times k}$ is the document-topic matrix and $\boldsymbol{\Sigma}_k \in \mathbb{R}^{k \times k}$.

In step 5: We denote the corresponding document frequency vector of a word w as $\boldsymbol{t} \in \mathbb{Z}_+^n$. To compare different words in the same low-dimensional space (topic space), we project $\boldsymbol{t}$ to $\Sigma_k^{-1} \boldsymbol{V}_k^T \boldsymbol{t}$. We then use task-specific distance to measure the similarity between projected word vectors.

LSA is easy to implement as only constructing a word-document occurrence matrix and running a Truncated SVD are required. It is effective in dealing with word similarity problems such as synonymy, polysemy (Rosario, 2001). However, it shows poor performance on word relation tasks such as word analogy (Turney, 2004).

GLOVE *Global Vectors* (*GloVe*, Pennington, Socher, and C. Manning, 2014) proposes to model the word-word co-occurrence through a log-bilinear regression model, which takes advantages of global corpus statistics and local context window information at the same time.

In step 2: We construct a word-word co-occurrence matrix $\boldsymbol{Q} \in \mathbb{Z}_+^{v \times v}$ where Q_{ij} represents how many times that word w^j appears in the context window of word w^i.

In step 3: We calculate the probability that word w^j (target word) appears in the context window of word w^i (context word), denoted as follows: $P_{ij} = P(w^j | w^i) = \frac{Q_{ij}}{Q_i}$, where $Q_i = \sum_{j=1}^{v} Q_{ij}$.

In step 4: The authors argue that the relationship between word w^i and w^j should not be directly represented by P_{ij}. Instead, they propose to examine it by studying the ratio of their co-occurrence using various auxiliary words, i. e. , by studying $\frac{P_{ik}}{P_{jk}}$ with different k. In order to have linear operation in the vector space, $\frac{P_{ik}}{P_{jk}}$ is modeled as:

$$\frac{P_{ik}}{P_{jk}} = \exp(\tilde{\boldsymbol{x_k}}^\top (\boldsymbol{x_i} - \boldsymbol{x_j})), \tag{2.40}$$

where $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ represent the embedding vector of target word and context word respectively. With some extra consideration of symmetry

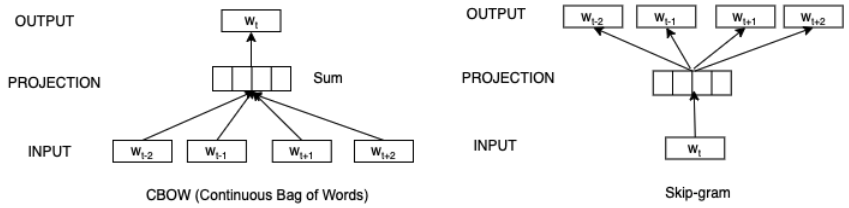


Figure 2.9 – Overview of Word2vec model architectures: CBOW and Skip-gram. A window size of 2 (two words on the left and two words on the right) is used in defining the surrounding words.

for swapping the role of context and target word, the model finally aims to minimize

$$\sum_{i,j=1}^{v} g(Q_{ij}) \left(\boldsymbol{x}_i^T \tilde{\boldsymbol{x}}_j + r_i + \tilde{r}_j - \log Q_{ij} \right)^2, \tag{2.41}$$

where g is a weight function which penalizes low frequency co-occurrence Q_{ij} due to lack of confidence, r_i and $\tilde{r}_j$ are both bias terms.

Thus, while exploiting statistical information, GloVe forces the model to learn linear relationship between words in the vector space which fill the weakness of LSA in word analogy tasks.

2.4.2.2 *Prediction-based*

Prediction-based word embedding models learn word vectors generally by solving a task of predicting a target word's occurrence given its context or vice versa.

WORD2VEC *Word2vec* (Mikolov, K. Chen, et al., 2013; Mikolov, Sutskever, et al., 2013) is one of the most famous and used word embedding methods. Two architectures are proposed: *continuous bag of words (CBOW)* and *skip-gram*. For CBOW, we predict the occurrence of the target word using its surrounding words. In reverse, skip-gram predicts the occurrences of the surrounding words given the target word. Figure 2.9 presents an overview of these two structures. As they are similar in design, we only present the skip-gram architecture in detail at following.

Given a center word w_t as input, skip-gram tries to predict the occurrence of its surrounding words $\{w_{t+j} | -c \leqslant j \leqslant c, j \neq 0\}$ as output, where c is the window size. To be more precise, the objective is to maximize

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leqslant j \leqslant c, j \neq 0} \log p(w_{t+j} | w_t), \tag{2.42}$$

where the conditional probability $p(w_O | w_I)$ is defined as the similarity score between the **output (context)** word w_O and the **input (target)**

word w_I, normalized by softmax over the whole word vocabulary, formulated as follows:

$$p(w_O|w_I) = \frac{\exp(s(\boldsymbol{x}_{w_I}, \boldsymbol{x}'_{w_O}))}{\sum_{w \in V} \exp(s(\boldsymbol{x}_{w_I}, \boldsymbol{x}'_{w}))}. \tag{2.43}$$

$\boldsymbol{x}_w$ and $\boldsymbol{x}'_w$ are the embedding vector of word w as input word and as output word respectively, V is the word vocabulary, s is the score function which measures the similarity between two word embedding vectors. Specifically, we take s as dot product function here, i. e. ,

$$s(\boldsymbol{x}_{w_I}, \boldsymbol{x}'_{w_O}) \triangleq \boldsymbol{x}_{w_I}{}^{\top} \boldsymbol{x}'_{w_O}. \tag{2.44}$$

Directly calculating the full softmax is costly, especially when vocabulary V is large. It can be calculated efficiently using *hierarchical softmax* (Morin and Yoshua Bengio, 2005) which uses a Huffman tree (Huffman, 1952) to reduce calculation or *negative sampling* (Mikolov, Sutskever, et al., 2013) which approximates it by sampling negative instances. According to the authors[7], hierarchical softmax works better for infrequent words while negative sampling works better for frequent words and better with low dimensional vectors. We describe the negative sampling strategy below.

Negative sampling approximates the multi-class classification task related to softmax by $k+1$ binary classification tasks. The goal changes from predicting the occurrence of one word in v words to distinguishing an input-output pair of words presented on the training data from k input-random pairs that are not ($v \gg k$). To be more precise, the objective term $\log p(w_O|w_I)$ in Equation 2.42 is replaced by

$$\log \sigma(s(\boldsymbol{x}_{w_I}, \boldsymbol{x}'_{w_O})) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-s(\boldsymbol{x}_{w_I}, \boldsymbol{x}'_{w_i})], \tag{2.45}$$

where k is the number of random words sampling from a given noise distribution P_n (e. g. , uniform distribution) and σ is the sigmoid function. Thus the objective of the new task is to distinguish the output word w_O from k random sampled noise words, given input word w_I.

FASTTEXT Word2vec methods treat each word completely individually which ignores word's morphological structure. To consider word's internal structure, *fastText* (Bojanowski et al., 2016) propose to represent a word as a bag of terms (character n-grams with $n \in [3, 6]$ and the word itself). Through learning the embedding vector for each term, we obtain the embedding of word as a sum of the embedding vectors of its terms. Specifically, in order to distinguish the same n-gram in different positions (prefix, middle of the word, suffix), special characters: < and >, are added at the beginning and at the end of each

7. Word2vec source code: https://code.google.com/archive/p/word2vec/

word respectively. Take the word *actor* and $n = 3$ as an example, it will be represented by the term set including its 3-grams: `<ac, act, cto, tor, or>` and itself (as a special character sequence): `<actor>`.

As regard of the learning process, we follow the Word2vec (Equation 2.42 and 2.43), while only changing the score function s (Equation 2.44) to

$$s(\boldsymbol{x}_{w_I}, \boldsymbol{x}'_{w_O}) \triangleq \sum_{g \in \mathcal{G}_{w_I}} \boldsymbol{x}_g{}^\top \boldsymbol{x}'_{w_O}, \tag{2.46}$$

where $\boldsymbol{x}_g$ is the embedding vector of term g in the word w_I's term set $\mathcal{G}_{w_I}$, $\boldsymbol{x}_{w_O}$ is the embedding vector of word w_O as a special character sequence. Thus, fastText can provide a reasonable guess of the unseen word's embedding by treating each word as a set of its n-grams, which can not be done by Word2vec.

2.4.3 *Contextual word embedding*

Previously presented word embedding methods learn a fixed vector for each word, however, one word may have different meanings depending on the context they are used in. For example, the word *right* in "a **right** choice" and "turn **right**" clearly have different meanings. Contextualized word embedding methods such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2018) try to overcome this issue by taking account the **contextual** information on the word's embedding. It makes the embedding vector of a word changeable depending on its context.

Contextualized word embedding is often packaged as a pre-trained *language model* on a large unannotated corpus. In the following part of this subsection, we introduce the general language model, and two popular language model-based word embedding methods (ELMo and BERT).

LANGUAGE MODEL (LM) Given a sentence $(w_1, \ldots, w_T)$, a forward *language model (LM)* decomposes the probability of the sentence as

$$p(w_1, w_2, \cdots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_1, w_2, \ldots, w_{t-1}), \tag{2.47}$$

while for a backward language model

$$p(w_1, w_2, \cdots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t+1}, w_{t+2}, \ldots, w_T). \tag{2.48}$$

Taking the forward LM as an example, the goal is to maximize the log-likelihood $log p(w_1, \ldots, w_T)$ where the conditional probability of present word w_t knowing its previous words $w_1, \cdots, w_{t-1}$ is commonly modeled and learned using neural networks (Yoshua Bengio et al., 2003), the embedding of words are produced during the learning process.

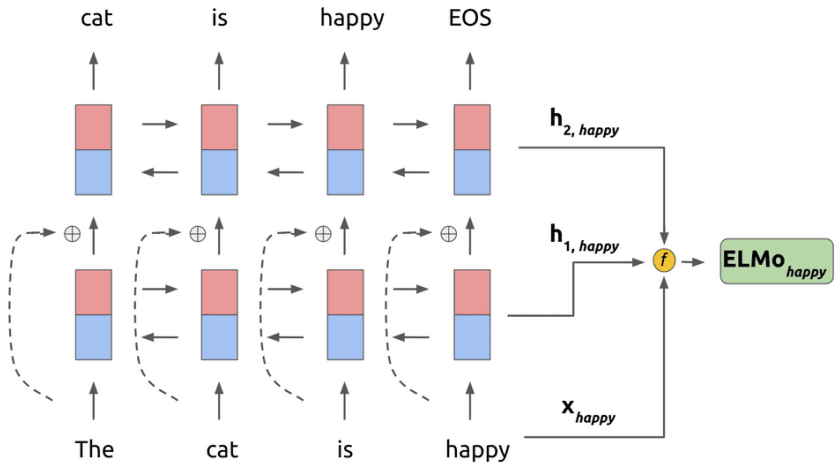


Figure 2.10 – Simple overview of ELMo's structure. Each red cell represents a forward LSTM cell and blue cell represents a backward LSTM cell. The ELMo embedding of each word is a weighted sum of its representation in each layer. Figure taken from Karan Purohit's blog [8].

ELMO The base of *Embeddings from Language Models* (*ELMo*, Peters et al., 2018) is a *bidirectional LM (biLM)*, which contains a forward pass and a backward pass at the same time in order to better capture the context information flows in both directions of the sequence, trained using unannotated corpus. The ELMo embedding is task-specific and is calculated as a weighted sum of all the intermediate embeddings produced from the pre-trained biLM, with the weights learned using task-specific data.

Precisely, the dependency between a present word and its context (preceding words for the forward LM and following words for the backward LM) is encoded using a stacked L-layer bidirectional LSTM (see Figure 2.10 with $L = 2$).

The objective is to maximize

$$\begin{aligned}\sum_{t=1}^{T}(&\log p(w_t \mid w_1, w_2, \ldots, w_{t-1}; \boldsymbol{\mathcal{W}}_x, \overrightarrow{\boldsymbol{\mathcal{W}}}_{LSTM}, \boldsymbol{\mathcal{W}}_s) \\ &+ \log p(w_t \mid w_{t+1}, w_{t+2}, \ldots, w_T; \boldsymbol{\mathcal{W}}_x, \overleftarrow{\boldsymbol{\mathcal{W}}}_{LSTM}, \boldsymbol{\mathcal{W}}_s)),\end{aligned} \tag{2.49}$$

where $\boldsymbol{\mathcal{W}}_x, \overrightarrow{\boldsymbol{\mathcal{W}}}_{LSTM}, \overleftarrow{\boldsymbol{\mathcal{W}}}_{LSTM}, \boldsymbol{\mathcal{W}}_s$ are the parameters of the embedding layer, the forward LSTM layers, the backward LSTM layers, the output layer respectively, learned jointly using unannotated corpus.

For each word w_t, we denote its representation produced by the embedding layer as $\boldsymbol{x}_t$, its representation produced by the bidirectional LSTM (hidden state of forward and backward LSTM) at l-th layer as

8. Learn contextual word embeddings with ELMo: https://medium.com/saarthi-ai/elmo-for-contextual-word-embedding-for-text-classification-24c9693b0045

$\boldsymbol{h}_{t,l} = [\overleftarrow{\boldsymbol{h}}_{t,l}, \overrightarrow{\boldsymbol{h}}_{t,l}]$. In total, we have $2L+1$ representations of w_t, denoted as

$$\begin{aligned} R_t &= \{\boldsymbol{x}_t, \overleftarrow{\boldsymbol{h}}_{t,l}, \overrightarrow{\boldsymbol{h}}_{t,l} | l = 1, \cdots, L\} \\ &= \{\boldsymbol{h}_{t,l} | l = 0, \cdots, L\}. \end{aligned} \tag{2.50}$$

For a supervised downstream task, given the pre-trained biLM, the task-specific ELMo embedding $\mathbf{ELMo}_t^{task}$ of a word w_t is a weighted sum of its different representations in R_t, described as follows:

$$\mathbf{ELMo}_t^{task} = \gamma_{task} \sum_{l=0}^{L} \boldsymbol{s}_l^{task} \boldsymbol{h}_{t,l}, \tag{2.51}$$

where the weight vectors $\boldsymbol{s}^{task}$ (normalized by softmax) and the scale factor γ_{task} are both learned during the training of the downstream task.

BERT Unlike ELMo which uses a stacked multi-layer bidirectional LSTM to encode contextual information in the sentence, *Bidirectional Encoder Representations from Transformers* (*BERT*, Devlin et al., 2018) use *Transformers* (Vaswani et al., 2017) based on *self-attention* mechanism which largely improve state-of-the-art results in many NLP tasks (Devlin et al., 2018).

Transformers are widely used in *sequence-to-sequence (seq2seq)* tasks, such as machine translation (Junczys-Dowmunt, 2019; Xiaodong Liu et al., 2020), text summarization (P. J. Liu et al., 2018; Radford et al., 2019), question answering (Shao et al., 2019), where we are asked to transform a source sequence to a target sequence (both can be of arbitrary length). In the following discussion, we denote the source sequence as $(x_1, \cdots, x_T)$ and the target sequence as $(y_1, \cdots, y_{T'})$ where T, T' are the sequence lengths.

For seq2seq tasks, *encoder-decoder* framework are widely adopted. Generally speaking, the encoder first compresses the source sentence to **a fixed length** vector $\boldsymbol{c}$, often referred as *context vector*, as a summary of the sentence. Then the decoder uses this compressed vector to produce the target sentence. A common example of encoder-decoder framework is shown in Figure 2.11 where we aim to transform the source sequence "ABC" to the target sequence "WXYZ". The encoder uses its last hidden state, corresponding to the input token "C, as the context vector. During the training phase, the decoder takes the shifted target sequence "<EOS>WXYZ" (shifted one position to the left side) as input, with the help of the context vector, it tries to output the target sequence "WXYZ<EOS>", where <EOS> represents the special token indicating the beginning or the ending of the sequence. The embedding of tokens and the encoder, decoder parameters are trained jointly using source-target sequence pairs. During the test (inference) phase, taking the context vector produced by encoder as initial value of the decoder's hidden state, the decoder produces the output tokens

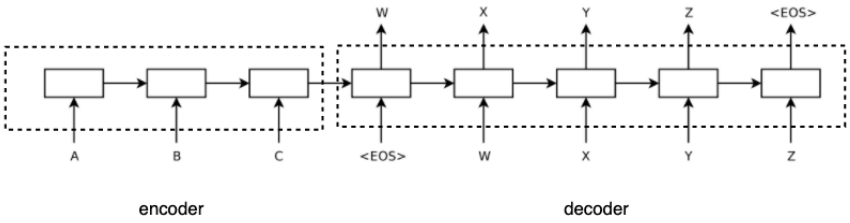


Figure 2.11 – A simple encoder-decoder framework example where we aim to transform the source sequence "XYZ" to the target sequence "WXYZ". <EOS> represents the special token indicating the beginning or the end of the sequence. Figure taken and modified from Sutskever, Vinyals, and Q. V. Le, 2014.

iteratively and continuously, until the <EOS> token is outputted (or the maximum allowed length for the sequence is attained in practice).

RNN based encoder and decoder: Sutskever, Vinyals, and Q. V. Le, 2014 suggest to use (multi-layer) RNN (LSTM) as both encoder and decoder. We denote the hidden states of RNN encoder as $(\boldsymbol{h}_1, \cdots, \boldsymbol{h}_T)$, the hidden states of RNN decoder as $(\boldsymbol{s}_1, \cdots, \boldsymbol{s}_{T'})$. Under this setting, the context vector c is constructed as follows:

$$\boldsymbol{c} = m(\boldsymbol{h}_1, \cdots, \boldsymbol{h}_T), \tag{2.52}$$

where m is a aggregation function that merges all hidden states of encoder $(\{\boldsymbol{h}_t\}_{t=1}^T)$ to one single vector. Quite often, the last hidden state $\boldsymbol{h}_T$ of the RNN encoder is served as the context vector (Sutskever, Vinyals, and Q. V. Le, 2014), i. e. ,

$$m(\boldsymbol{h}_1, \cdots, \boldsymbol{h}_T) = \boldsymbol{h}_T. \tag{2.53}$$

Then the RNN decoder is initialized with this context vector ($\boldsymbol{s}_1 = \boldsymbol{c}$) and outputs the distribution of each target word through a dense layer with softmax activation function.

RNN based encoder and decoder with attention: Although using LSTM reduces the issues of memorizing long term dependencies compared to standard RNN, common RNN based approaches are still prone to have information loss as the entire sentence is compressed to one single context vector. Bahdanau, Cho, and Yoshua Bengio, 2016 resolve this issue by introducing an *attention* mechanism. Under this mechanism, when producing each output word $y_{t'}$ in the target sentence, the decoder looks at the **entire** source sentence with different attention to create a **dynamic** context vector $\boldsymbol{c}_{t'}$ (dynamic representation of the source sentence), instead of a fixed one. $\boldsymbol{c}_{t'}$ is calculated as a weighted sum of all the hidden states in the source sentence produced by the encoder, as follows:

$$\boldsymbol{c}_{t'} = \sum_{t=1}^{T} \alpha_{t't} \boldsymbol{h}_t, \tag{2.54}$$

where the weight score $\alpha_{t't}$ represents the degree of importance of input word x_t to produce current output word y'_t. $\alpha_{t't}$ is actually calculated as an alignment score between the decoder's previous state $\boldsymbol{s}_{t'-1}$ and the encoder's hidden state $\boldsymbol{h}_t$ (**softmax normalized over** t), defined as follows:

$$\begin{aligned}\alpha_{t't} &= align(\boldsymbol{s}_{t'-1}, \boldsymbol{h}_t) \\ &\triangleq \frac{\exp(score(\boldsymbol{s}_{t'-1}, \boldsymbol{h}_t))}{\sum_{i=1}^{T} \exp(score(\boldsymbol{s}_{t'-1}, \boldsymbol{h}_i))},\end{aligned} \tag{2.55}$$

where different formulas of *score* function are proposed in the literatures (Alex Graves, Wayne, and Danihelka, 2014; Luong, H. Pham, and C. D. Manning, 2015; Bahdanau, Cho, and Yoshua Bengio, 2016; Vaswani et al., 2017). Here the authors propose to use a FFNN with one hidden layer (known as *Additive attention*) to present the *score* function.

Attention-based Transformer: Following the same idea but slightly different from the traditional attention mechanism, *self-attention* (J. Cheng, Dong, and Lapata, 2016; Zichao Yang et al., 2016; Vaswani et al., 2017), also known as *intra-attention*, is adopted, in the case where only the source sequence is involved in the model (i. e., no target sequence). The goal is to have an adjustable summary for the source sequence when looking at different positions of it. For any position $t' \in [1, T]$ in the source sequence, the corresponding summary of the sequence is formulated as follows:

$$\boldsymbol{c}_{t'} = \sum_{t=1}^{T} align(\boldsymbol{x}_{t'}\boldsymbol{W}^q, \boldsymbol{x}_t\boldsymbol{W}^k)\boldsymbol{x}_t\boldsymbol{W}^v, \tag{2.56}$$

where $\boldsymbol{W}^q$,$\boldsymbol{W}^k$,$\boldsymbol{W}^v$ are the transformation matrices that applied on the original sequence $\{\boldsymbol{x}_t\}_{t=1}^{T}$, learned during the training.

Actually, aforementioned two attention mechanisms can be generalized to a *query-key-value* ($\boldsymbol{q}$-$\boldsymbol{k}$-$\boldsymbol{v}$) architecture where based on a list of key-value vector pairs, a dynamic value vector for a given query is formed, by calculating the alignment score between the query and each key in the list, then taking the weighted sum of the corresponding values using the alignment scores as weights. Thus, the value vector $\boldsymbol{c}_{t'}$ for a query $\boldsymbol{q}_{t'}$ is calculated as follows:

$$\begin{aligned}\boldsymbol{c}_{t'} &= Attention(\boldsymbol{q}_{t'}, \{\boldsymbol{k_t}\}, \{\boldsymbol{v_t}\}) \\ &\triangleq \sum_{t=1}^{T} align(\boldsymbol{q}_{t'}, \boldsymbol{k}_t)\boldsymbol{v}_t \\ &= \sum_{t=1}^{T} \frac{exp(score(\boldsymbol{q}_{t'}, \boldsymbol{k}_t))}{\sum_{i=1}^{T} exp(score(\boldsymbol{q}_{t'}, \boldsymbol{k}_i))}\boldsymbol{v}_t\end{aligned} \tag{2.57}$$

where $\boldsymbol{q}_{t'}$, $\{\boldsymbol{k_t}\}$,$\{\boldsymbol{v_t}\}$ are the query, the keys, the values, respectively, *score* is a score function that measures the similarity between a query and a key.

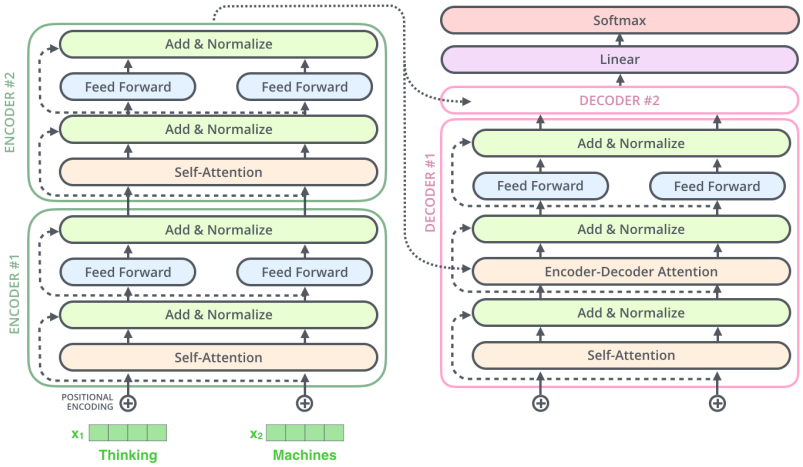


Figure 2.12 – Overview of the Transformer architecture where the encoding (resp. decoding) part has two encoders (resp. decoders). Figure taken from Jalammar's blog [9].

Specially we have

$$\boldsymbol{q}_{t'} = \boldsymbol{s}_{t'-1}, \boldsymbol{k}_t = \boldsymbol{h}_t, \boldsymbol{v}_t = \boldsymbol{h}_t, \tag{2.58}$$

for the standard attention mechanism (Equation 2.54 and Equation 2.55) and

$$\boldsymbol{q}_{t'} = \boldsymbol{x}_{t'}\boldsymbol{W}^q, \boldsymbol{k}_t = \boldsymbol{x}_t\boldsymbol{W}^k, \boldsymbol{v}_t = \boldsymbol{x}_t\boldsymbol{W}^v, \tag{2.59}$$

for the self-attention mechanism (Equation 2.56).

Transformer architecture (Vaswani et al., 2017) takes advantages of both attention mechanisms where the input of encoder and decoder are both transformed through a self-attention layer before feeding to the next layers, and the decoder takes the encoder's information using standard attention layer (Figure 2.12). Specifically, for generating the context vector, *Scaled Dot-Product Attention* is used here where the *score* function is defined as follows:

$$score(\boldsymbol{q}_{t'}, \boldsymbol{k}_t) = \frac{\boldsymbol{q}_{t'}^\top \boldsymbol{k}_t}{\sqrt{d_k}}, \tag{2.60}$$

where d_k is the dimension of the key and the query vector.

In fact, in order to further capture different subspace information, *Multi-Head (Self) Attention* is applied, where the query, key, value vectors are created respectively through multiplying a separate matrix for each head, compared to a single head attention as shown in Equation 2.56. Then the produced context vector of each head are concatenated and transformed again to form the final representation of the input. i. e. ,

$$\begin{aligned} &MultiHead(\boldsymbol{q}_{t'}, \{\boldsymbol{k}_t\}, \{\boldsymbol{v}_t\}) = Concat(head_1, \cdots, head_h)\boldsymbol{W}^o \\ &\text{where } head_i = Attention(\boldsymbol{q}_{t'}\boldsymbol{W}_i^q, \{\boldsymbol{k}_t\boldsymbol{W}_i^k\}, \{\boldsymbol{v}_t\boldsymbol{W}_i^v\}) \end{aligned} \tag{2.61}$$

9. The Illustrated Transformer: http://jalammar.github.io/illustrated-transformer/

with $\boldsymbol{W}_i^q, \boldsymbol{W}_i^k, \boldsymbol{W}_i^v$ the transformation matrix for query, key, value of head i respectively, and $\boldsymbol{W}^o$ the final transformation matrix.

There are two details need to be mentioned when applying attention mechanism in Transformer. 1) As in attention mechanism, the sequential aspect of the input is ignored. A fixed or learned *positional-embedding* is added to the embedding of each input (see left bottom of Fig. 2.12), which gives the model the information about the position of the current input in the sequence. The authors (Vaswani et al., 2017) propose to use a fixed embedding vector (denoted as PE) constructed by sine and cosine functions of different frequencies, defined as follows:

$$\begin{aligned} PE(pos, 2i) &= sin(pos/10000^{2i/d}), \\ PE(pos, 2i+1) &= cos(pos/10000^{2i/d}), \end{aligned} \tag{2.62}$$

where pos is the position of the input in the input sequence, i is the position of the entry in the positional embedding vector, d is the dimension of the positional embedding vector. 2) For self-attention mechanism, each position in the sequence can access the whole sequence. The first (Multi-Head) self-attention layer of the decoder (see bottom right of Fig. 2.12), is modified to a *Masked (Multi-Head) self-attention* layer, in order to prevent the current position from seeing its following positions. The mask is done by setting the value of the *score* function (Eq. 2.60) directly to $-\infty$ of all illegal connections between two positions.

Transformer-based BERT: *BERT* (Devlin et al., 2018) is basically a stack of **Transformer encoders** which is pre-trained through a *Masked Language Model (MLM)* task and a *Next Sentence Prediction (NSP)* task. It is fine-tuned to serve specific downstream tasks without changing the architecture (except for the output layers, see Figure 2.13 for an overview).

Concerning the pre-training of BERT, our first task is a Masked Language Model (MLM) where the goal is to learn a good contextual representation of each word. Traditional language models (LMs) (Section 2.4.3) don't allow to have multi-layer bidirectional blocks because in higher layer, each word can actually 'see' itself which makes the target word prediction task trivial to solve. Actually, ELMo's bidirectional LSTM is still unidirectional where the information flows either forward or backward, compared to self-attention which is truly bidirectional. BERT resolves this issue by using a MLM where 15% of the input is masked at random by a special `<MASK>` token and then the model predicts these masked words just like in traditional LM. In fact, in the fine-tuning step of downstream tasks, `<MASK>` token will never appear. In order to mitigate this mismatch between pre-training and fine-tuning, for the word chosen to be masked, it is replaced with the `<Mask>` token for 80% of the time, with a random word for 10% of the time and with itself for the rest 10% of the time.

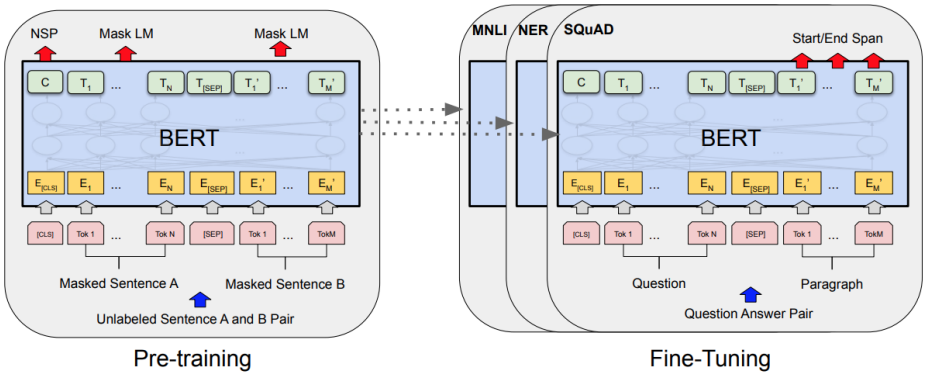


Figure 2.13 – Overview of BERT's pre-training and find-tuning procedure. Pre-training and fine-tuning share the same architecture, except for the output layers. During fine-tuning, all parameters get fine-tuned. `[CLS]`, `[SEP]` are special tokens where `[CLS]` is added at the beginning of each sentence and `[SEP]` serves as a separator of two sentences. Figure taken from Devlin et al., 2018.

Our second task is next sentence prediction (NSP) task where the goal is to understand the relationship between sentences, which plays an important role in solving *Question Answering (QA)* and *Natural Language Inference (NLI)* tasks. In this NSP task, for a given sentence pair (A, B), we want to predict if B is the next sentence of A. When constructing the training pairs, we let B follow A for 50% of the time.

It's worth to mention that for each input word, except for its embedding vector, a positional embedding vector (indicating the position of the word in the input sequence) and a *segment embedding* vector (indicating which sentence it belongs to) are summed together, to feed BERT. Different from Transformer, the positional embedding in BERT is trainable.

After the pre-training process, for downstream tasks, we simply plug BERT with specific output layers and fine-tune all or part of the parameters using task-specific data. Depending on the type of tasks, different parts of the representations produced by BERT are used. For example, for sentence-level tasks such as sentence classification, only the representation of the `[CLS]` token is used as a summary of the entire sequence. For token-level tasks such as sequence tagging, the contextual representation of each word is used.

2.5 DOCUMENT EMBEDDING

Document embedding models often consider the document as a flatten sequence of words while discarding its sub-level paragraph or sentence structure. From now on, we refer the term "document" to a sequence of words in general, ranging from sentence, paragraph

to document. Under this setting, many applications and tasks, such as sentiment analysis, document classification, duplicate questions identification [10], are document-level problems.

As regard of document embedding models, while supervised model structures exist, our focus here is *self-supervised* (unsupervised) approaches that learn a semantically meaningful representation of the document which can be used directly in downstream tasks. These self-supervised approaches are often an extension of word embedding techniques presented in Section 2.4.2. We separate these document embedding approaches to two categories: those which *aggregate pre-trained word embeddings* and those which *produce directly the document embedding*. In the following discussion, we consider the document d as a sequence of words $(w_1, \ldots, w_T)$ where T is the length of the document.

2.5.1 *Aggregate pre-trained word embeddings*

For approaches in this category, we assume that we have already the vector representation of each word $\{\boldsymbol{x}_w | w \in V\}$ at hand. The word representation can be produced by various word embedding models, contextual or not, such as LSA, Word2vec, fastText, BERT (Section 2.4.2), or simply by using one-hot encoding. The document's representation vector $\boldsymbol{x}_d$ can be simply calculated as an average of all the word vectors in the document, either in a uniform manner like *Bag-of-Words* (*BoW*, C. D. Manning, Raghavan, and Schütze, 2008) or in a weighed manner like *Term Frequency–Inverse Document Frequency* (*TF-IDF*, Sparck Jones, 1988). Thus, in a general form, $\boldsymbol{x}_d$ is formulated as:

$$\boldsymbol{x}_d = \frac{1}{T} \sum_{w \in d} g(w, d, D) \boldsymbol{x}_w \tag{2.63}$$

where $g(w, d, D)$ is a weight function depending on the word w, the document d and the document collection D.

BAG-OF-WORDS Regarding the Bag-of-Words (BoW) representation, the document is treated simply as a bag of its words where the order of words gets ignored but their frequencies are kept. Precisely, we have

$$g(w, d, D) = c_w, \tag{2.64}$$

where c_w represents the raw count of word w in document d. It is widely used as a simple and easy to implement method which converts a variable-length word sequence into a fixed-length vector.

TERM FREQUENCY–INVERSE DOCUMENT FREQUENCY Other than the raw count which takes only the information of w in the document

10. Quora Question Pairs competition: https://www.kaggle.com/c/quora-question-pairs

d, the weight function $g(w, d, D)$ can have other forms that involve information in the whole collection D. TF-IDF is a commonly used form, where TF stands for *Term Frequency* (Luhn, 1957) and IDF stands for *Inverse Document Frequency*. The TF-IDF weight function is formulated as follows:

$$g(w, d, D) = TF(w, d) \times IDF(w, D), \tag{2.65}$$

with

$$IDF(w, D) = log \frac{|D|}{|\{d \in D : w \in d\}|}, \tag{2.66}$$

where $TF(w, d)$ is the term frequency of word w in d, $|D|$ is the number of documents in the collection D and $|\{d \in D : w \in d\}|$ represents the number of documents in the collection that contain the word w. For $TF(w, d)$, various choices can be made. It can be the raw count c_w, logarithm of the raw count $log(1 + c_w)$ in order to lower the impact of extremely frequent word, etc. For $IDF(w, D)$, the intuition is that if a word appears commonly in the collection (e. g. , *stop-words*: 'the', 'is', etc.), it does not provide valuable information to represent the document, which leads to a small IDF weight. Conversely, if a word appears rarely in the collection (e. g. , appears only in the document d), it provides a lot of information, which leads to a large IDF weight. By combing the TF and IDF weights together, the TF-IDF weight takes both the local and the global importance of words, which generally conducts better results than simple BoW.

2.5.2 *Produce directly the document embedding*

Similar to word embeddings (Section 2.4.2), document embedding approaches of present category can further be divided into two main categories: *count-based* and *prediction-based*.

For count-based approaches, we refer to models that extract document embeddings based on corpus statistics, such as LSA (Dumais, 2004) in Section 2.4.2.1 which factorizes the word-document occurrence matrix as a product of word-topic and topic-document matrix, where the embedding of words and documents are obtained at the same time and both are on the topic space.

For prediction-based approaches, we refer to models that directly extend prediction-based word embedding models in Section 2.4.2.2, where the context information used to predict the target word contains the document embedding vector. The document embedding is trained jointly with the word embeddings in a self-supervised way. In the following, we introduce *Paragraph Vector* (*Doc2vec*, Q. Le and Mikolov, 2014) model and *Document Vector through Corruption* (*Doc2vecC*, M. Chen, 2017) model which we mainly use in our work.

DOC2VEC Word2vec (Mikolov, K. Chen, et al., 2013) model has two architectures (Figure 2.9), CBOW and skip-gram, which are similar in

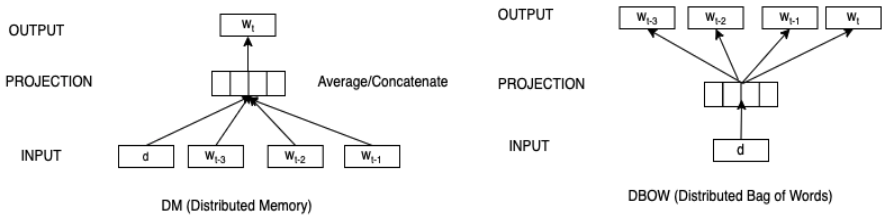


Figure 2.14 – Overview of the Doc2vec model structure. d, w represent the document, the word, respectively.

design. CBOW uses surrounding words as contextual information to predict center word and skip-gram uses center word as contextual information to predict surrounding words (see Section 2.4.2.2 for details). For Doc2vec model, similar designs are inherited (Figure 2.14), where the *Distributed Memory (DM)* and *Distributed Bag of Words (DBOW)* architectures are analogy to CBOW and skip-gram of Word2vec, respectively. As they are similar in design, we only introduce the DBOW architecture in detail.

For DBOW version of Doc2vec model, the embedding vector of document is used as contextual information to predict the occurrence of the words in the document. To be more precise, given a document d, represented as a sequence of words $(w_1, \cdots, w_T)$, the objective is to maximize

$$\frac{1}{T}\sum_{t=1}^{T}\log p(w_{t+j}|d) \tag{2.67}$$

where the conditional probability $p(w_O|d)$ is defined as follows:

$$p(w_O|d) \triangleq \frac{\exp({\boldsymbol{x}_d}^\top \boldsymbol{x}'_{w_O})}{\sum_{w\in V}\exp({\boldsymbol{x}_d}^\top \boldsymbol{x}'_{w})}\,. \tag{2.68}$$

$\boldsymbol{x}'_{w_O}$ and $\boldsymbol{x}_d$ are the embedding vectors of the output (context) word w_O and of the document d respectively. At the training stage, the embedding of words and the embedding of document are jointly learned using an unannotated document corpus. At the inference stage, the pre-trained embedding vectors of words are kept fixed, the embedding vectors of unseen documents get trained accordingly. In the end, the authors use a concatenation of the embedding vectors obtain by DM and DBOW as the final embedding vector of the document.

DOC2VECC Although Doc2vec model improves results on several text classification and sentiment analysis tasks (Q. Le and Mikolov, 2014), however, its main limitation is that in the inference phase, a non-trival training procedure for producing the document's embedding vector is involved. It contains sufficient steps of updating the document's embedding vector, which is costly when we have large number of documents to infer. *Doc2vecC* model (M. Chen, 2017) mitigates this by representing each document as a simple average of embedding vectors of its words. Thus, in the inference phase, no training

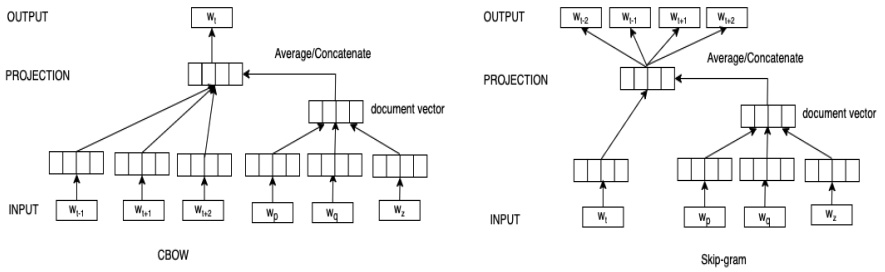


Figure 2.15 – Overview of the Doc2vecC model structure.

is needed. Again, for Doc2vecC model, we have two architectures analogy to CBOW and skip-gram respectively. An overview of these two architectures can be seen in Figure 2.15.

Let's take the skip-gram version of Doc2vecC for more details. Unlike Doc2vec model which always uses the whole document d as context information to predict the output words, Doc2vecC uses a dynamic one. A sampled document d_t is produced for every position t, by removing each word of the document with a probability q. Our objective is to maximize

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leqslant j\leqslant c, j\neq 0}\log p(w_{t+j}|d_t, w_t), \tag{2.69}$$

where the conditional probability $p(w_O|d_t, w_t)$ is defined as follows:

$$p(w_O|d_t, w_t) \triangleq \frac{\exp((\boldsymbol{x}_{d_t}+\boldsymbol{x}_{w_t})^\top \boldsymbol{x}'_{w_O})}{\sum_{w\in V}\exp((\boldsymbol{x}_{d_t}+\boldsymbol{x}_{w_t})^\top \boldsymbol{x}'_{w})}, \tag{2.70}$$

with

$$\boldsymbol{x}_{d_t} = \frac{\sum_{w\in d_t}\boldsymbol{x}_w}{(1-q)\cdot T} \tag{2.71}$$

$\boldsymbol{x}'_w$, $\boldsymbol{x}_w$ refer to the embedding vectors of the word w as output word and as input word respectively, V is the word vocabulary and q is the probability that we drop each word when constructing the sampled document.

At the inference stage, the representation of a document d is simply calculated as the average of the embeddings of its words, i. e. ,

$$\boldsymbol{x}_d = \frac{1}{T}\sum_{w\in d}\boldsymbol{x}_{\boldsymbol{w}}, \tag{2.72}$$

which can be calculated easily and efficiently.

3

PREDICTING CONVERSIONS IN DISPLAY ADVERTISING BASED ON URL EMBEDDINGS

In online display advertising (J. Wang, W. Zhang, and S. Yuan, 2017), advertisers promote their products by embedding ads on the publisher's web page. The majority of all these online display ads are served through *Real-Time Bidding* (*RTB*) (Google, 2011). RTB allows the publishers to sell their ad placements via the *Supply-Side Platform* (*SSP*) and the advertisers to purchase these via the *Demand-Side Platform* (*DSP*). More specifically, each time a user visits a website that contains a banner placement, an auction is triggered. The publisher sends user's information to the SSP, which forwards this information to the *Ad exchange* (*AdX*), and finally the AdX sends a bid request to the DSPs. Then each DSP decides if it will submit or not a bid response for this impression, based on its information about user, advertisement, urls, etc. Once the DSPs send back to the AdX their bids, a public auction takes place with the impression to be sold to the highest bidder. Figure 3.1 briefly illustrates the procedure of online display advertising.

DSPs are agent platforms that help advertisers optimize their advertising strategies. Roughly speaking, DSPs try to estimate the optimal bid price for an ad impression in order to maximize the audience of the campaigns of their advertisers, given some budget constraints. The bid price of an ad impression is highly related to the additive value that this impression could have on the advertising campaign (i.e., the number of ad impressions, clicks or conversions, etc.). In this context, advertisers have at their disposal a number of different pricing models. In the case where the objective of the advertisers is to maximize the exposure of their advertising message to a targeted audience, paying per impression, referred as *cost-per-mille* (CPM), is probably the best option for them. Nevertheless, in most of the cases, performance display advertising is more attractive to advertisers that are interested in accomplishing specific goals reducing their risks. In this case, advertisers are willing to pay for an ad impression if and only if that impression will drive the user to take a predefined action (Mahdian and Tomak, 2007), such as a visit on the advertiser's website, a purchase of a product, etc. Two performance payment models have been introduced for this purpose, referred as *cost-per-click* (CPC) and *cost-per-action* (CPA).

In performance-driven display advertising, DSPs submit a bid for a given ad impression based on the CPC or CPA that the advertiser is willing to pay. To determine the optimal bid price for an ad impres-

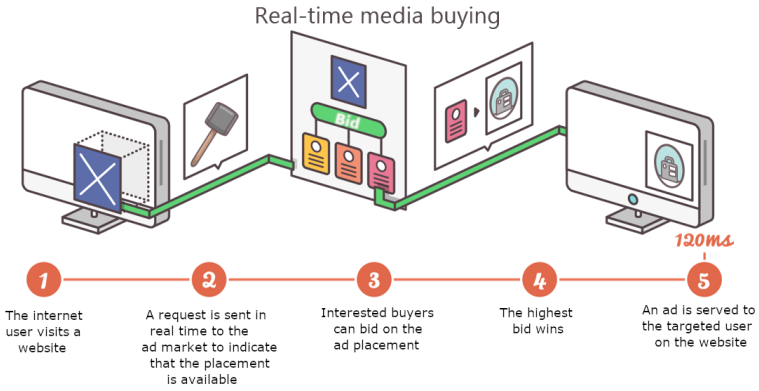


Figure 3.1 – A high-level overview of RTB procedure.

sion, DSPs estimate the *expected cost per impression*, called eCPI, which is either equal to the *click-through-rate* (*CTR*) for this impression multiplied by the value of CPC, or the *conversion rate* (*CVR*) multiplied by the value of CPA (Y. Chen et al., 2011). As a result, accurate CTR/CVR prediction plays an important role in the success of online advertising. For this purpose, DSPs build CTR/CVR prediction models able to estimate the probability a user converting after their exposure to an advertisement. The accuracy of these models is of high importance for the success of the campaign as if we overestimate click or conversation rates, we will probably submit quite higher bids than we should do, winning possible useless ad impressions. On the other hand, if these rates are underestimated, we will probably miss ad impressions likely to lead to a user action.

In the work presented on this chapter, we examine the user conversion problem, where given an advertiser, we want to predict if a user will convert or not the next day. In contrast to previous works that use a number of features related to the user profile, ad information and context information, we consider only the user's browsing history. More specifically, each user is represented as a sequence of URLs visited by him in a single day. Therefore, the problem examined in this work can be formally described as: **given a user's sequence of URLs from a single day, predict the probability this user to take a predefined action on the next day**. In our case, a user is considered as *converted* if they visit the advertiser's website. Due to the high cardinality and diversity of URLs, a compact semantically meaningful representation of URLs is of high importance. For this purpose, we build and examine three URL embedding models following the idea of *word embeddings* (Mikolov, Sutskever, et al., 2013). The sequential dependency between URLs in the user's browsing history has also been considered by using a *Recurrent Neural Network* (*RNN*, A. Graves, 2012). In total, ten different prediction conversion models have been introduced. A number of large scale experiments have been executed on a data collected from a real-world advertising platform in order to reveal and compare the prediction abilities of the proposed prediction schemes. Finally, our empirical analysis validates our claims about the

effectiveness of our representation models showing that they achieve to group together URLs of the same category. It means that URLs with the same or similar context are also close on the embedding space.

3.1 RELATED WORK

As the performance of a campaign is directly related on how precisely the CVR/CTR is estimated, it has been the objective of considerable research in the past few years. Typically, the problem of CVR/CTR estimation is formulated as a standard binary classification problem. Logistic regression has been extensively used to accurately identify conversion events (M. Richardson, Dominowska, and Ragno, 2007; H. B. McMahan et al., 2013; O. Chapelle, E. Manavoglu, and R. Rosales, 2014). Graepel et al., 2010 introduced a Bayesian learning model, called Bayesian probit regression, which quantifies the uncertainty over a model's parameters and hence about the CTR of a given ad-impression. A precise user representation (a set of features describing user behaviour) constitutes also the foundation for building a linear model able to estimate CVR with high accuracy. Nevertheless, in most cases it requires a lot of feature engineering effort and the knowledge of the domain. Moreover, linear models are not capable to reveal the relationship among feature. To overcome this problem, models such as *Factorization Machines* (*FM*, R. J. Oentaryo et al., 2014; Ta, 2015) and *Gradient Boosted Regression Trees* (*GBRT*,X. He et al., 2014) have been also proposed to capture higher order information among features. A number of different deep learning methods have been also proposed recently for CTR prediction (Y. Zhang et al., 2014; Q. Liu et al., 2015; Chan et al., 2018; G. Zhou et al., 2018).

Representation learning has been applied with success in several applications and has become a field in itself (Y. Bengio, Courville, and Vincent, 2013) in the recent years. The URL representation architectures presented in this manuscript have been inspired by those used in *Natural Language Processing* (*NLP*) tasks. Learning high-quality representations of phrases or documents is a long-standing problem in a wide range of NLP tasks. WORD2VEC (Mikolov, Sutskever, et al., 2013) is one of the most well-known word embeddings algorithms. The main idea behind WORD2VEC is that words that appear in similar contexts should be close in the learned embedding space. For this purpose, a (shallow) neural network model is applied that consists of an input, a projection, and an output layer. Its simple architecture makes the training extremely efficient. Grbovic et al., 2016 proposed SEARCH2VEC model that learns user search action representations based on contextual co-occurrence in user search sessions. URLNET (H. Le et al., 2018) is a well-known work that learns a URL representation but for the task of malicious URL detection. In contrast to our self-supervised representation scheme, URLNet is an end-to-end (supervised) deep

learning framework where its character-level and word-level CNNs are jointly optimized to learn the prediction model.

3.2 PROPOSED CONVERSION PREDICTION ARCHITECTURE

The goal of this work is to predict the probability that a user will convert one day after, given their browsing history on a single day. More specifically, we consider each user as an ordered sequence of URLs, sorted chronologically. The notion of conversion corresponds to an action of interest for the advertiser, such as visit on the landing page, purchase of a product, registration, etc. Therefore, we can treat the problem of predicting the user conversion as a binary classification problem (Bishop, 2006), where given a sequence of URLs visited by a user

$$u_n = [url_1^n, \ldots, url_{T_n}^n], n = 1, 2, \ldots, N, \tag{3.1}$$

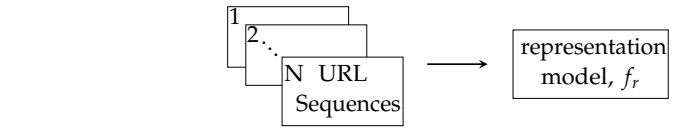
we want to predict if u_n will be converted or not (with label $y_n \in \{0, 1\}$). The length of the URL sequence, T_n, may be different for each user.

As an analogy to text classification, we view a sequence of URLs as a document, or a sequence of sentences. In our case, a URL is itself a sequence of tokens, of length **at most three** (we ignore the rest tokens as they are quite noisy). Each URL [1] is split with a '/' (slash) character, where the first token corresponds to the domain name. For instance, `https://en.wikipedia.org/wiki/Main_Page` is mapped to [`en.wikipedia.org`, `wiki`, `Main_Page`].

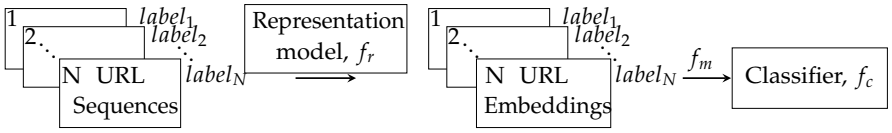
In order to apply any supervised classification model, such as logistic regression, etc., a semantically meaningful representation of each URL is needed. Therefore, a key intermediate step in our model is the learning of a URL representation. More precisely, the proposed conversion prediction scheme is composed of two consecutive training phases. The first one corresponds to the learning of the URL representations, while the second one corresponds to the training of a classifier using the learned URL representations. A high-level overview of the pipelines of these two training phases is illustrated at Fig.3.2.

Due to the high cardinality of URLs, we learn the URL representations implicitly by learning and aggregating their tokens representations. In this study, we present and examine four different URL representation models, f_r: $url \rightarrow \boldsymbol{x}_{url}$, where $\boldsymbol{x}_{url} \in \mathbb{R}^r$ and r is the dimensionality of the embedding space. The first one is the simple *one-hot encoding* that treats tokens as atomic units. The URL representation is calculated as an average of the one-hot encoding vectors of its tokens. The main limitation of this model is that the representation is sparse with its size growing with the size of the dataset and it doesn't consider the similarity between URLs. To overcome these

1. The `http(s)://` and `www` parts of each URL are stripped.



(a) URL representation learning



(b) Conversion prediction model learning

Figure 3.2 – URL representation and conversion classifier learning pipeline. The binary labels are not needed for training the URL representation model.

issues, we propose three URL embedding models that learn dense and semantically meaningful URL representation.

After having trained a representation model, f_r, and given a training set $\mathcal{D} = \{(u_n, y_n)\}_{n=1}^N$, we produce a new dataset $\mathcal{D}' = \{(\boldsymbol{X}_n, y_n)\}_{n=1}^N$ where $\boldsymbol{X}_n = [\boldsymbol{x}^n_{url_1}, \ldots, \boldsymbol{x}^n_{url_{T_n}}]$ is a sequence of length T_n with elements $\boldsymbol{x}^n_{url_t} = f_r(url^n_t)$. In a nutshell, $\mathcal{D}'$ contains sequences of URL embedding vectors along with their labels. Then, we apply mapping f_m: $\boldsymbol{X} \to \boldsymbol{z}$ that aggregates the URLs embeddings into an embedding vector $\boldsymbol{z} \in \mathbb{R}^m$, where m can be different from r. It results in a single compact representation $\boldsymbol{z}$ for each sequence of URLs. Next, our goal is to discover a classification model f_c: $\boldsymbol{z} \to \hat{y}$ from a set $\mathcal{F}$ of possible models with the minimum empirical risk

$$\min_{f_c \in \mathcal{F}} \mathbb{E}_{(\boldsymbol{X},y)\sim\mathcal{D}'}[\ell(f_c(f_m(\boldsymbol{X})), y)], \tag{3.2}$$

where ℓ is a non-negative loss function. In fact, classifier f_c is trained on dataset $\mathcal{D}'' = \{(\boldsymbol{z}_n, y_n)\}_{n=1}^N$. In this work, we use logistic regression as classifier f_c. To learn the unknown model parameters, the cross-entropy loss is applied.

Next, we are describing the three different mapping functions f_m adopted in our work: "average", "dense" and "LSTM". The first one returns the average of the URLs embedding vectors presented on a sequence:

$$f_m^{(1)}(\boldsymbol{X}) = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}_{url_t}. \tag{3.3}$$

The second one considers the dependencies among the features of the embedding vector returned by the first mapping function. To be more precise, on top of $f_m^{(1)}(\boldsymbol{X})$ (average of the URLs embedding vectors), a dense layer with *rectified linear units* (ReLU, Nair and Geoffrey E. Hinton, 2010) is applied, i. e. ,

$$f_m^{(2)}(\boldsymbol{X}) = g(\boldsymbol{w}^{(1)\top} f_m^{(1)}(\boldsymbol{X}) + \boldsymbol{b}^{(1)}), \tag{3.4}$$

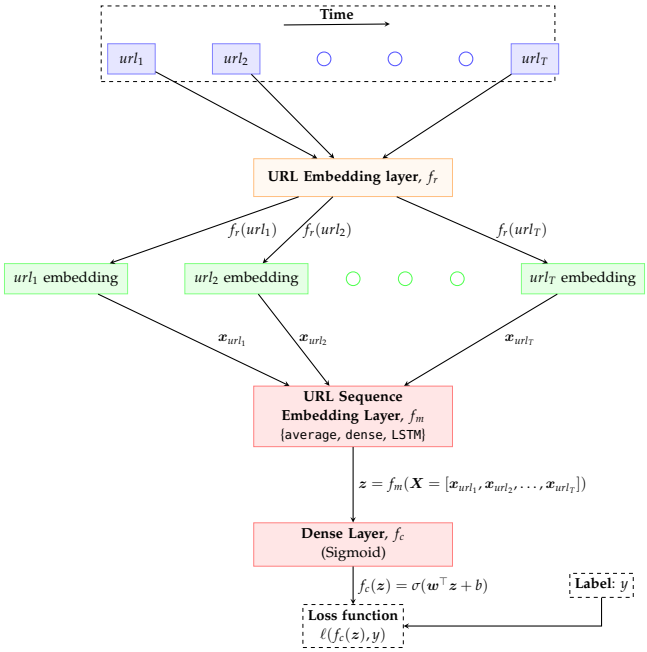


Figure 3.3 – The proposed conversion prediction model architecture. It consists of three parts: i) URL embedding layer (f_r), ii) URL sequence embedding layer (f_m), and iii) Logistic regression classifier (f_c). Only the unknown parameters of the classifier layer and those of "LSTM" and "dense" mappings are trainable.

where g is the ReLU activation function with $g(z) = \max\{0, z\}$. The main limitation of applying one of the two aforementioned mappings is that they do not take into account the sequential order in which the URLs appeared on the sequence. To overcome this limitation, we resort to the well-known *Long Short-Term Memory network* (Hochreiter and Schmidhuber, 1997) that is a special kind of *RNNs* (David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, 1988) and is suitable to process variable-length sequences. To be more precise, the third mapping function $f_m^{(3)}$ is an LSTM network (see Section 2.3.3 for details) which is able to map an input sequence of URL embeddings X to a fixed-sized vector $\boldsymbol{z}$, that can be considered as the representation of the whole sequence, i. e. ,

$$f_m^{(3)}(\boldsymbol{X}) = LSTM(\boldsymbol{X}) \tag{3.5}$$

In all cases, we are feeding the produced vector $\boldsymbol{z}$ to a final dense layer with sigmoid activation function. In the rest contents, we denote as `LR`, `DLR` and `RNN` the prediction conversion models which are using the "average", "dense" and "LSTM" mapping functions, respectively. A graphical illustration of the proposed conversion prediction model architecture is presented at Fig. 3.3.

3.3 URL REPRESENTATION SCHEMES

This section introduces the four URL representation models proposed in our work. These models can be divided in two categories: i) one-hot encoding, and ii) embedding representation. There is no need for learning in the case of one-hot encoding. On the other hand, the embedding representations of URL tokens are learned in advance and then used to form the final URL representation. A representation is also learned for the so-called `<rare>` and `<Pad>` tokens, respectively. A token is considered `<rare>` if it is present less than a predefined number (we set it equal to 20) of times in our data. And the `<Pad>` token is used to pad the URL with few tokens (explained later in this section). We denote the representation vector of token as $\boldsymbol{x}_{token}$.

ONE-HOT ENCODING First, we introduce a variant of the standard one-hot encoding that treats the token as atomic unit instead of the entire URL. The model using one-hot encoding is served as our baseline model. As already mentioned, the token representations are used to get the URL representation. To be more precise, we encode a URL by taking the average of the one-hot encodings of its tokens:

$$f_r(url) = \frac{1}{n\_tokens} \sum_{i=1}^{n\_tokens} \boldsymbol{x}_{token_i}. \tag{3.6}$$

The cardinality of the one-hot encoding is equal to the number of all possible tokens appearing in our data.

EMBEDDING LEARNING Despite its simplicity, the aforementioned one-hot encoding vector is sparse and does not take into account the similarity between URLs, while on the same time its dimension grows linearly with the corpus. To tackle these problems, we propose three representation schemes inspired by the idea of WORD2VEC (Mikolov, Sutskever, et al., 2013) which learn dense and semantically meaningful vectors. More specifically, our representation schemes use the *skip-gram* model that given a target word (URL in our case), we try to predict its context words. More formally, the *skip-gram* model tries to find word representations that can be used for predicting the words in its neighborhood. Therefore, given a sequence of words (URLs) $url_1, \ldots, url_T$, our objective is the maximization of the average log probability

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leqslant j \leqslant c, j \neq 0} \log p(url_{t+j} | url_t), \tag{3.7}$$

where c specifies the neighborhood of target URL. The conditional probability is defined by using the softmax function, as

$$p(url_c | url_t) \triangleq \frac{\exp(\boldsymbol{x}_{url_t}^{\top} \boldsymbol{x}'_{url_c})}{\sum_{url \in V} \exp(\boldsymbol{x}_{url_t}^{\top} \boldsymbol{x}'_{url})}, \tag{3.8}$$

where $\boldsymbol{x}_{url_t}$ and $\boldsymbol{x}'_{url_c}$ are the representations for target and context URLs respectively, and V is the vocabulary of URLs.

Due to the softmax calculation in Eq. 3.8, the direct optimization of Eq. 3.7 is computationally expensive when $|V|$ is large. Thus, we adopt the *negative sampling* approach (Mikolov, Sutskever, et al., 2013). In negative sampling, we treat the word's representation learning as a binary classification task where we try to distinguish the target-context pairs of words presented on the training data from those that are not (see Section 2.4.2.2 for details). Following the suggestions of Mikolov, Sutskever, et al., 2013, for each positive pair we are creating k negative (target-context) pairs.

In contrast to the original WORD2VEC model, the proposed representation learning architectures try to learn the tokens representations, instead of the URL representations directly. A token representation is also learned for the case of a `<Pad>` token. For instance, `https://en.wikipedia.org/` is mapped to [`en.wikipedia.org`, `<Pad>`, `<Pad>`]. Since URLs are padded, the number of tokens of each URL is equal to three. Then, by combining the token representations we form the final URL representation that will be used for the training of the conversion prediction classifier. Actually, the second phase of our model (conversion prediction classifier) can be seen as a way to test the effectiveness of our representation models. The main difference between the three proposed embedding representation models is how the token representations are combined to form the final URL embedding vector:

- `Domain_only` representation uses only the representation of the first token to represent the URL, ignoring the representations of the other two tokens.
- `Token_avg` representation takes the average of the token embedding vectors to represent the URL.
- `Token_concat` representation concatenates the token embedding vectors to represent the URL. In this case, the dimension of the URL embedding vector is three times the dimension of the token embedding vectors.

For instance, let $\{\boldsymbol{x}_{token_i}\}_{i=1}^{3}$ be the token embedding vectors of the three tokens presented on a target URL, represented as $url = [token_1, token_2, token_3]$. Then, the `Domain_only` representation is equal to $\boldsymbol{x}_{token_1}$, the `Token_avg` representation is equal to $\frac{1}{3}\sum_{i=1}^{3}\boldsymbol{x}_{token_i}$, and the `Token_concat` representation is given as $[\boldsymbol{x}_{token_1}^{\top}, \boldsymbol{x}_{token_2}^{\top}, \boldsymbol{x}_{token_3}^{\top}]^{\top}$. A graphical illustration of the proposed embedding learning architecture is provided at Fig. 3.4.

3.4 EXPERIMENTS

This section presents the results of our empirical analysis.

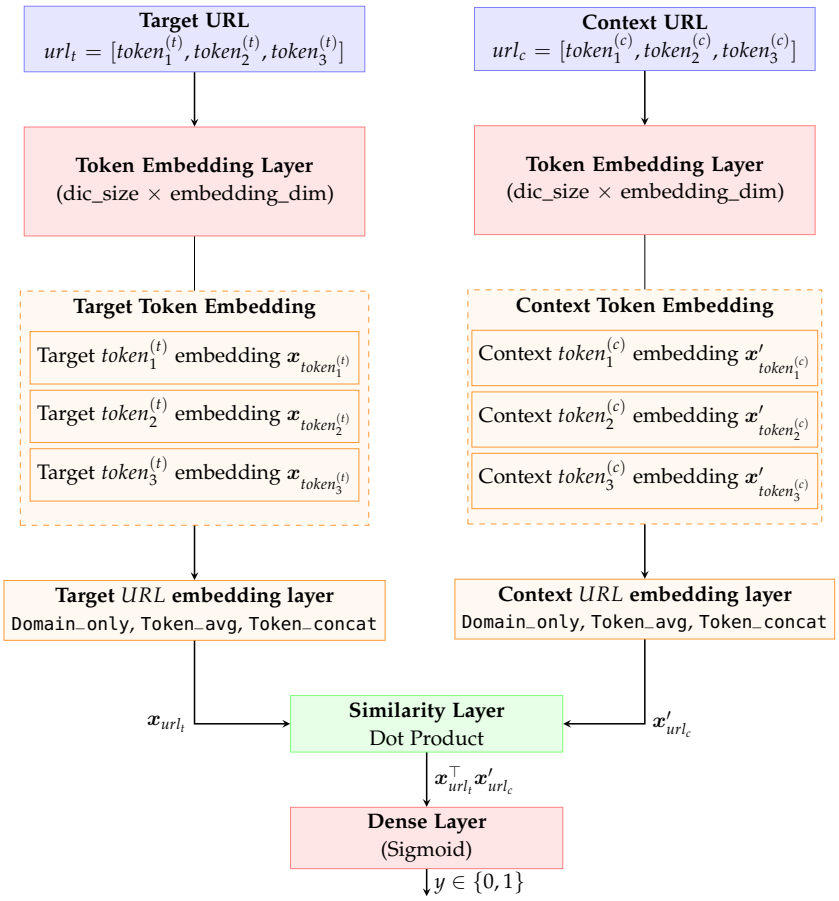


Figure 3.4 – The Skip-gram model architecture used for learning token embeddings. Only the (unknown) parameters of the red blocks are trainable. The dimensionality of the embedding matrices is equal to the number of tokens × the preferable size of the embedding space.

3.4.1 *Datasets*

A real-world RTB dataset was used to train and analyze the performance of the proposed prediction models. We built our dataset by using the auction system logs from campaigns launched in France. It should be also mentioned that our dataset is anonymized, and only visited URLs are considered. In this way, each record of the dataset corresponds to a chronologically ordered sequence of visited URLs along with a binary label (specific to the advertiser) that indicates whether or not a conversion has happened on the advertiser's website on the next day. More precisely, the data composed of sequences of URLs along with their labels of three successive dates, $\mathcal{D}_d$, $\mathcal{D}_{d+1}$, and $\mathcal{D}_{d+2}$, where $\mathcal{D}_d$ is used for learning representations (conversion labels are ignored here), and $\mathcal{D}_{d+1}$ and $\mathcal{D}_{d+2}$ for training and testing the prediction models, respectively. Moreover, the maximum length of a

Table 3.1 – Number of converted vs. non-converted records for each one of the 5 advertisers on the training and testing data.

Advertiser Category	**Training** $(5,452,577)$	**Testing** $(7,164,109)$
Banking	$(3,746-5,448,831)$	$(8,539-7,155,570)$
E-shop	$(1,463-5,451,114)$	$(1,821-7,162,288)$
Newspaper_1	$(1,406-5,451,171)$	$(2,923-7,161,186)$
Newspaper_2	$(1,261-5,451,316)$	$(1,291-7,162,818)$
Telecom	$(1,781-5,450,796)$	$(2,201-7,161,908)$

URL sequence is set equal to 500, where only the most recently visited URLs are kept in each sequence.

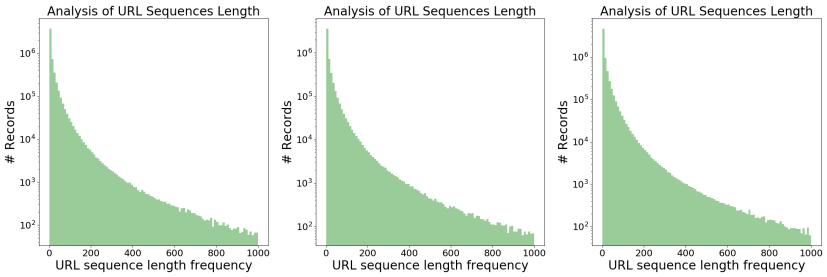
To provide more details about the statistics of the used data, in Figure 3.5, we present the distribution of URL sequence lengths for each dataset. As it can be easily observed, the length of the URL sequences for most of the records is less than 500. In Figure 3.6, we show the token frequency distribution of the first three URL tokens for each dataset. As it was expected, the tokens at the last two spots are more *rare* in the data. Also, the number of unique tokens in the first place (*domain token*) is significantly smaller than the number of unique tokens on the other two spots. In our analysis, we consider a token "rare" if it appears less than 20 times.

In total we examine the performance of the models on five advertisers, belonging to four different categories: banking, e-shop, newspaper, and telecommunications (see Table 3.1).

3.4.2 *Settings*

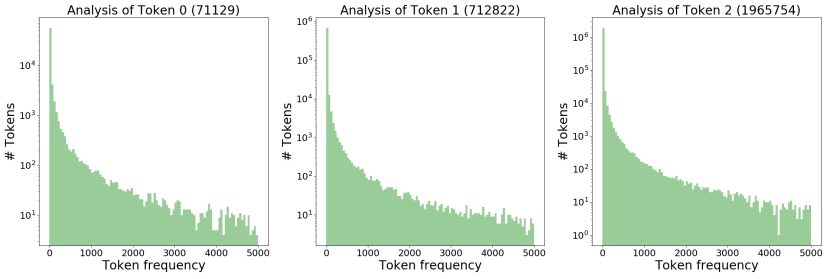
All our predictors assume the existence of a URL representation model in order to vectorize the sequence of URLs for each one of the dataset records. Our baseline, `One_hot/LR`, represents the URL sequences using a one-hot encoding vector of size 193,409, where the first two entries correspond to the `<unknown>` (out of vocabulary) and `<rare>` tokens, respectively. The other models rely on already trained token embedding matrices. Each token is embedded into a 100-dimensional vectors, different for context URL and target URL. The first three rows correspond to the `<unknown>`, `<rare>`, and `<Pad>` tokens, respectively. The rest rows contain the embedding vector of all non-rare tokens observed in the dataset $\mathcal{D}_d$. The number of non-rare tokens is 22,098 for the `Domain_only`, and 187,916 for both the `Token_Average` and `Token_Concatenation`. To train representations, we have considered two different *{pos:neg}* ratios of negative sampling: `{1:1}` (corresponding to $k = 1$) and `{1:4}` (corresponding to $k = 4$).

The number of units of the hidden dense layer (dimensionality of its output space) of `DLR` model is set to 30. Furthermore, the number of hidden units of LSTM is set to 10 on the `RNN` model. A dense layer

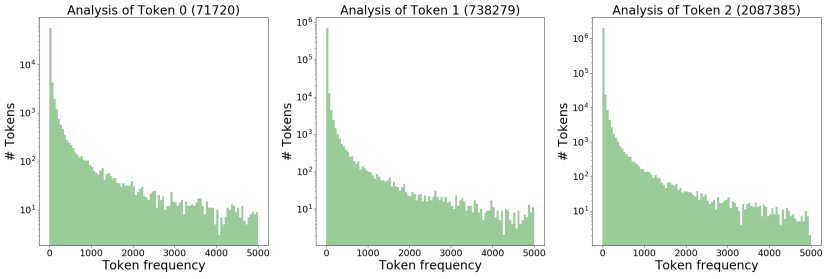


(a) Dataset $\mathcal{D}_d$ (b) Dataset $\mathcal{D}_{d+1}$ (c) Dataset $\mathcal{D}_{d+2}$

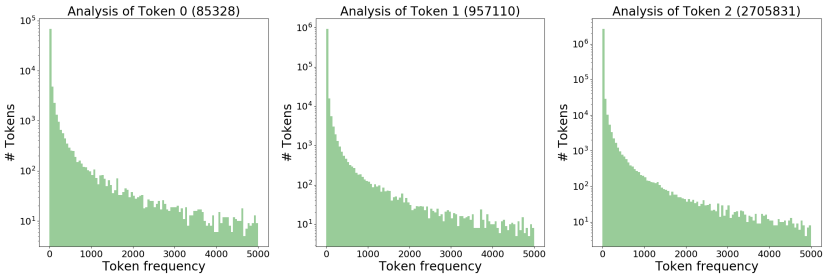
Figure 3.5 – Analysis of the URL sequences lengths for the data, $\mathcal{D}_d$, $\mathcal{D}_{d+1}$, and $\mathcal{D}_{d+2}$.



(a) Dataset $\mathcal{D}_d$



(b) Dataset $\mathcal{D}_{d+1}$



(c) Dataset $\mathcal{D}_{d+2}$

Figure 3.6 – Analysis of URL tokens frequencies. X-axis represents the number of times a token is present in the dataset and y-axis shows the number of tokens. In parentheses we give the number of unique tokens.

with a sigmoid activation function is applied at the end of each one of the three mapping functions f_m ("average", "dense", "LSTM") in order to form a binary classifier. Through our empirical analysis we have observed that the `DLR` and `RNN` prediction models are prone to overfitting. To enhance the generalization capabilities of these two models, we are using dropout that is set to 0.5. To be more precise, a dropout layer is added right after the f_m layer of the `DLR` model, while both the dropout and the recurrent_dropout parameter of LSTM layer are set equal to 0.5 on the `RNN` model.

For the training of all representation and prediction models, the mini-batch stochastic optimization has been applied by using Adam optimizer with the default settings in Tensorflow 2.0 [2] (i.e., lr=0.001, $\text{beta}_1 = 0.9$, $\text{beta}_2 = 0.999$). More precisely, to train the representation models we are doing one full pass over the whole data that is divided to 200 parquet [3] files. The total number of epochs is 200, equal to the number of parquet files. At each epoch we are producing the positive and negative pairs based on the data contained on a single parquet file and are feeding them to the representation model. On the other hand, the size of batches for training prediction models is 64, while the number of epochs and number of steps per epoch is set to 100 in both cases. To tackle the problem of our unbalanced dataset (see Table 3.1), the ratio of positive and negative URL sequences is `{1:1}` in the batches used for the training of the classifiers.

3.4.3 *Results*

In this section, we formally present the results of our empirical analysis.

3.4.3.1 *Visualization*

Firstly, to get an intuition about the ability of the three introduced URL embedding schemes (Sec. 3.3) to group together URLs that belong to the same category (i.e., sports, news, etc.), we will visualize (Fig. 3.7) the embedding vectors of 24 selected domains along with those of the thirty closest URLs of each one of them. To be more precise, we are using the embedding matrices learned by the `Domain_only/1:1` (Fig. 3.7a) and `Domain_only/1:4` (Fig. 3.7b) representation models. Cosine similarity has been used to measure the similarity between the embedding vectors of two URLs. To project the original high-dimensional embedding vectors on a 2-dimensional space, we apply the Barnes-Hut t-SNE [4] algorithm that is able to preserve the pairwise

2. Tensorflow guide: https://www.tensorflow.org/guide/effective_tf2
3. Apache Parquet: https://parquet.apache.org/
4. We have used t-SNE provided by scikit-learn library with its perplexity to be equal to 15 and using cosine similarity metric.

(a) `Domain_only/1:1` URL representation model

(b) `Domain_only/1:4` URL representation model

Figure 3.7 – t-SNE visualization of the thirty closest neighbors of 24 different domains. The colors of the points indicate the closest domain of each URL.

distance between points (i.e. nearby points correspond to similar URLs and that distant points correspond to dissimilar URLs).

To verify that the URLs belong to the same category with that of their closest domain, Tables 3.2 and 3.3 provide the 30-nearest URLs for each one of the 24 domains for the `Domain_only/1:1` and `Domain_only/1:4`, respectively. For instance, in both cases, all URLs that are on the neighborhood of expedia.fr are related to *traveling*. Moreover, all the neighbors of sport.fr (16) are URLs about *sports*.

The visualization of Figs. 3.7a and 3.7b illustrate the ability of the `Domain_only/1:1` and `Domain_only/1:4` representation models to produce semantically meaningful embeddings. Actually, it becomes apparent that we are getting 24 clearly distinguished clusters in both cases. Moreover, the clusters of 'similar' domains are also close on the embedding space. For example, the URLs embeddings of clusters (10) [auto-moto.com] and (19) [renault-laguna.com] are close as they are related to the *automobile* category. The same also holds for the

Table 3.2 – The 30-nearest neighbors of 24 different domains according to our trained `Domain_only/1:1` representation model.

Domain	30-nearest neighbors (URLS)
huffingtonpost.es	sevilla.abc.es; elconfidencial.com; smoda.elpais.com; verne.elpais.com; elmon.cat; elcorreo.com; expansion.com; infolibre.es; vozpopuli.com; blogs.elconfidencial.com; eldiario.es; okdiario.com; cope.es; m.eldiario.es; elespanol.com; elperiodico.com; levante-emv.com; diariodesevilla.es; eleconomista.es; elcaso.elnacional.cat; periodistadigital.com; farodevigo.es; guiadelocio.com; libertaddigital.com; 20minutos.es; motor.elpais.com; mitele.es; diariodenavarra.es; lasexta.com; diariovasco.com
lesechos.fr	business.lesechos.fr; latribune.fr; afrique.latribune.fr; financedemarche.fr; bfmbusiness.bfmtv.com; mieuxvivre-votreargent.fr; photo.capital.fr; capital.fr; challenges.fr; journaldunet.com; lopinion.fr; mtf-b2b-asq.fr; marianne.net; hbrfrance.fr; contrepoints.org; journaldeleconomie.fr; boursier.com; zonebourse.com; investopedia.com; manager-go.com; e-marketing.fr; actufinance.fr; argent.boursier.com; btf-b2b-asq.fr; linkedin.com; mtf-finance-asq.fr; nasdaq.com; btf-finance-asq.fr; lecoindesentrepreneurs.fr; 07-ros-btf-laplacemedia-3.fr
orange.fr	actu.orange.fr; login.orange.fr; finance.orange.fr; tendances.orange.fr; lemoteur.orange.fr; meteo.orange.fr; messagerie.orange.fr; programme-tv.orange.fr; sports.orange.fr; news.orange.fr; boutique.orange.fr; auto.orange.fr; zapping-tv.orange.fr; webmail.orange.fr; people.orange.fr; occasion.auto.orange.fr; mescontacts.orange.fr; videostreaming.orange.fr; pro.orange.fr; mail02.orange.fr; agenda.orange.fr; tv.orange.fr; cineday.orange.fr; mail01.orange.fr; belote-coinchee.net; 118712.fr; ww.orange.fr; cliquojeux.com; freecell.fr; mundijeux.fr
leparisien.fr	btf-actu-asq.fr; lefigaro.fr; marianne.net; scoopnest.com; legorafi.fr; cnews.fr; actu17.fr; bfmtv.com; tendanceouest.com; atlasinfo.fr; opex360.com; jeuneafrique.com; lejdd.fr; video.lefigaro.fr; lalibre.be; rtl.be; liberation.fr; fdesouche.com; causeur.fr; 24matins.fr; amp.lefigaro.fr; courrierinternational.com; bladi.net; tunisienumerique.com; lci.fr; courrier-picard.fr; 7sur7.be; air-defense.net; mixcloud.com; pss-archi.eu
reddit.com	pcgaming.reddit.com; smashbros.reddit.com; old.reddit.com; funny.reddit.com; gaming.reddit.com; europe.reddit.com; france.reddit.com; soccer.reddit.com; askreddit.reddit.com; anime.reddit.com; memes.reddit.com; globaloffensive.reddit.com; starcitizen.reddit.com; freefolk.reddit.com; nintendoswitch.reddit.com; popular.reddit.com; games.reddit.com; aww.reddit.com; leagueoflegends.reddit.com; gameofthrones.reddit.com; politics.reddit.com; redditad.com; dota2.reddit.com; all.reddit.com; totalwar.reddit.com; knowyourmeme.com; pcmasterrace.reddit.com; movies.reddit.com; tinder.reddit.com; nba.reddit.com
expedia.fr	skyscanner.fr; kayak.fr; momondo.fr; fr.hotels.com; fr.lastminute.com; vol.lastminute.com; liligo.fr; quandpartir.com; esky.fr; govoyage.fr; opodo.fr; lonelyplanet.fr; virail.fr; carnetdescapades.com; voyage.lastminute.com; lastminute.com; bravofly.fr; edreams.fr; easyvols.fr; tripadvisor.fr; locations.lastminute.com; ebookers.fr; voyages.bravofly.fr; opodo.com; reservation.lastminute.com; trainhotel.lastminute.com; flights-results.liligo.fr; voyageforum.com; voyagespirates.fr; govoyages.com
tractorfan.fr	forum.farm-connexion.com; discountfarmer.com; heure-ouverture.com; technikboerse.com; meteo-sud-aveyron.over-blog.com; ovniclub.com; materiel-agricole.annuairefrancais.fr; enviedechasser.fr; mash70-75.com; unimog-mania.com; pecheapied.net; renault5.forumactif.com; srxteam.forums-actifs.net; palombe.com; spa-du-dauphine.fr; foreca.fr; meteopassion.com; fiatagri.superforum.fr; meteosurfcanarias.com; super-tenere.org; actu-automobile.com; opel-mokka.forumpro.fr; motoconseils.com; fr.agrister.com; esoxiste.com; v-strom.superforum.fr; view.robothumb.com; sudoku-evolution.com; refugebeauregard.forumactif.com; m.meteorama.fr
welt.de	zeit.de; tagesspiegel.de; sueddeutsche.de; faz.net; sport1.de; badische-zeitung.de; merkur.de; rp-online.de; kicker.de; handelsblatt.com; tz.de; tvmovie.de; abendblatt.de; sportbild.bild.de; nzz.ch; bild.de; seattletimes.com; express.de; morgenpost.de; bz-berlin.de; netzwelt.de; bunte.de; derwesten.de; t-online.de; general-anzeiger-bonn.de; mopo.de; transfermarkt.de; techbook.de; finanzen.net; aargauerzeitung.ch
foreca.fr	meteo81.fr; meteosurfcanarias.com; sudoku-evolution.com; meteopassion.com; mxcircuit.fr; ledicodutour.com; pecheursunisdelille.com; moncompte-espaceclient.com; impactfm.fr; discountfarmer.com; meteo-normandie.fr; monde-du-velo.com; fr.tutiempo.net; meteojura.fr; ovniclub.com; palombe.com; sur.ly; videos-chasse-peche.com; retroplane.net; pointeduraz.info; carriere.annuairefrancais.fr; meteo-sud-aveyron.over-blog.com; horaires-douverture.fr; banquesenfrance.fr; genealogic.review; fr.meteovista.be; baboun57.over-blog.com; meteonews.ch; pecheapied.net; fournaise.info
auto-moto.com	feline.cc; largus.fr; news.autojournal.fr; auto-mag.info; test-auto.auto-moto.com; automobile-magazine.fr; caradisiac.com; neowebcar.com; essais.autojournal.fr; promoneuve.fr; autojournal.fr; latribuneauto.com; motorlegend.com; turbo.fr; actu-moteurs.com; worldscoop.forumpro.fr; palais-de-la-voiture.com; voiture.autojournal.fr; autoplus.fr; moniteurautomobile.be; recherche.autoplus.fr; news.sportauto.fr; abcmoteur.fr; fiches-auto.fr; ww2.autoscout24.fr; constructeur.autojournal.fr; zeperfs.com; essais-autos.com; motoservices.com; sportauto.fr
az-online.de	alittlecraftinyourday.com; theorganisedhousewife.com.au; brittanyherself.com; directoalpaladar.com.mx; mikseri.net; homeplans.com; thesurvivalgardener.com; dmv.org; milliondollarjourney.com; symbols.com; xataka.com.co; kiwilimon.com; mu-43.com; houseofjoyfulnoise.com; leinetal24.de; turniptheoven.com; wetterkanal.kachelmannwetter.com; thinksaveretire.com; lovechicliving.co.uk; wereparents.com; cookrepublic.com; foodinjars.com; lettermenrow.com; raegunramblings.com; thedailytay.com; losreplicantes.com; intmath.com; arthritis-health.com; ourpaleolife.com; juneauempire.com
tempsdecuisson.net	cuisinenligne.com; mamina.fr; scally.typepad.com; audreycuisine.fr; atelierdeschefs.fr; temps-de-cuisson.info; aux-fourneaux.fr; uneplumedanslacuisine.com; cuisine-facile.com; gateaux-et-delices.com; chefnini.com; humcasentbon.over-blog.com; yummix.fr; fruitsdelamer.com; toutlemondeatabl.canalblog.com; lesjoyauxdesherazade.com; gateaux-chocolat.fr; petitsplatsentreamis.com; pechedegourmand.canalblog.com; certiferme.com; yumelise.fr; quelquesgrammesdegourmandise.com; toques2cuisine.com; ricardocuisine.com; companionetmoi.com; recettessimples.fr; docteurbonnebouffe.com; cuisinealafrancaise.com; lighttome.fr; recetteramadan.com
cnn.com	us.cnn.com; grimsbytelegraph.co.uk; stadiumtalk.com; uk.reuters.com; euronews.com; expressandstar.com; thedailymash.co.uk; politico.eu; nbcnews.com; thedailybeast.com; trendscatchers.co.uk; lancashiretelegraph.co.uk; blogs.spectator.co.uk; itpro.co.uk; standard.co.uk; kentonline.co.uk; doityourself.com; deliaonline.com; puzzles.independent.co.uk; breakingnews.ie; oxfordmail.co.uk; slashdot.org; sportinglife.com; abcactionnews.com; huffpost.com; indy100.com; anagrammer.com; drivepedia.com; nigella.com; trumpexcel.com
portail-cloture.ooreka.fr	mur.ooreka.fr; bricolage-facile.net; forum-maconnerie.com; decoration.ooreka.fr; porte.ooreka.fr; abri-de-jardin.ooreka.fr; fr.rec.bricolage.narkive.com; expert-peinture.fr; muramur.ca; amenagementdujardin.net; tondeuse.ooreka.fr; desinsectisation.ooreka.fr; aac-mo.com; fenetre.ooreka.fr; bricoleurpro.com; recuperation-eau-pluie.ooreka.fr; pergola.ooreka.fr; volet.ooreka.fr; papier-peint.ooreka.fr; arrosoirs-secateurs.com; carrelage.ooreka.fr; assainissement.ooreka.fr; pierreetsol.com; poele-cheminee.ooreka.fr; forumbrico.fr; poimobile.fr; parquet.ooreka.fr; forum-plomberie.com; deconome.com; serrure.ooreka.fr
sport.fr	infomercato.fr; topmercato.com; mercatofootanglais.com; parisfans.fr; footlegende.fr; pariskop.fr; le1osport.com; allpaname.fr; le-onze-parisien.fr; vipsg.fr; planetepsg.com; mercatoparis.fr; paristeam.fr; buzzsport.fr; canal-supporters.com; football.fr; culturepsg.com; footparisien.com; foot-sur7.fr; footradio.com; mercatolive.fr; olympique-et-lyonnais.com; planetelille.com; footballclubdemarseille.fr; blaugranas.fr; 90min.com; sportune.fr; footmarseillais.com; livefoot.fr; foot01.com
anti-crise.fr	echantillonsclub.com; cfid.fr; gesti-odr.com; plusdebonsplans.com; promoalert.com; forum.anti-crise.fr; madstef.com; franceechantillonsgratuits.com; cataloguemate.fr; mesechantillonsgratuits.fr; argentdubeurre.com; auchan.fr; maximum-echantillons.com; forum.madstef.com; tous-testeurs.com; jeu-concours.biz; vos-promos.fr; tiendeo.fr; echantillonsgratuits.fr; echantinet.com; pubeco.fr; bons-plans-astuces.com; lp.testonsensemble.com; toutdonner.com; conforama.fr; clubpromos.fr; vente-unique.com; ofertolino.fr; promo-conso.net; fr.testclub.com
auchan.fr	conforama.fr; but.fr; rueducommerce.fr; vente-unique.com; cdiscount.com; touslesprix.com; webmarchand.com; anti-crise.fr; fr.shopping.com; cataloguemate.fr; leguide.com; offrespascher.com; promoalert.com; fr.xmassaver.net; promobutler.be; plusdebonsplans.com; tiendeo.fr; promopascher.com; destockplus.com; meilleurvendeur.com; idealo.fr; pubeco.fr; cdiscountpro.com; argentdubeurre.com; clubpromos.fr; mistergooddeal.com; prixreduits.net; clients.cdiscount.com; horaires.lefigaro.fr; fr.clasf.com
paris-sorbonne.academia.edu	cnrs.academia.edu; ehess.academia.edu; uclouvain.academia.edu; ephe.academia.edu; univ-paris1.academia.edu; univ-paris8.academia.edu; univ-lorraine.academia.edu; mindtools.com; univ-catholyon.academia.edu; ancient.eu; ffmedievale.forumgratuit.org; oxford.academia.edu; unil.academia.edu; iprofesional.com; theartstory.org; univ-amu.academia.edu; marineinsight.com; mapsofindia.com; trend-online.com; coniugazione.it; diggita.it; docsity.com; biografiasyvidas.com; infoplease.com; fr.actualitix.com; docplayer.es; thelocal.de; lectures49.over-blog.com; diariodocentrodomundo.com.br; cinemagia.ro
renault-laguna.com	megane3.fr; gps-carminat.com; megane2.superforum.fr; r25-safrane.net; diagnostic-auto.com; renault-zoe.forumpro.fr; techniconnexion.com; gamblewiz.com; forum-super5.fr; lesamisdudiag.com; forum.autocadre.com; minivanchrysler.com; renault-clio-4.forumpro.fr; bmw-one.com; club.caradisiac.com; lesamisdelaprog.com; forum-bmw.fr; alfaromeo-online.com; marcopolo.superforum.fr; v2-honda.com; alfa147-france.net; question-auto.fr; forum308.com; qashqai-passion.info; automobile-conseil.fr; fr.motocrossmag.be; magmotardes.com; auto-evasion.com; btf-automoto-asq.fr; fr.bmwfans.info
excel-plus.fr	tech-connect.info; jetaide.com; officepourtous.com; lame.buanzo.org; it.ccm.net; lecompagnon.info; patatos.over-blog.com; faclic.com; abracadabrapdf.net; faqword.com; windows.developpez.com; thehackernews.com; jiho.com; cartoucherecharge.fr; questionbureautique.over-blog.com; comment-supprimer.com; cgsecurity.org; silkyroad.developpez.com; technologie.toutcomment.com; java.developpez.com; br.ccm.net; phptester.net; lephpfacile.com; blogosquare.com; monpc-pro.fr; python.developpez.com; astuces.jeanviet.info; poftut.com; tecadmin.net; blog-nouvelles-technologies.fr
jeuxvideo.org	pvpro.com; fallout.fandom.com; pokecommunity.com; xbox-mag.net; make-fortnite-skins.com; garrycity.fr; en.riotpixels.com; brainly.com; gtalogo.com; pngimg.com; 3daim-trainer.com; creativeuncut.com; twitchoverlay.com; en.magicgameworld.com; frondtech.com; discord.me; 11anim.com; kiranico.com; dllme.com; jeugeek.com; myinstants.com; online-voice-recorder.com; mugenarchive.com; honga.net; strawpoll.com; chompy.app; gamepressure.com; cleverbot.com; rp-manga.forum-canada.com; filedropper.com
farmville2free.com	fv2freegifts.org; goldenlifegroup.com; fb1.farm2.zynga.com; juegossocial.com; fv-zprod-tc-0.farmville.com; megazebra-facebook-trails.mega-zebra.com; zy2.farm2.zynga.com; gameskip.com; fv-zprod.farmville.com; actiplay-asn.com; iscool.iscoolapp.com; farmvilledirt.com; secure1.mesmo.tv; megazebra-facebook.mega-zebra.com; prod-web-pool.miniclip.com; banner2.cookappsgames.com; puzzledhearts.com; zynga.com; buggle.cookappsgames.com; apps.facebook.com; deliresetamities.1fr1.net; bubble-coco.cookappsgames.com; apps.fb.miniclip.com; rummikub-apps.com; amomama.fr; gifwi.com; jigsawpuzzlequest.com:3000; pengle.cookappsgames.com; webgl.exoty.com; fr.opossumsauce.com
vogue.fr	vanityfair.fr; vogue.com; fr.metrotime.be; vivreparis.fr; o.nouvelobs.com; apartmenttherapy.com; vice.com; lefaso.net; timeout.fr; unjourdeplusaparis.com; whosdatedwho.com; pariszigzag.fr; wwd.com; gq.com; taddlr.com; vanityfair.com; elle.com; brain-magazine.fr; admagazine.fr; fashiongonerogue.com; people.com; glamourparis.com; leplus.nouvelobs.com; unilad.co.uk; hellomagazine.com; maliactu.net; noisey.vice.com; france-hotel-guide.com; thisisinsider.com; lanouvelletribune.info
tripadvisor.fr	monnuage.fr; cityzeum.com; fr.hotels.com; voyageforum.com; rome2rio.com; virail.fr; carnetdescapades.com; petitfute.com; kayak.fr; voyages.michelin.fr; lonelyplanet.fr; expedia.fr; partir.com; quandpartir.com; salutbyebye.com; mackoo.com; actualitix.com; voyages.ideoz.fr; voyage.linternaute.com; week-end-voyage-lisbonne.com; skyscanner.fr; toocamp.com; plages.tv; routard.com; momondo.fr; gotoportugal.eu; evous.fr; esky.fr; l-itineraire.paris; lepetitmoutard.fr

Table 3.3 – The 30-nearest neighbors of 24 different domains according to our trained `Domain_only/1:4` representation model.

Domain	30-nearest neighbors (URLS)
huffingtonpost.es	cope.es; m.eldiario.es; okdiario.com; verne.elpais.com; blogs.elconfidencial.com; vozpopuli.com; elespanol.com; smoda.elpais.com; libertaddigital.com; cadenaser.com; sevilla.abc.es; elmon.cat; elperiodico.com; levante-emv.com; kiosko.net; elcorreo.com; motor.elpais.com; cronicaglobal.elespanol.com; elplural.com; ara.cat; rac1.cat; eldiario.es; heraldo.es; elperiodico.cat; eleconomista.es; diariodesevilla.es; guiadelocio.com; lasexta.com; periodistadigital.com; mismarcadores.com
lesechos.fr	latribune.fr; afrique.latribune.fr; business.lesechos.fr; bfmbusiness.bfmtv.com; financedemarche.fr; challenges.fr; investopedia.com; actufinance.fr; lopinion.fr; contrepoints.org; rfi.fr; etudiant.lefigaro.fr; hbrfrance.fr; capital.fr; e-marketing.fr; marianne.net; journaldunet.com; 05-habillages-theplacetobid.fr; courrierinternational.com; nouvelobs.com; mtf-b2b-asq.fr; lexpansion.lexpress.fr; zonebourse.com; start.lesechos.fr; lentreprise.lexpress.fr; boursier.com; manager-go.com; investing.com; boursedirect.fr; journaldeleconomie.fr
orange.fr	actu.orange.fr; lemoteur.orange.fr; messagerie.orange.fr; login.orange.fr; finance.orange.fr; sports.orange.fr; meteo.orange.fr; tendances.orange.fr; programme-tv.orange.fr; news.orange.fr; boutique.orange.fr; pro.orange.fr; chaines-tv.orange.fr; ww.orange.fr; agenda.orange.fr; people.orange.fr; zapping-tv.orange.fr; mescontacts.orange.fr; mail01.orange.fr; occasion.auto.orange.fr; video-streaming.orange.fr; musique.orange.fr; tv.orange.fr; mail02.orange.fr; auto.orange.fr; webmail.orange.fr; 118712.fr; cineday.orange.fr; belote-coinchee.net; mahjonggratuit.fr
leparisien.fr	cnews.fr; atlasinfo.fr; lefigaro.fr; lejdd.fr; jforum.fr; marianne.net; video.lefigaro.fr; tendanceouest.com; bladi.net; observalgerie.com; causeur.fr; scoopnest.com; etudiant.lefigaro.fr; actu17.fr; lalibre.be; fdesouche.com; people.bfmtv.com; rmc.bfmtv.com; bfmtv.com; ici.radio-canada.ca; nouvelobs.com; amp.lefigaro.fr; breizh-info.com; fr.euronews.com; observers.france24.com; tunisienumerique.com; courrierinternational.com; lopinion.fr; sfrpresse.sfr.fr; 94.citoyens.com
reddit.com	imgur.com; old.reddit.com; askreddit.reddit.com; pcgamer.com; anime.reddit.com; france.reddit.com; gamefaqs.gamespot.com; totalwar.reddit.com; nintendoswitch.reddit.com; gaming.reddit.com; knowyourmeme.com; redditad.com; pcgaming.reddit.com; funny.reddit.com; europe.reddit.com; leagueoflegends.reddit.com; all.reddit.com; freefolk.reddit.com; pcmasterrace.reddit.com; soccer.reddit.com; globaloffensive.reddit.com; gamesradar.com; dankmemes.reddit.com; gfycat.com; gameofthrones.reddit.com; overwatch.reddit.com; popular.reddit.com; smashbros.reddit.com; competitiveoverwatch.reddit.com; www.reddit.com
expedia.fr	momondo.fr; skyscanner.fr; kayak.fr; fr.lastminute.com; fr.hotels.com; flights-results.liligo.fr; ebookers.fr; esky.fr; opodo.com; secure.lastminute.com; bravofly.fr; opodo.fr; vol.lastminute.com; vols.idealo.fr; locations.lastminute.com; quandpartir.com; opodo.ch; reservation.lastminute.com; quellecompagnie.com; trainhotel.lastminute.com; lonelyplanet.fr; easyvols.fr; voyages.bravofly.fr; edreams.fr; sejour.lastminute.com; rome2rio.com; voyagespirates.fr; jetcost.com; voyage.lastminute.com; virail.fr
tractorfan.fr	discountfarmer.com; forum.farm-connexion.com; angleterre.meteosun.com; songs-tube.net; materieltp.fr; assovttroc.clicforum.fr; opel-mokka.forumpro.fr; spa-du-dauphine.fr; vanvesactualite.blog4ever.com; calcul-frais-de-notaire.fr; sectr.net; cuir-creation.forum-box.com; fc-fief-geste.footeo.com; classements.snt-voile.org; rjm-radio.fr; gps-tomtom.fr; v-strom.superforum.fr; migrateurs.forumgratuit.org; gazoline.forumactif.com; recuperation-metaux.annuairefrancais.fr; voitures-societe.ooreka.fr; fcplouay.footeo.com; equishopping.com; annonce123.com; 36kines.com; bastia.onvasortir.com; renault5.forumactif.com; globaldjmix.com; enviedechasser.fr; squidtv.net
welt.de	zeit.de; sueddeutsche.de; faz.net; tagesspiegel.de; sport1.de; kicker.de; saarbruecker-zeitung.de; tz.de; bild.de; sportbild.bild.de; hartgeld.com; nzz.ch; abendblatt.de; express.de; n-tv.de; bunte.de; handelsblatt.com; merkur.de; bz-berlin.de; aargauerzeitung.ch; badische-zeitung.de; promiflash.de; focus.de; t-online.de; transfermarkt.de; finanzen.net; sport.de; flashscore.de; rp-online.de; stylebook.de
foreca.fr	my-meteo.com; fr.meteovista.be; fr.tutiempo.net; meteopassion.com; de.sat24.com; nosvolieres.com; meteo-sud-aveyron.over-blog.com; xn--mto-bmab.fr; palombe.com; calculerdistance.fr; meteo81.fr; meteosurfcanarias.com; meteo-normandie.fr; meteobelgique.be; planete-ardechoise.com; parisbrestparis2007.actifforum.com; indicatifs.htpweb.fr; etatdespistes.com; easycounter.com; hauteurdeneige.com; testadsl.net; m.meteorama.fr; discountfarmer.com; prevision-meteo.ch; refugeanimalierdupaysdelanderneau.over-blog.com; infosski.com; grottes-france.com; meteolanguedoc.com; impactfm.fr; pont-ile-de-re.com
auto-moto.com	caradisiac.com; largus.fr; news.autojournal.fr; test-auto.auto-moto.com; auto-mag.info; feline.cc; motorlegend.com; essais.autojournal.fr; automobile-magazine.fr; turbo.fr; autoplus.fr; promoneuve.fr; automobile-sportive.com; neowebcar.com; latribuneauto.com; moniteurautomobile.be; palais-de-la-voiture.com; fr.automobiledimension.com; motoservices.com; autotitre.com; fiches-auto.fr; autojournal.fr; blogzineauto.com; voiture.autojournal.fr; essais-autos.com; notice-utilisation-voiture.fr; sportauto.fr; abcmoteur.fr; recherche.autoplus.fr; news.sportauto.fr
az-online.de	abountifulkitchen.com; thesurvivalgardener.com; leinetal24.de; brittanyherself.com; symbols.com; ourpaleolife.com; msl24.de; milliondollarjourney.com; arthritis-health.com; thehollywoodunlocked.com; preschoolmom.com; lovechicliving.co.uk; paleoglutenfree.com; lettermenrow.com; theeasyhomestead.com; raegunramblings.com; evolving-science.com; mu-43.com; juneauempire.com; mikseri.net; e1.ru; thedailytay.com; alittlecraftinyourday.com; comfortablydomestic.com; chicksonright.com; brepurposed.porch.com; kiwilimon.com; grandforksherald.com; catholicstand.com; greatandhra.com
tempsdecuisson.net	cuisine-facile.com; temps-de-cuisson.info; yummix.fr; aux-fourneaux.fr; cuisinenligne.com; audreycuisine.fr; mamina.fr; uneplumedanslacuisine.com; cnz.to; ricardocuisine.com; chefnini.com; cuisinealafrancaise.com; toutlemondeatabl.canalblog.com; marciatack.fr; atelierdeschefs.fr; lesepicesrient.fr; yumelise.fr; lacuisinededoria.over-blog.com; cuisinebasse-temperature.com; recettessimples.fr; perleensucre.com; la-cuisine-des-jours.over-blog.com; lesjoyauxdesherazade.com; fraichementpresse.ca; gustave.com; gateaux-chocolat.fr; toques2cuisine.com; recettesduchef.fr; petitsplatsentreamis.com; amandinecooking.com
cnn.com	us.cnn.com; stadiumtalk.com; thedailybeast.com; itpro.co.uk; uk.reuters.com; euronews.com; theargus.co.uk; theatlantic.com; thedailymash.co.uk; trendscatchers.co.uk; grimsbytelegraph.co.uk; lancashiretelegraph.co.uk; digg.com; spectator.co.uk; politico.eu; blogs.spectator.co.uk; newstatesman.com; huffingtonpost.co.uk; expressandstar.com; puzzles.bestforpuzzles.com; chroniclelive.co.uk; derbytelegraph.co.uk; irishexaminer.com; globalnews.ca; sportinglife.com; slashdot.org; rte.ie; farandwide.com; kentonline.co.uk; thenational.scot
portail-cloture.ooreka.fr	bricolage-facile.net; mur.ooreka.fr; pierreetsol.com; bricolage.jg-laurent.com; abri-de-jardin.ooreka.fr; aac-mo.com; fr.rec.bricolage.narkive.com; decoration.ooreka.fr; piscineinfoservice.com; bricoleurpro.com; amenagementdujardin.net; betonniere.ooreka.fr; assainissement.ooreka.fr; fenetre.ooreka.fr; expert-peinture.fr; terrasse.ooreka.fr; tondeuse.ooreka.fr; debroussailleuse.ooreka.fr; peinture.ooreka.fr; carrelage.ooreka.fr; parquet.ooreka.fr; amenagement-de-jardin.ooreka.fr; toiture.ooreka.fr; poimobile.fr; schema-electrique.net; installation-electrique.ooreka.fr; forum-maconnerie.com; muramur.ca; wc.ooreka.fr; plaque-de-cuisson.ooreka.fr
sport.fr	infomercato.fr; parisfans.fr; topmercato.com; vipsg.fr; footradio.com; mercatofootanglais.com; le10sport.com; buzzsport.fr; footparisien.com; foot-sur7.fr; planetepsg.com; pariskop.fr; paristeam.fr; footlegende.fr; le-onze-parisien.fr; mercatoparis.fr; 90min.com; canal-supporters.com; allpaname.fr; sportune.fr; livefoot.fr; culturepsg.com; jeunesfooteux.com; losclive.com; mercatolive.fr; footballclubdemarseille.fr; parisunited.fr; football-addict.com; olympique-et-lyonnais.com; football.fr
anti-crise.fr	cfid.fr; forum.anti-crise.fr; gesti-odr.com; echantillonsclub.com; plusdebonsplans.com; cataloguemate.fr; promoalert.com; argentdubeurre.com; madstef.com; forum.madstef.com; vos-promos.fr; mesechantillonsgratuits.fr; franceechantillonsgratuits.com; tiendeo.fr; tous-testeurs.com; maximum-echantillons.com; ofertolino.fr; auchan.fr; pubeco.fr; echantinet.com; echantillonsgratuits.fr; promo-conso.net; bons-plans-astuces.com; commerces.com; lp.testonsensemble.com; grattweb.fr; promobutler.be; jeu-concours.biz; mafamillezen.com; hitwest.com
auchan.fr	but.fr; conforama.fr; vente-unique.com; rueducommerce.fr; fr.shopping.com; cdiscount.com; touslesprix.com; promobutler.be; webmarchand.com; mistergooddeal.com; offres-pascher.com; argentdubeurre.com; cataloguemate.fr; leguide.com; promoalert.com; vos-promos.fr; fr.xmassaver.net; cdiscountpro.com; clients.cdiscount.com; promopascher.com; meilleurvendeur.com; clubpromos.fr; tiendeo.fr; plusdebonsplans.com; promo-conso.net; iziva.com; destockplus.com; pubeco.fr; meonho.info; fr.clasf.com
paris-sorbonne.academia.edu	flux-info.fr; elmostrador.cl; makaan.com; univ-montp3.academia.edu; e-lawresources.co.uk; babycenter.com; newocr.com; insight.co.kr; grandes-inventions.com; police-scientifique.com; entrainementfootballpro.fr; muyinteresante.es; ancient.eu; iprofesional.com; slovnik.aktuality.sk; midis101.com; quemas.mamaslatinas.com; adventureinyou.com; wardrawings.be; esky.ro; puzzle-futoshiki.com; univ-paris8.academia.edu; letssingit.com; learn101.org; mindtools.com; medicoresponde.com.br; br.rfi.fr; lazycatkitchen.com; realmenrealstyle.com; eve-adam.over-blog.com
renault-laguna.com	megane3.fr; gps-carminat.com; megane2.superforum.fr; lesamisdudiag.com; car-actu.com; diagnostic-auto.com; r25-safrane.net; lesamisdelaprog.com; forum.autocadre.com; renault-clio-4.forumpro.fr; renault-zoe.forumpro.fr; v2-honda.com; ccfrauto.com; minivanchrysler.com; techniconnexion.com; forum308.com; lemecano.fr; obd-data.com; forum-super5.fr; club.caradisiac.com; toutsurlamoto.com; cliomanuel.org; forum-kia-sportage.com; tlemcen-electronic.com; focusrstteam.com; 306inside.com; cmonofr.net; 207.fr; automobile-conseil.fr; gamblewiz.com
excel-plus.fr	tech-connect.info; thehackernews.com; lecompagnon.info; panoptinet.com; slice42.com; aliasdmc.fr; astuces.jeanviet.info; nalaweb.com; patatos.over-blog.com; jiho.com; abavala.com; ohmymac.fr; br.ccm.net; easy-pc.org; wisibility.com; filedesc.com; semageek.com; windows.developpez.com; jetaide.com; galaxynote.fr; fr.stealthsettings.com; chezcyril.over-blog.com; tuto4you.fr; faclic.com; alvinalexander.com; thegeekstuff.com; hirensbootcd.org; faqword.com; openshot.org; zoomonapps.com
jeuxvideo.org	alsumaria.tv; minecraft-zh.gamepedia.com; infovisual.info; everyonepiano.com; footstream.live; memedroid.com; darkandlight.gamepedia.com; mbti.forumactif.fr; gachagames.net; honga.net; en.magicgameworld.com; garrycity.fr; satelis-passion.forumactif.com; blog-insideout.com; fallout.fandom.com; howtomechatronics.com; cshort.org; fr.trackitonline.ru; openthefile.net; lutain.over-blog.com; alphabetagamer.com; solveyourtech.com; online-voice-recorder.com; png2jpg.com; ffxforever.over-blog.com; arkhamhorrorfr.forumactif.com; en.riotpixels.com; pexiweb.be; petri.com; rasage-traditionnel.com
farmville2free.com	goldenlifegroup.com; fv2freegifts.org; juegossocial.com; fv-zprod-tc-0.farmville.com; fb1.farm2.zynga.com; zy2.farm2.zynga.com; gameskip.com; fv-zprod.farmville.com; megazebra-facebook-trails.mega-zebra.com; farmvilledirt.com; megazebra-facebook.mega-zebra.com; iscool.iscoolapp.com; secure1.mesmo.tv; belote-prod-multi.iscoolapp.com; jigsawpuzzlequest.com:3000; fr.puzzle-loop.com; connect.arkadiumhosted.com; pengle.cookappsgames.com; buggle.cookappsgames.com; banner2.cookappsgames.com; apps.fb.miniclip.com; goobox.fr; prod-web-pool.miniclip.com; actiplay-asn.com; apps.facebook.com; snf-web.popreach.com; bubblecoco.cookappsgames.com; rummikub-apps.com; watersplash.cookappsgames.com; zynga.com
vogue.fr	vanityfair.fr; vogue.com; vivreparis.fr; fr.metrotime.be; brain-magazine.fr; o.nouvelobs.com; parismatch.be; pariszigzag.fr; admagazine.fr; unilad.co.uk; konbini.com; monblogdefille.com; affairesdegars.com; neonmag.fr; vice.com; nordpresse.be; leplus.nouvelobs.com; lataille.fr; people-bokay.com; commeuncamion.com; fr.euronews.com; demotivateur.fr; star24.tv; madmoizelle.com; vl-media.fr; whosdatedwho.com; sciencepost.fr; physiquedereve.fr; ztele.com; twog.fr
tripadvisor.fr	fr.hotels.com; cityzeum.com; voyages.michelin.fr; lonelyplanet.fr; monnuage.fr; voyageforum.com; rome2rio.com; toocamp.com; virail.fr; partir.com; carnetdescapades.com; quellecompagnie.com; mackoo.com; expedia.fr; momondo.fr; salutbyebye.com; orangesmile.com; skyscanner.fr; voyages.ideoz.fr; cars.liligo.fr; quandpartir.com; kelbillet.com; gotoportugal.eu; officiel-des-vacances.com; busradar.fr; week-end-voyage-lisbonne.com; les-escapades.fr; voyage.linternaute.com; bouger-voyager.com; voyagespirates.fr

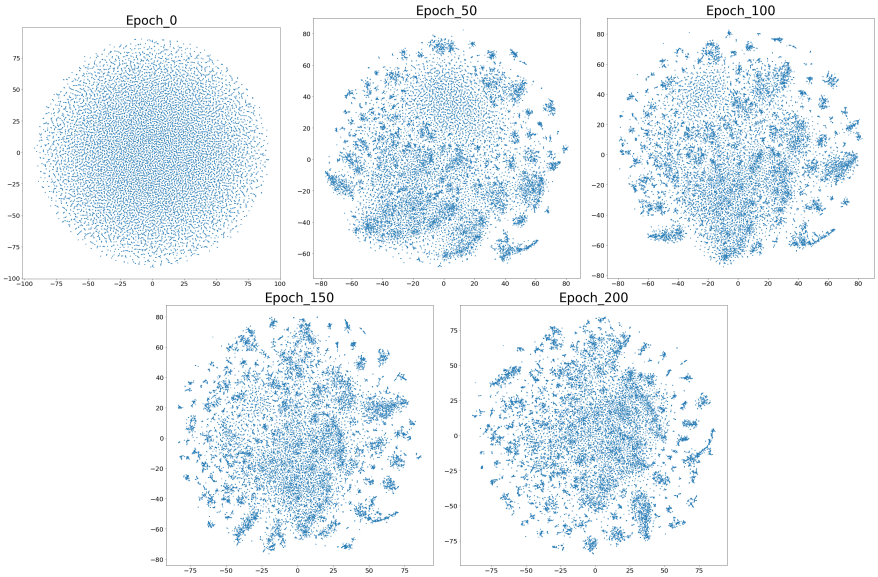


Figure 3.8 – t-SNE visualization of the embedding matrix trained by `Domain_only/1:4` representation model after 0, 50, 100, 150, and 200 epochs, respectively.

URLs embeddings of clusters (2) [lesechos.fr], (4) [leparisien.fr], and (23) [vogue.fr] that belong to the *news* category.

We also observe that clusters close on the embedding space produced by `Domain_only/1:1` are also close on the embedding space of `Domain_only/1:4`. For instance, clusters (6) [expedia.fr] and (24) [tripadvisor.fr] are close in both cases. That is normal, as these two clusters contain URLs about *travelling*. The same also holds for the URLs embeddings of clusters (2) [lesechos.fr], (4) [leparisien.fr], and (23) [vogue.fr] that belong to the *news* category. Moreover, clusters (10) [auto-moto.com] and (19) [renault-laguna.com] are also close as both are related to *automobile*. Another example is that of clusters (16) [anti-crise.fr] and (17) [auchan.fr] are about *promotions (online shopping)*.

Finally, Fig. 3.8 illustrates the t-SNE visualization of the embedding matrix ($22,101 \times 100$) trained by `Domain_only/1:4` representation model after 0, 50, 100, 150, and 200 epochs, respectively.

3.4.3.2 *Numerical results*

Next, we formally present the numerical results of ten prediction models on five different advertisers. In order to distinguish the models, their names consist of three parts separated by a slash ('/') character apart from the `One_hot/LR` model. The first part indicates the type of representation (`Domain_only`, `Token_avg`, `Token_concat`), the second defines the kind of prediction model (`LR`, `DLR`, `RNN`), and the last one shows the *{pos:neg}* ratio (`{1:1}`, `{1:4}`) hyper-parameter used for the training of the representation model.

Table 3.4 – Avg (%) and std of the area under ROC curves (5 independent runs) of the 10 prediction models on 5 advertisers.

Method \ Advertiser	Banking	E-shop	Newspaper_1	Newspaper_2	Telecom
One_hot/LR	65.7 ± 0.093	66.4 ± 0.053	75.5 ± 0.379	73.3 ± 0.400	65.4 ± 0.085
Domain_only/LR/1:1	64.9 ± 0.138	66.7 ± 0.237	76.1 ± 0.109	72.9 ± 0.219	63.4 ± 0.479
Domain_only/LR/1:4	64.6 ± 0.300	66.6 ± 0.217	75.7 ± 0.507	73.2 ± 0.345	63.2 ± 0.168
Domain_only/DLR/1:1	69.2 ± 0.268	70.3 ± 0.383	77.4 ± 0.397	76.7 ± 0.333	67.0 ± 0.370
Domain_only/DLR/1:4	69.0 ± 0.214	69.7 ± 0.234	76.8 ± 0.342	75.7 ± 0.658	66.6 ± 0.303
Domain_only/RNN/1:1	$\mathbf{71.9} \pm 0.258$	72.9 ± 0.149	80.3 ± 0.149	79.7 ± 0.146	71.7 ± 0.252
Domain_only/RNN/1:4	71.4 ± 0.144	72.6 ± 0.422	80.3 ± 0.168	79.4 ± 0.281	71.2 ± 0.250
Token_avg/LR/1:1	63.6 ± 0.188	66.4 ± 0.438	76.1 ± 0.076	72.3 ± 0.466	62.3 ± 0.380
Token_avg/LR/1:4	64.5 ± 0.241	67.2 ± 0.390	76.4 ± 0.152	73.1 ± 0.184	62.9 ± 0.468
Token_avg/DLR/1:1	68.9 ± 0.148	71.5 ± 0.219	78.9 ± 0.175	77.3 ± 0.279	67.3 ± 0.263
Token_avg/DLR/1:4	69.4 ± 0.294	72.1 ± 0.263	79.2 ± 0.242	77.7 ± 0.274	67.9 ± 0.348
Token_avg/RNN/1:1	71.7 ± 0.116	72.9 ± 0.184	80.8 ± 0.121	79.5 ± 0.119	71.4 ± 0.209
Token_avg/RNN/1:4	$\mathbf{71.9} \pm 0.082$	$\mathbf{73.1} \pm 0.246$	$\mathbf{81.2 \pm 0.153}$	$\mathbf{80.2 \pm 0.322}$	$\mathbf{71.8 \pm 0.153}$
Token_concat/LR/1:1	64.5 ± 0.142	67.3 ± 0.149	76.6 ± 0.174	73.3 ± 0.216	63.3 ± 0.672
Token_concat/LR/1:4	64.8 ± 0.241	67.2 ± 0.060	76.7 ± 0.179	73.4 ± 0.273	63.6 ± 0.425
Token_concat/DLR/1:1	69.0 ± 0.187	71.2 ± 0.274	78.7 ± 0.304	76.7 ± 0.232	67.0 ± 0.469
Token_concat/DLR/1:4	69.1 ± 0.222	70.8 ± 0.400	78.2 ± 0.285	76.7 ± 0.255	66.9 ± 0.310
Token_concat/RNN/1:1	71.5 ± 0.214	72.5 ± 0.264	80.9 ± 0.286	79.4 ± 0.318	71.2 ± 0.180
Token_concat/RNN/1:4	71.5 ± 0.224	72.5 ± 0.460	80.5 ± 0.192	79.1 ± 0.130	70.5 ± 0.278

To evaluate and compare the effectiveness of the models we are using the *area under ROC curve* (*AUC*) metric. More specifically, we consider the average (%) AUC across five independent runs (see Table 3.4), where each run corresponds to a specific seed used for the models initialization. Moreover, Figures 3.9 and 3.10 illustrate the average ROC curves of the ten prediction models for each advertiser, where `{1:1}` and `{1:4}` *{pos:neg}* ratios are used as hyper-parameters for training representation models, respectively. The right lower corner of each figure presents the overall performance (AUC scores) of each model on 25 independent runs (5 advertisers $\times$ 5 runs for each advertiser).

Based on the results presented in Table 3.4, it can be easily verified that the prediction model's performance is not so sensitive to the change of the *{pos:neg}* ratio used in negative sampling, with the `{1:4}` to give slightly better results. Actually, the `Token_avg/RNN/1:4` model is more effective in predicting user's conversions compared to the rest models. Precisely, the `Token_avg/RNN/1:4` model has the highest average AUC in all advertisers. All models achieve their highest performance on the newspaper advertisers. It is also worth noticing that the performances of `Domain_only/LR`, `Token_avg/LR`, and `Token_concat/LR` are highly competitive, compared to our baseline, `One_hot/LR`, for both *{pos:neg}* ratio. More specifically, `Token_concat/LR` clearly outperforms `One_hot/LR` in 2 out of 5 advertisers (E-shop, Newspaper_1), and has slightly better or equal performance in one of them (Newspaper_2). That remark validates our claims that the proposed representation models produce meaningful embeddings, by

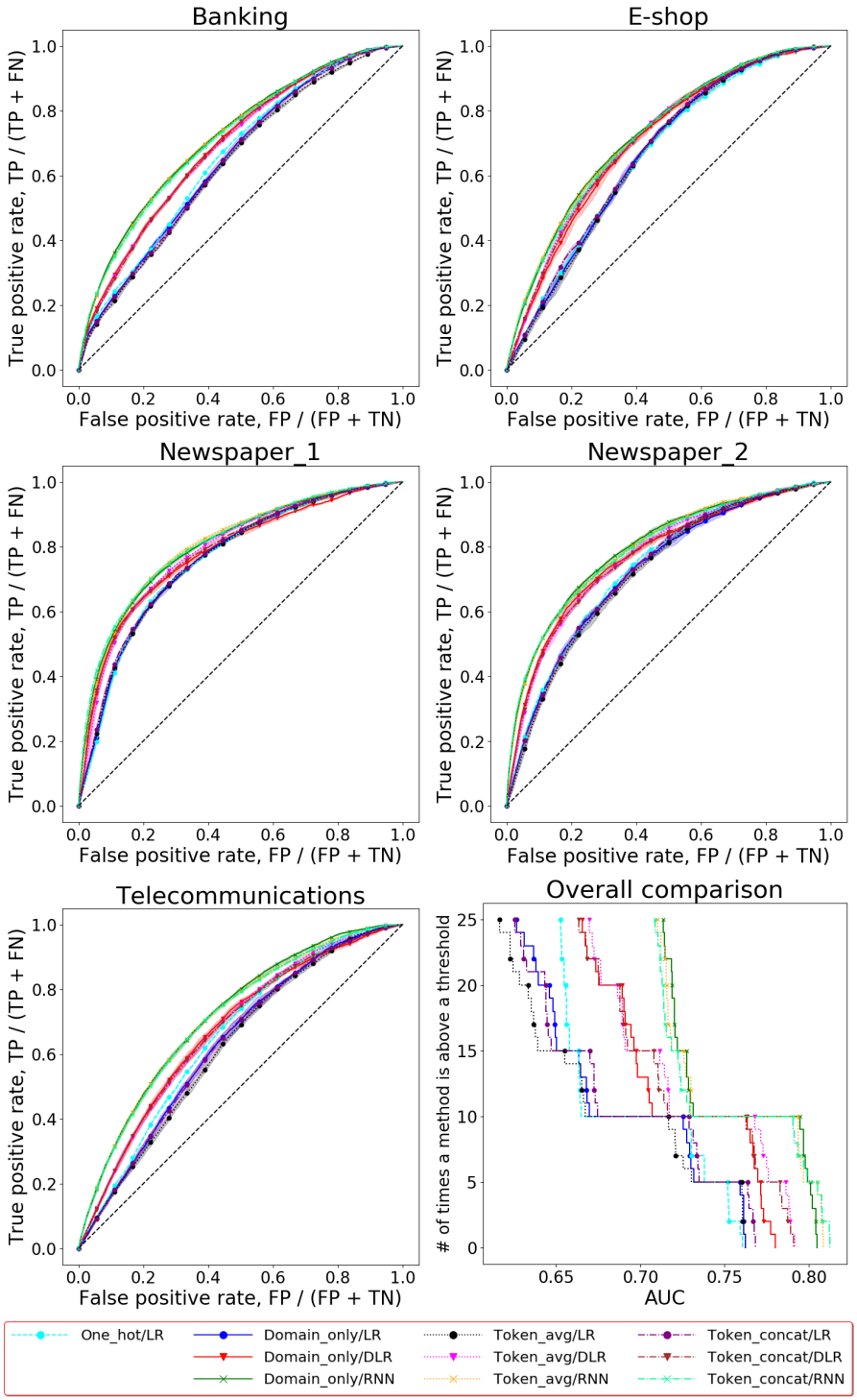


Figure 3.9 – Average ROC curves of the ten conversion prediction ({1:1} *pos-neg* ratio) models on the five advertisers. Shaded regions represent the standard deviations over 5 independent runs. The bottom right plot presents the AUC for each one of the 25 independent runs (5 advertisers $\times$ 5 independent runs for each advertiser) of each model. The $\bullet$, $\blacktriangledown$ and $\times$ marks indicate the `LR`, `DLR` and `RNN` classification models, respectively.

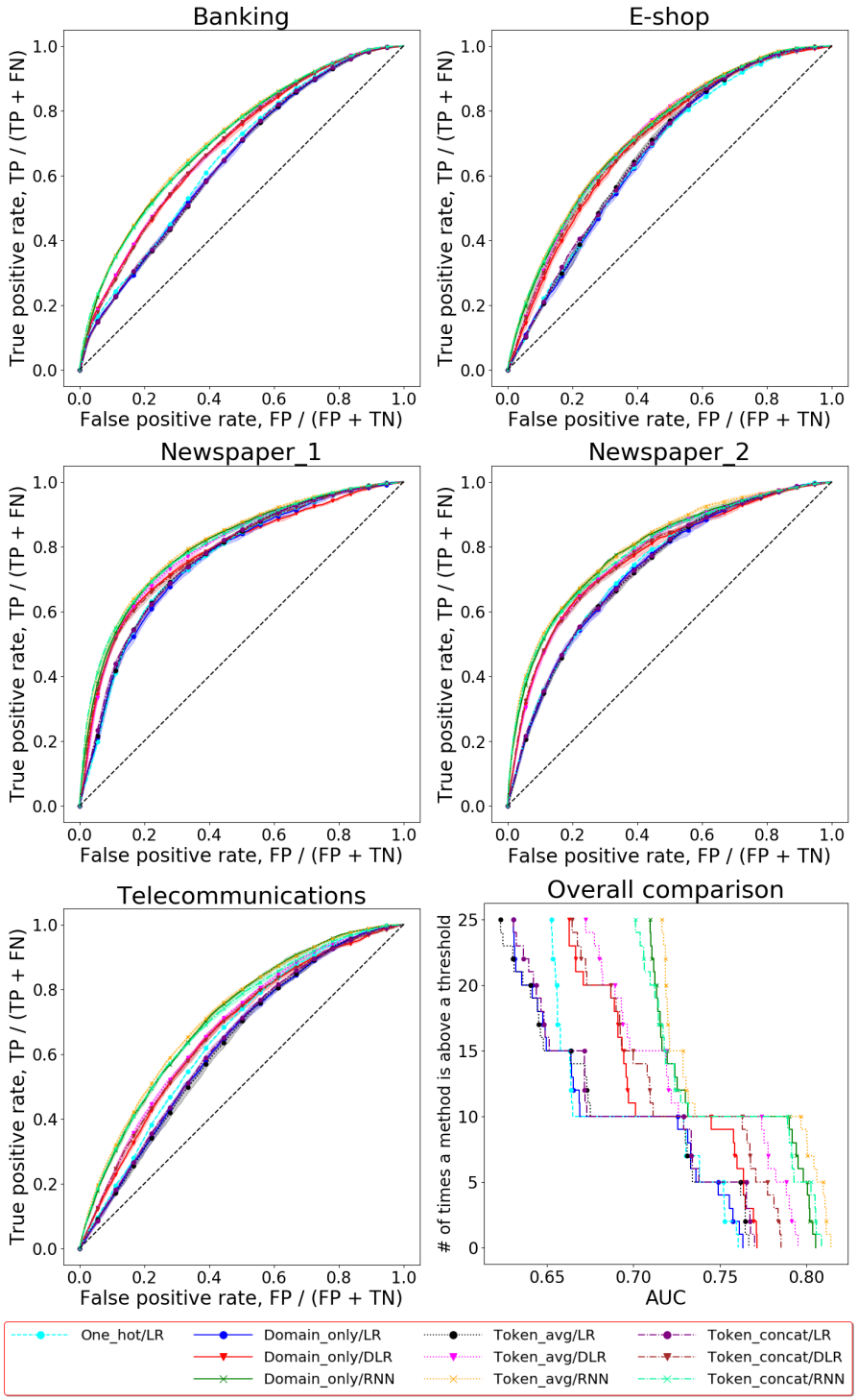


Figure 3.10 – Average ROC curves of the ten conversion prediction ({1:4} *pos-neg* ratio) models on the five advertisers. Shaded regions represent the standard deviations over 5 independent runs. The bottom right plot presents the AUC for each one of the 25 independent runs (5 advertisers $\times$ 5 independent runs for each advertiser) of each model. The $\bullet$, $\blacktriangledown$ and $\times$ marks indicate the `LR`, `DLR` and `RNN` classification models, respectively.

distinguishing URLs of the same category and placing them close to the embedding space. Taking a closer look at the standard deviations, we can see that the performance of `One_hot/LR` is quite stable in the three advertisers. This was expected as the `One_hot` representation is identical over all runs, and therefore the only variation of the performance of `One_hot/LR` comes only from the training of the `LR` classifier. The performance of `LR` model is almost the same for each representation model. The same also holds for `DLR` and `RNN` model where its performance is more or less the same in the cases of `Domain_only` and `Token_concat`, and slightly better in the case of `Token_avg`. On the other hand, `DLR` performs significantly better when it is combined with `Token_avg` and `Token_concat`, with `Token_avg` to be more preferable (around 1% gain in the case of `{1:4}`).

Let us now compare the impact of the type of classifier on the performance of the prediction model.

The overall comparisons presented at Figs. 3.9 and 3.10 demonstrate that both `DLR` and `RNN` performs significantly better compared to `LR`, with the `RNN` to be the best one. More precisely, the AUC of `RNN` is around ∼7% and ∼3% higher compared to those of `LR` and `DLR`, respectively. This means that the consideration of the sequential order in which the URLs appeared on the sequence is of high importance. On the other hand, choosing `DLR` over `LR` improves around ∼4% the performance of the prediction models independent to the representation model.

3.4.3.3 *Summary*

To sum up, the main conclusions of our empirical analysis are:

- All three proposed URL embedding models are able to learn high-quality vector representations that capture precisely the URL relationships. Even the `Domain_only/LR`, `Token_avg/LR`, and `Token_concat/LR` models are able to predict with high accuracy the probability that a user converts, and their performance is quite close or better compared to that of our baseline `One_hot/LR`.
- It can be easily verified by the reported results that the prediction model performance is not so sensitive on the *{pos:neg}* ratio used for training representation models.
- `{1:1}` performs better on the `Domain_only` representation compared to `{1:4}` that performs better on `Token_avg`. On the other hand, the performance of both *{pos:neg}* ratios are almost equivalent for the `Token_concat` model.
- Among the three representation models, `Token_avg` seems to be more adequate to capture the relationships between URLs, with the `Token_concat` to be the second best. Moreover, the

performance of `Domain_only` representation is quite close to that of `Token_concat`.

- The consideration of the chronological order of the visited URLs (`RNN`) and the learning of dependencies among the embedding features (`DLR`) are also of high importance as both improve significantly the performance of the conversion prediction model. To be more precise, it is clear (see bottom right plot of Fig. 3.9 and Fig. 3.10) that the `RNN` model surpass the performance of the rest two classifiers, while `DLR` performs better compared to `LR`.
- The best prediction conversion models are `Domain_only/RNN` and `Token_avg/RNN` for `{1:1}` and `{1:4}` *{pos:neg}* ratios, respectively (see bottom right plot of Fig. 3.9 and Fig. 3.10).

3.5 CONCLUSIONS AND FUTURE DIRECTIONS

In this chapter, we considered the problem of user conversion prediction in display advertising. Ten conversion prediction models were proposed to predict users response based on their browsing history. To represent the sequence of visited URLs, four different URL representations were examined. The effectiveness of the proposed models has been experimentally demonstrated using real-world data for five different advertisers. The impact of the sequential dependency between user's visited URLs on the performance of the predictors has also been examined. The main conclusions of our empirical analysis were that all three proposed URL embedding models produce a meaningful URL representation, and considering the sequential order of the visited URLs by using `RNN` significantly improves the model's performance. In the future, we intend to take the time information into account in our model to fully model the user's browsing activities.

4

AUDIENCE EXTENSION

As digital advertising becomes more and more popular, the necessity of targeting advertising is more present than ever. Digital advertising gives the opportunity to the advertisers to deliver promotional content by executing campaigns through various channels, i.e. display advertising, social media advertising, search-based advertising and contextual advertising. Regardless of the advertising channel, the main objective for the advertisers remains the maximization of the return of their investment. To achieve their objective, advertisers should deliver advertisements about their products or services to the right audience. Some of the most common metrics of effectiveness of advertising campaigns are the *cost-per-click* (CPC) ones, and *cost-per-lead* or *cost-per-action* (CPA) ones, where the end goal is the maximization of the number of clicks, or the number of visits on the advertiser's website.

Audience expansion, also named *look-alike targeting*, is a widely used technique in online digital advertising that has been deployed with success in different Ads serving platforms, i.e., LinkedIn (H. Liu et al., 2016), Yahoo (Qiang Ma et al., 2016), Pinterest (deWet and Ou, 2019), WeChat (Y. Liu et al., 2019), Ant Financial (Z. Liu et al., 2020; Zhuang et al., 2020). Audience expansion systems aim to discover groups of users who look like a set of *seed* users provided by the advertiser in advance. In this way, advertisers promote their products by displaying ads to audiences with similar behavior (e.g., visit, click, purchase, etc.) to the seed-set. The main advantage of an audience expansion system resides in its simplicity in improving the performance of ads campaigns. Instead of specifying some 'targeting criteria' based on their own experience, in an audience expansion system the advertisers need only to provide a list of seed users along with the desired size of the targeted audience.

In this work, we focus on online display advertising (J. Wang, W. Zhang, and S. Yuan, 2017), where advertisers embed ads on a publisher's web page in order to promote their products or services. In this context, we introduce and examine a number of different audience expansion schemes that are based on the users' browsing history. Specifically, given only the browsing history of a set of seed users, the proposed audience expansion schemes are able to identify groups of users with similar browsing interests. As seed users we consider the users that have been converted after their exposure to an advertisement.

The proposed audience expansion schemes are mainly based on different self-supervised (unsupervised) representation models that consider the sequential order of visited URLs in order to represent users. Inspired by natural language processing (NLP), we treat each user as a document where words correspond to visited URLs. Learning high-quality representations of sentences or documents is a long-standing problem in NLP. In order to learn a semantically meaningful user representation, we follow the idea of document embeddings (Mikolov, K. Chen, et al., 2013). In practice, each user is mapped to a vector in an embedding space, where users with similar 'browsing' interests are close in the embedding space. Having computed high quality users' representations, we can compute the affinity scores between candidate and seed users by executing a computationally efficient similarity function, i.e. cosine similarity, euclidean distance. Then, in order to expand our audience, we rank the candidate users based on their maximum affinity scores to any user in the seed set and we select the top-ranked users as the target audience. This is known as *similarity-based* audience expansion strategy (see Section 4.1).

More precisely, the proposed audience expansion schemes can be categorised into four different categories:

i) In the first group of audience expansion schemes (Sec. 4.2.1), the users are simply represented as an unweighted or weighted set of URLs. Then, using the standard or weighted Jaccard similarity we compute the affinity scores between candidate and seed users. In the case of weighted Jaccard similarity we apply a novel weight function inspired by the *inverse document frequency* (Jones, 1973).

ii) Every audience expansion method included in this category (Sec. 4.2.2) represents users based on pre-trained URL embeddings able to represent URLs in a semantic meaningful way. Specifically, we introduce the URL2VEC representation model that is built upon the well-known WORD2VEC model (Mikolov, K. Chen, et al., 2013; Mikolov, Sutskever, et al., 2013). Then, we can consider either the (weighted) average URL embeddings to represent users or Gaussian user representation (Nikolentzos et al., 2017) that models each user as a multivariate Gaussian distribution. A modified version of the famous Word Mover's Distance (WMD, Kusner et al., 2015) is also introduced that computes the distance between users.

iii) The third group of audience expansion schemes (Sec. 4.2.3) adopts the USER2VEC (known as DOC2VEC in the field of NLP) model introduced by Q. Le and Mikolov, 2014 to represent users. In fact, we have considered both DBOW and DM versions of USER2VEC model.

iv) Finally, we introduce a number of audience expansion schemes (Sec. 4.2.4) that adopt Document Vector through Corruption model (M. Chen, 2017) in order to represent users. We call this model as USER2VECC, and it allows us to get users representations by just

computing the (weighted) average of the embeddings of all visited URLs.

Except for the audience expansion schemes that belong to the first group, the rest use different representation models to generate high-quality user representations. All these representation models are trained in a self-supervision way and they are independent to the advertisers. That means that we can train these representation models offline and apply them to expand audiences for different advertisers by simply computing the affinity scores between candidate and seed users (different for each advertiser). Specifically, we have considered two simple and computationally efficient similarity functions, the euclidean distance and the cosine similarity. Extensive offline experiments are conducted to verify the effectiveness and efficiency of the proposed audience expansion schemes. For this purpose, we use a real dataset that is based on real logged events collected from an advertising platform.

4.1 RELATED WORK

Audience expansion approaches can be classified generally into two main categories:

SIMILARITY-BASED AUDIENCE EXPANSION To discover look-alike users, similarity based methods compare directly the similarity of all possible pairs between seed and candidate users using a pairwise similarity function, i.e. such as cosine similarity, euclidean distance, or Jaccard index. Having ranked the users based on their maximum affinity scores to any user in the seed set, we are selecting the top-ranked users as the target audience. Locality-sensitive hashing (LSH, Slaney and Casey, 2008) technique is widely used for reducing the computational cost of pairwise similarity and allows the audience expansion systems to be applied to millions of users (H. Liu et al., 2016; Q. Ma et al., 2016; Qiang Ma et al., 2016).

The learning of high quality user representations plays a key role in the performance of these kinds of methods. The Youtube DNN model (Covington, Adams, and Sargin, 2016) is adopted to learn user representations in Y. Liu et al., 2019 where an attention merge layer replaces the concatenation layer to handle heterogeneous and multi-fields features. deWet and Ou, 2019 propose a neural model that embeds users and Pin topics on the same low dimensional embedding space to capture relationships between users. In Doan, Yadav, and Reddy, 2019, authors propose an adversarial factorization auto-encoder that learns a binary user representation able to encode complex feature interactions.

REGRESSION-BASED AUDIENCE EXPANSION All the methods that belong to this category consider the audience expansion task as a standard binary classification problem. Actually, we treat *seed* users as positive samples, while negative samples are sampled from the non-seed users (i.e., non-converted users who have seen advertiser's ads). Yan Qu et al., 2014 use logistic regression to predict the probability of an unknown user to belong to the seed set. Then, Doan, Yadav, and Reddy, 2019 examines the performance of different powerful classifiers, such as 1-class SVM, Factorization Machine, Gradient Boosting Tree. Qiu et al., 2020 propose different prediction models for estimating the probability of a user converting, given their history of visited URLs. In an abstract point of view, it can be considered as the model-based version of our *similarity-based* audience expansion scheme where a different prediction model is trained for each advertiser. It is worth mentioning that we use the same representation model for learning URL embeddings with the one used in Qiu et al., 2020.

The careful selection of seed (positive) and non-seed (negative) samples is important for the performance of the regression-based audience expansion models. Jiang et al., 2019 examine multiple sampling techniques of selecting negative samples from 'unlabeled' data (non-seed users). Hubble (Zhuang et al., 2020) is a two-stage audience expansion system that uses the well-designed knowledge distillation mechanism, to eliminate the *coverage bias* (the gap between the seeds and the actual audiences) introduced by the provided seed set. A Graph Neural Network is also used to learn user representations, that is based on a user-campaign bipartite graph. A reweighting mechanism is used in Z. Liu et al., 2020 that adjusts the weight of each positive sample to detect noise users within seeds.

4.2 THE PROPOSED AUDIENCE EXPANSION METHODS

This section presents the main components of the proposed look-alike model. The main objective of our audience expansion method is to help advertisers to increase the reach of their marketing campaigns. The audience expansion problem can be formally defined as follows:

Definition 4.2.1 (Audience expansion)**.** *Audience expansion task seeks to discover a subset of users from a set of candidate users $\mathcal{C}$, $\mathcal{C}^* \subset \mathcal{C}$, which are similar to a set of seed users $\mathcal{S}$. In most cases, the expansion framework is conversion-oriented.*

Having computed the affinity matrix $\boldsymbol{A} \in \mathbb{R}_{[0,1]}^{|\mathcal{C}|\times|\mathcal{S}|}$, where $\boldsymbol{A}_{uu'} \in \mathbb{R}_{[0,1]}$ is the entry of $\boldsymbol{A}$ that indicates the affinity score between $u \in \mathcal{C}$ and $u' \in \mathcal{S}$ users, we can discover the closest seed user for each one of the candidates users. Then, we rank the candidate users based on

their similarity score to the closest seed user, and we keep the top, let's say K, users. It can be formally defined as:

$$\mathcal{C}^* = \{u \in \mathcal{C} : |\{u' \in \mathcal{C} : f(u) < f(u')\}| < K\}, \tag{4.1}$$

where $f(u) = \max_{u' \in \mathcal{S}} \boldsymbol{A}_{uu'}$.

In our scenario, each user is represented as a sequence of URLs visited by them in a single day. Actually, user u is considered as a chronologically sorted sequence of URLs:

$$u = [url_1, \ldots, url_{\tau_u}], \tag{4.2}$$

where $\tau_u \geqslant 1$ represents the length of the sequence and varies for each user. Each URL is split with a '/' (slash) character, and is considered itself as a sequence of tokens. In our case, we consider only the first token that corresponds to the domain name, ignoring the other tokens.

Let's also assume that users select to visit URLs from a finite size vocabulary $\mathcal{V}$ of URLs ($|\mathcal{V}| = v$). We denote as $\boldsymbol{b} \in \mathbb{Z}_+^v$ and $\boldsymbol{d} \in \mathbb{R}_+^v$ the bag-of-words (BoW) and normalized bag-of-words (nBoW) user representations, respectively. Specifically, if the i^{th} URL in the vocabulary (url^i) is present c_i times in the user's browsing history, its entry on the BoW vector is $b_i = c_i$ and $d_i = \frac{c_i}{\sum_{j=1}^{n} c_j}$ on the nBoW vector. Table 4.1 lists all the symbols used in this chapter.

Next, we introduce different techniques to compute the affinity score between users. Actually, the proposed audience expansion schemes can be grouped into four main categories.

4.2.1 *Audience expansion based on set similarity metrics*

The most trivial way to represent users is as unweighted or weighted sets of URLs. (Weighted) Jaccard similarity metric can be used to compute the similarity between two users. We denote as $\{u\}$ the set of (unique) URLs visited by u.

Jaccard Similarity Jaccard, 1901 is a simple measure of similarity between two sets. The Jaccard similarity between two users u and u' is computed as the number of shared URLs over the total number of the unique URLs visited by both users:

$$J(u, u') = \frac{|\{u\} \cap \{u'\}|}{|\{u\} \cup \{u'\}|}. \tag{4.3}$$

Weighted Jaccard Similarity (S. Ioffe, 2010), also known as Ruzicka similarity, is a natural generalization of Jaccard similarity:

$$WJ(u, u') = \frac{\sum_{url \in \{u\} \cup \{u'\}} \min(w_u(url), w_{u'}(url))}{\sum_{url \in \{u\} \cup \{u'\}} \max(w_u(url), w_{u'}(url))}, \tag{4.4}$$

Table 4.1 – Notations used in this chapter.

Notation	Description
$\mathcal{S}$	The set of seed users
$\mathcal{C}$	The set of candidate users
$\boldsymbol{A} \in \mathbb{R}_{[0,1]}^{\lvert\mathcal{C}\rvert \times \lvert\mathcal{S}\rvert}$	Affinity matrix that keeps the scores between $\mathcal{C}$ and $\mathcal{S}$
u	User's browsing history $[url_1, \ldots, url_{\tau_u}]$
$\{u\}$	The set of unique URLs visited by u
$\tau_u \geqslant 1$	The total number of URLs visited by u
$\mathcal{V}$	The vocabulary of all URLs appeared in data ($\lvert\mathcal{V}\rvert = v$)
$\boldsymbol{b} \in \mathbb{Z}_+^v$	Bag-of-words (BoW) user representation vector
$\boldsymbol{b}_c \in \mathbb{Z}_+^v$	BoW representation vector of the local context
$\boldsymbol{d} \in \mathbb{R}_+^v$	Normalized BoW (nBoW) user representation vector
$\boldsymbol{X} \in \mathbb{R}^{h \times v}$	Projection matrix that projects URLs to a hidden space of size h
$\boldsymbol{V} \in \mathbb{R}^{h \times v}$	Projection matrix from hidden to original space
$\boldsymbol{Z} \in \mathbb{R}^{h \times n}$	Projection matrix that projects users to a hidden space of size h, used by User2Vec model
$\boldsymbol{T} \in \mathbb{R}^{v \times v}$	Auxiliary 'transport' matrix used on WMD, $\boldsymbol{T}_{ij} \geqslant 0$

where $w_u : url \to \mathbb{R}_+$ specifies the importance of url for user u. The weight function proposed in our work is inspired by the *inverse document frequency* (*idf*, Jones, 1973) and is defined as:

$$w_u(url) = \begin{cases} 0, & \text{if } url \notin \{u\} \\ \log_e\left(\frac{\tau_u}{b_{url}} + 1\right), & \text{otherwise.} \end{cases} \tag{4.5}$$

It can be considered as a measure of how much information the url provides. We intentionally penalize (i. e. , give lower weight to) high-frequency URLs appeared in the user's browsing history, as it's more likely due to the automatic refresh of the corresponding website (see Section 1.5, Paragraph DATA NOISE, Block **bid request noise** for more details). We also add 1 on the ratio inside the logarithm in order to treat the special case where the user has visited only one website

through the day. In this case, we get $w_u(url) = \log_e(2)$, which is the lower bound of the weight function. Note that the weighted Jaccard similarity (Eq. 4.4) is equivalent to the standard Jaccard similarity (Eq. 4.3) if we use binary BoW user representation without any term weighting: $w_u(url) = 1$ if $url \in \{u\}$, and $w_u(url) = 0$ otherwise.

4.2.2 *Audience expansion based on URL embeddings*

Despite its simplicity and interpretability, (weighted) Jaccard similarity is not able to capture the semantic similarity between the interests of two users. Let's consider a simple example with two users $u = [\texttt{leparisien.fr}]$ and $u' = [\texttt{france24.com}]$ each visiting a single URL. The Jaccard score between these two users is equal to 0, implying the interests of these two users are completely different. This is far from true as the context of these two URLs is more or less the same (belong to *news* category). A more efficient approach to measure similarity between users is by transforming users into real-valued embedding vectors that embodies their interests. To be more precise, our objective is to build a user representation space where users looking for the same context are close on the embedding space.

In this section, we present different user representation models $f_m : u \to \boldsymbol{z}$, where $\boldsymbol{z} \in \mathbb{R}^h$ and h is the dimension of the embedding space. All these representation schemes are based on URL embeddings that are high-dimensional vectors able to represent URLs in a semantic meaningful way. In this context, we consider users as documents where URLs play the role of words. Similar to Qiu et al., 2020, we build a URL representation model $f_r : url \to \boldsymbol{x}$, where $\boldsymbol{x} \in \mathbb{R}^h$, by employing the idea of Word2Vec (Mikolov, Sutskever, et al., 2013). Specifically, we use the skip-gram model that given a target URL predicts its context URLs. More formally, the skip-gram model defines the probability that the context (output) URL $url_c \in [url_{t-k}, \ldots, url_{t-1}, url_{t+1}, \ldots, url_{t+k}]$ to have been visited by the user given a target (input) URL url_t as:

$$p(url_c|url_t) \triangleq \frac{\exp(\boldsymbol{v}_{url_c}^\top \boldsymbol{x}_{url_t})}{\sum_{url \in \mathcal{V}} \exp(\boldsymbol{v}_{url}^\top \boldsymbol{x}_{url_t})}, \tag{4.6}$$

where $\boldsymbol{x}_{url}$ and $\boldsymbol{v}_{url}$ denote the columns vectors (correspond to url) of the input $\boldsymbol{X} \in \mathbb{R}^{h \times v}$ and output $\boldsymbol{V} \in \mathbb{R}^{h \times v}$ projection matrices, respectively. Therefore, given a sequence of URLs $[url_1, \ldots, url_\tau]$, our objective is the maximization of the average log probability

$$\frac{1}{\tau} \sum_{t=1}^{\tau} \sum_{-k \leqslant j \leqslant k, j \neq 0} \log p(url_{t+j}|url_t), \tag{4.7}$$

where window k specifies the neighborhood of target URL.

As the direct optimization of Eq. 4.7 is computationally expensive, we are adopting the *negative sampling* approach (Mikolov, Sutskever,

et al., 2013). In negative sampling, we treat the learning of the URL's representation as a binary classification task where we try to distinguish the target-context pairs of URLs presented on the training data (i. e. , positive pairs) from those that are not (i. e. , negative pairs). Following the suggestions of Mikolov, Sutskever, et al., 2013, for each positive pair we are creating multiple negative pairs.

Let's now assume that we have trained our URL2VEC representation model f_r and therefore we have access to the embedding matrix $\boldsymbol{X} \in \mathbb{R}^{h \times v}$ for a finite size vocabulary $\mathcal{V}$ of URLs ($|\mathcal{V}| = v$). The i^{th} column, $\boldsymbol{x}_i \in \mathbb{R}^h$, corresponds to the embedding vector of the i^{th} URL of $\mathcal{V}$. We also denote as $\boldsymbol{w} \in \mathbb{R}^v$ the user representation with each entry to be the normalized inverse document frequency of a URL (see Eq. 4.5). Then, we present the audience expansion schemes that are based on the URL embeddings returned by our URL2VEC model.

Average URL embeddings is a simple but effective way to calculate a vector representation for a user u. Given $\boldsymbol{X}$ and nBoW vector $\boldsymbol{d}$ of user, the user's representation can be easily calculated as:

$$\boldsymbol{z} = \boldsymbol{X}\boldsymbol{d}. \tag{4.8}$$

We also consider the weighted average of the URL embeddings to represent a user:

$$\boldsymbol{z} = \boldsymbol{X}\boldsymbol{w}. \tag{4.9}$$

Having computed the vector representations $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ of the two users u_1 and u_2, their affinity score can be calculated either by using cosine similarity

$$sim(\boldsymbol{z}, \boldsymbol{z}') = \frac{\boldsymbol{z}^\top \boldsymbol{z}'}{\|\boldsymbol{z}\|_2 \|\boldsymbol{z}'\|_2}, \tag{4.10}$$

or euclidean distance, $D(\boldsymbol{z}_1, \boldsymbol{z}_2) = \|\boldsymbol{z}_1 - \boldsymbol{z}_2\|_2$,

$$sim(\boldsymbol{z}, \boldsymbol{z}') = \frac{1}{1 + D(\boldsymbol{z}_1, \boldsymbol{z}_2)}. \tag{4.11}$$

Gaussian user representation Nikolentzos et al., 2017 model each user as a multivariate Gaussian distribution. Specifically, we assume that the embedding vectors of the URLs visited by user u are generated by a multivariate Gaussian distribution:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{4.12}$$

where $\boldsymbol{\mu} \in \mathbb{R}^h$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{h \times h}$ are its mean vector and covariance matrix, respectively. The *maximum likelihood* solution (Bishop, 2006) for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ is given by the sample mean (see Eq. 4.8) and the empirical covariance matrix:

$$\boldsymbol{\Sigma} = \frac{1}{\|\boldsymbol{b}\|_1} (\boldsymbol{U} - \boldsymbol{\mu})(\boldsymbol{U} - \boldsymbol{\mu})^\top, \tag{4.13}$$

where $\boldsymbol{U} \triangleq \boldsymbol{X} diag(\boldsymbol{b})$ and $diag(\boldsymbol{b}) \in \mathbb{R}^{v \times v}$ returns a square diagonal matrix with the elements of vector $\boldsymbol{b}$ on the main diagonal and zeros on the entries outside of the main diagonal.

The similarity between two users u_1 and u_2 is set equal to the convex combination between the similarities of their mean vectors $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ and their covariance matrices $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$:

$$sim(u, u') \triangleq \alpha sim(\boldsymbol{\mu}, \boldsymbol{\mu}') + (1 - \alpha) sim(\boldsymbol{\Sigma}, \boldsymbol{\Sigma}'), \tag{4.14}$$

where $\alpha \in [0, 1]$. The similarity between the mean vectors $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ is computed using cosine similarity (see Eq. 4.10) and the similarity between matrices $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$ is computed as:

$$sim(\boldsymbol{\Sigma}, \boldsymbol{\Sigma}') = \frac{\sum \boldsymbol{\Sigma} \circ \boldsymbol{\Sigma}'}{\|\boldsymbol{\Sigma}\|_F \times \|\boldsymbol{\Sigma}'\|_F}, \tag{4.15}$$

where $(\cdot \circ \cdot)$ is the element-wise (Hadamard) product, $\|\cdot\|_F$ is the Frobenius norm.

Word Mover's Distance (WMD) is a powerful method introduced by Kusner et al., 2015 for measuring the dissimilarity between two documents. WMD builds upon the idea of Tao, Cuturi, and Yamamoto, 2012 that uses the Earth Mover's Distance (*EMD* [1], Rubner, Tomasi, and Guibas, 1998) to compute the distance between two documents. Let $\boldsymbol{d}$, $\boldsymbol{d}'$ be the v-dimensional nBoW vectors for two users, and $\boldsymbol{T} \in \mathbb{R}^{v \times v}$ be an auxiliary 'transport' matrix where $\boldsymbol{T}_{ij} \geqslant 0$ describes how much of URL i in $\boldsymbol{d}$ travels to URL j in $\boldsymbol{d}'$. Formally, WMD solves the following linear program in order to learn $\mathbf{T}$ that minimize the cumulative cost of moving $\boldsymbol{d}$ to $\boldsymbol{d}'$:

$$D(u, u') = \min_{\boldsymbol{T} \geqslant 0} \sum_{i,j}^{v} \boldsymbol{T}_{ij} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2, \quad \text{s.t.} \tag{4.16}$$

$$\sum_{j}^{v} \boldsymbol{T}_{ij} = d_i \text{ and } \sum_{i}^{v} \boldsymbol{T}_{ij} = d'_j, \quad \forall i, j \in \{1, \ldots, v\},$$

where d_i (d'_i) is the i^{th} element of vector $\boldsymbol{d}$ ($\boldsymbol{d}'$).

A relaxed version of WMD, called *RWMD*, has been also introduced by Kusner et al., 2015 to accelerate WMD that comes with the cost of high complexity. To achieve this, we relax the WMD optimization problem by removing one of the two constraints. The solution of RWMD yields a lower bound for the WMD, and is much faster. Next, we propose a normalized version of WMD that considers the diversity of the visited URL:

$$D(u, u') = \frac{D(u, u')}{\sum_{i=1}^{v} \mathbb{1}_{d_i > 0} \sum_{i=1}^{v} \mathbb{1}_{d'_i > 0}}. \tag{4.17}$$

In the end, having computed the WMD distance between users u and u', we calculate their similarity using Eq. 4.11.

1. EMD is also known as the Wasserstein distance (Levina and Bickel, 2001).

4.2.3 *Audience expansion based on* User2Vec *model*

Paragraph Vectors is an self-supervised learning algorithm introduced first in Q. Le and Mikolov, 2014. It is able to learn vector representations of input sequences (sequences of URLs in our case) of variable length. Specifically, each user is now mapped to a single vector, represented by a column of the projection matrix $\boldsymbol{Z} \in \mathbb{R}^{h \times n}$ (n is equal to the number of users in data), that aims to capture the interests of the user. Similar to the Url2Vec model, every URL is also mapped to a vector represented by a column of the projection matrix $\boldsymbol{X} \in \mathbb{R}^{h \times v}$. Then, the user and context URL representation vectors are averaged to predict next URL (url_t) in context:

$$p(url_t | \boldsymbol{b}_c, \boldsymbol{z}) = \frac{\exp(\boldsymbol{v}_{url_t}^\top (\boldsymbol{X}\boldsymbol{b}_c + \boldsymbol{z}))}{\sum_{url' \in \mathcal{V}} \exp(\boldsymbol{v}_{url'}^\top (\boldsymbol{X}\boldsymbol{b}_c + \boldsymbol{z}))}, \tag{4.18}$$

where $\boldsymbol{v}_{url}$ denotes the column in output projection matrix $\boldsymbol{V}$ for url, $\boldsymbol{b}_c$ is the BoW representation vector of the local context $[url_{t-k}, \ldots, url_{t-1}]$ and $\boldsymbol{z} \in \boldsymbol{Z}$ is the vector representation of the user. This model is known as Distributed Memory (DM) model.

A simplified version of the DM model, called Distributed Bag of Words (DBOW) model, ignores the context information in the input and predicts URLs randomly sampled by the user's browsing history (see Eq. 4.2):

$$p(url_t | \boldsymbol{z}) = \frac{\exp(\boldsymbol{v}_{url_t}^\top \boldsymbol{z})}{\sum_{url' \in \mathcal{V}} \exp(\boldsymbol{v}_{url'}^\top \boldsymbol{z})}, \tag{4.19}$$

Both DM and DBOW models consist of two phases: i) the learning phase of the URL embeddings ($\boldsymbol{X}$ and $\boldsymbol{V}$ projection matrices), and ii) the inference phase where we get the user embeddings $\boldsymbol{Z}$ by keeping fixed the other parameters of the model. Furthermore we can also represent a user as a combination of the two vectors returned by the DM and DBOW models. We call it CON and it actually concatenates the two vectors returned by the DM and DBOW models.

4.2.4 *Audience expansion based on* User2VecC *model*

The main limitation of the User2Vec model is its complexity that grows linearly with the number of users. Despite the fact that we can limit the size of URLs vocabulary $\mathcal{V}$, the size of training corpus can be extremely large. Apart from that, we need to execute an expensive inference to get the vector representations of unseen users. To overcome all these issues, we adopt Document Vector through Corruption (called User2VecC in our case) model introduced by M. Chen, 2017.

User2VecC model allows us to learn vector representations for unknown users as a simple average of the embeddings of all visited

URLs (just like in the case of the URL2VEC at Sec. 4.2.2). Therefore the model does not need to learn directly a separate projection matrix for representing users anymore. It means that its complexity depends only on the size of URL vocabulary $\mathcal{V}$ and not on the size of the corpus.

Similar to URL2VEC, USER2VECC consists of a projection layer $\boldsymbol{X}$ and an output layer $\boldsymbol{V}$ to predict the target URL. To represent each user we get the average of the embeddings of a set of URLs randomly sampled from the user's browsing history. For this purpose, an unbiased dropout corruption has been adopted to generate a global context $\tilde{\boldsymbol{b}}$ at each update,

$$\tilde{b}_i = \begin{cases} 0, & \text{with probability } q \\ \frac{b_i}{1-q}, & \text{otherwise} \end{cases} \tag{4.20}$$

where $q \in [0, 1]$ is the probability to dropout a URL appearing on the browsing history of a user. Then, the probability of observing a target URL (url_t) given its local $\boldsymbol{b}_c$ and global $\tilde{\boldsymbol{b}}$ contexts is defined as:

$$p(url_t|\boldsymbol{b}_c, \tilde{\boldsymbol{b}}) = \frac{\exp(\boldsymbol{v}_{url_t}^\top(\boldsymbol{X}\boldsymbol{b}_c + \frac{1}{\tau}\boldsymbol{X}\tilde{\boldsymbol{b}}))}{\sum_{url' \in \mathcal{V}} \exp(\boldsymbol{v}_{url'}^\top(\boldsymbol{X}\boldsymbol{b}_c + \frac{1}{\tau}\boldsymbol{X}\tilde{\boldsymbol{b}}))}. \tag{4.21}$$

Finally, having learned projection matrix $\boldsymbol{X}$, we can represent each user as the average (Eq. 4.8) or the weighted average (Eq. 4.9) of the embeddings of their visited URLs.

4.3 EMPIRICAL ANALYSIS

Dataset: We conducted experiments on a real-world RTB dataset[2] in order to evaluate the effectiveness of the different proposed audience expansion schemes. The performance of the audience expansion models have been examined on five advertisers, belonging to four different categories: BANKING, E-SHOP, NEWSPAPER, and TELECOMMUNICATIONS. Specifically, we have used the same dataset with the one used by Qiu et al., 2020. It is an anonymized dataset that has been constructed by using the auction system logs of campaigns launched in France. To be more precise, each record of the dataset corresponds to a chronologically ordered sequence of visited URLs along with a binary label (depends on the advertiser) that indicates whether or not a conversion has happened on the advertiser's website on the next day. The maximum length of a user's browsing history (on a single day) u is set equal to 500, where only the most recently visited URLs are kept, $\tau_u \leqslant 500$.

To be more precise, the dataset consists of sequences of URLs along with their labels of three successive dates, $\mathcal{D}_d$, $\mathcal{D}_{d+1}$, and $\mathcal{D}_{d+2}$. More precisely, $\mathcal{D}_d$ is used for training URL2VEC (Sec. 4.2.2), USER2VEC (Sec.

2. For a full presentation of the dataset please refer to Section 3.4.1 of Chapter 3.

Table 4.2 – Cardinality of seed ($\mathcal{S}$) and candidate ($\mathcal{C}$) sets for each one of the 5 advertisers.

Advertiser Category	**Seed Users** ($\mathcal{S}$)	**Candidate Users** ($\mathcal{C}$)
BANKING	3,746	17,078
E-SHOP	1,463	3,642
NEWSPAPER_1	1,406	5,846
NEWSPAPER_2	1,261	2,582
TELECOM	1,781	4,402

4.2.3) and USER2VECC (Sec. 4.2.4) representation models. As *seed users* $\mathcal{S}$ we consider the *converted* users of $\mathcal{D}_{d+1}$ where the dominant visited URL appears less than 20% on the user's browsing history. We examine the impact of this filtering threshold later in our empirical analysis (Sec. 4.3.2). The candidate set of users $\mathcal{C}$ consists of converted and non-converted users of $\mathcal{D}_{d+2}$, where the set of non-converted users has been randomly selected and its size is equal to that of the converted users. It is worth noting that the seed and candidate sets are different for each advertiser (see Table 4.2).

Representation models setup: Apart from the audience expansion schemes that are based on the Jaccard and Weighted Jaccard affinity metrics (Sec. 4.2.1), the rest expansion schemes assume the existence of a representation model (URL2VEC, USER2VEC, USER2VECC) that projects users into an embedding space. Each domain and user is projected into a 100-dimensional vector. A representation is also learned for the so-called 'rare' domains, respectively. We consider a domain as 'rare' if it appears less than 20 times in the dataset $\mathcal{D}_d$. The number of non-rare domains in $\mathcal{D}_d$ is equal to 22,098 ($|\mathcal{V}| = 22100$ as we also consider the 'rare' and 'unknown' domains). To compute WMD we use the Fast EMD library [3]. Similar to Nikolentzos et al., 2017 we set $\alpha = 0.5$ for URL2VEC+GAUSSIAN scheme. For the training of URL2VEC model the mini-batch stochastic optimization is applied by using Adam optimizer with the default TensorFlow 2.0 settings. In the case of USER2VEC model, the projection matrices $\boldsymbol{X}$ and $\boldsymbol{V}$ are initialised with the URL representation produced by the USER2VEC model. Then, we infer user representations by keeping fixed these two matrices. In USER2VECC model we use the `C` implementation of DOC2VECC [4].

Metrics: To evaluate the different audience expansion schemes we treat audience expansion as a standard binary classification problem. In this way, the affinity score of a user u to the closest *seed user* can be seen as the probability user u to be considered as positive on

3. PyEMD library: https://pypi.org/project/pyemd/
4. DOC2VECC source code: https://github.com/mchen24/iclr2017.

the classification task. To compare the effectiveness of the audience expansion schemes we are using the *area under ROC curve* (AUC) and *Average Precision* (AP, W. Su, Y. Yuan, and M. Zhu, 2015) metrics. More specifically, we consider the average (%) AUC and AP across five independent runs, where each run corresponds to a specific seed used for the initialization of the representation models.

4.3.1 *Results*

Next, we present the numerical results of different audience expansion schemes on five different advertisers. In total we have examined 22 different audience expansion schemes, each belongs to one of the four categories mentioned in Section 4.2:

1. JACCARD and W_JACCARD
2. URL2VEC+{AVG, IDF}+{COSINE,EUCLIDEAN}

   URL2VEC+{GAUSSIAN, WMD}
3. USER2VEC+ {DBOW,DM,CON}+{COSINE,EUCLIDEAN}
4. USER2VECC+{CBOW,SKIPGRAM}+{AVG,IDF}+ {COSINE,EUCLIDEAN}

where AVG (Eq. 4.8) and IDF (Eq. 4.9) refer to the way under which we compute the user representation given URL representations. On the other hand, COSINE (Eq. 4.10) and EUCLIDEAN (Eq. 4.11) indicates the way where we compute the affinity score between two users given their representations.

Tables 4.3 and 4.4 present the average and standard deviation of the AUC and AP across five independent runs, respectively. Based on the average AUC metric, the USER2VECC+CBOW+IDF+EUCLIDEAN model is the most effective audience expansion model as is capable of identifying and top-rank (assign high affinity scores) the users that are highly likely to be converted in the future. Actually, it achieves the best performance in the BANKING and E-SHOP, while it is the second best in the NEWSPAPER_2. On the other hand, URL2VEC+WMD has the best performance according to average AP metric, as it outperforms the other models in 3 out of the 5 advertisers, and is the second best in the other two advertisers.

It can be seen that WEIGHTED_JACCARD outperforms JACCARD according to both metrics. At the same time, its performance is close to these of the other user representation based models as regards the average AUC. Nevertheless, it does not hold on the AP metric, where it is clearly outperformed by the most of the user representation based models.

Among the audience expansion methods that are based to the URL embeddings computed by the URL2VEC model (Sec. 4.2.2), the URL2VEC+WMD is the based one. Also, the performance of the models, that represent users as the weighted average (IDF, Eq. 4.9) of the representations of their visited URLs, is significantly better compared

Table 4.3 – Avg (%) and std of the area under ROC curves (5 independent runs) of the 22 audience expansion models on 5 advertisers. Blue shows best results in the specific category and bold indicates best result for an advertiser.

Method			Adv	Banking	E-shop	Newspaper_1	Newspaper_2	Telecom
			jaccard	63.4 ± (0.000)	65.4 ± (0.000)	73.7 ± (0.000)	69.0 ± (0.000)	60.2 ± (0.000)
			w_jaccard	66.4 ± (0.000)	67.0 ± (0.000)	74.8 ± (0.000)	71.5 ± (0.000)	63.1 ± (0.000)
Url2Vec	avg		cosine	62.2 ± (0.140)	61.7 ± (0.293)	74.5 ± (0.138)	67.0 ± (0.301)	55.8 ± (0.099)
			euclidean	61.7 ± (0.172)	61.1 ± (0.229)	74.0 ± (0.077)	68.4 ± (0.046)	59.0 ± (0.067)
	tdf		cosine	63.7 ± (0.333)	61.6 ± (0.483)	77.0 ± (0.202)	70.5 ± (0.412)	59.8 ± (0.153)
			euclidean	64.3 ± (0.054)	63.4 ± (0.174)	76.8 ± (0.060)	72.0 ± (0.067)	61.6 ± (0.033)
			gaussian	61.7 ± (0.108)	63.1 ± (0.143)	73.1 ± (0.150)	67.1 ± (0.301)	55.7 ± (0.180)
			wmd	66.2 ± (0.083)	66.0 ± (0.032)	76.6 ± (0.036)	**74.0 ± (0.020)**	65.0 ± (0.030)
User2Vec	dbow		cosine	64.2 ± (0.281)	64.3 ± (0.443)	75.6 ± (0.109)	70.0 ± (0.492)	58.6 ± (0.178)
			euclidean	65.4 ± (0.218)	65.4 ± (0.387)	76.7 ± (0.214)	71.1 ± (0.345)	61.9 ± (0.339)
	dm		cosine	66.5 ± (0.131)	66.1 ± (0.130)	75.0 ± (0.189)	70.3 ± (0.055)	**66.8 ± (0.067)**
			euclidean	64.7 ± (0.237)	64.8 ± (0.154)	72.9 ± (0.191)	67.9 ± (0.109)	65.0 ± (0.154)
	con		cosine	67.0 ± (0.138)	66.7 ± (0.134)	76.2 ± (0.111)	71.4 ± (0.122)	66.2 ± (0.141)
			euclidean	66.3 ± (0.170)	66.0 ± (0.145)	75.4 ± (0.169)	70.5 ± (0.086)	66.0 ± (0.253)
User2Vec+	cbow	avg	cosine	62.5 ± (0.101)	64.3 ± (0.047)	74.7 ± (0.076)	68.8 ± (0.078)	58.0 ± (0.120)
			euclidean	65.2 ± (0.067)	68.0 ± (0.036)	73.1 ± (0.031)	69.9 ± (0.032)	59.4 ± (0.062)
		tdf	cosine	65.7 ± (0.069)	66.0 ± (0.141)	**77.4 ± (0.095)**	71.6 ± (0.121)	62.6 ± (0.204)
			euclidean	**67.6 ± (0.028)**	**68.6 ± (0.040)**	75.3 ± (0.042)	72.1 ± (0.027)	63.8 ± (0.085)
	skipgram	avg	cosine	58.2 ± (0.257)	63.3 ± (0.388)	73.6 ± (0.094)	65.7 ± (0.144)	56.1 ± (0.075)
			euclidean	60.5 ± (0.112)	63.3 ± (0.153)	67.2 ± (0.106)	61.9 ± (0.138)	55.0 ± (0.108)
		tdf	cosine	62.0 ± (0.201)	64.5 ± (0.316)	77.0 ± (0.128)	72.0 ± (0.442)	59.3 ± (0.189)
			euclidean	65.0 ± (0.089)	66.1 ± (0.107)	70.9 ± (0.079)	67.8 ± (0.115)	59.7 ± (0.094)

Table 4.4 – Avg (%) and std of the average precision (5 independent runs) of the 22 audience expansion models on 5 advertisers. Blue shows best results in the specific category and bold indicates best result for an advertiser.

Method			Adv	Banking	E-shop	Newspaper_1	Newspaper_2	Telecom
			jaccard	60.1 ± (0.000)	60.9 ± (0.000)	69.8 ± (0.000)	65.7 ± (0.000)	57.6 ± (0.000)
			w_jaccard	62.3 ± (0.000)	62.8 ± (0.000)	71.7 ± (0.000)	68.0 ± (0.000)	59.9 ± (0.000)
Url2Vec	avg		cosine	59.7 ± (0.079)	59.2 ± (0.306)	75.6 ± (0.079)	63.7 ± (0.255)	54.8 ± (0.145)
			euclidean	59.9 ± (0.086)	59.8 ± (0.148)	75.8 ± (0.033)	66.6 ± (0.091)	58.0 ± (0.055)
	tdf		cosine	59.9 ± (0.189)	59.2 ± (0.223)	77.5 ± (0.198)	68.6 ± (0.339)	58.3 ± (0.069)
			euclidean	63.0 ± (0.047)	62.4 ± (0.162)	**78.0 ± (0.064)**	71.9 ± (0.061)	60.7 ± (0.028)
			gaussian	58.9 ± (0.095)	59.8 ± (0.130)	73.4 ± (0.264)	63.9 ± (0.250)	54.6 ± (0.177)
			wmd	**65.2 ± (0.043)**	65.1 ± (0.042)	77.8 ± (0.019)	**74.6 ± (0.004)**	**65.0 ± (0.027)**
User2Vec	dbow		cosine	61.1 ± (0.258)	61.2 ± (0.321)	76.5 ± (0.110)	66.8 ± (0.371)	57.1 ± (0.229)
			euclidean	62.7 ± (0.191)	62.6 ± (0.176)	77.2 ± (0.128)	68.4 ± (0.317)	59.8 ± (0.368)
	dm		cosine	64.2 ± (0.108)	63.3 ± (0.135)	75.7 ± (0.203)	68.2 ± (0.142)	64.1 ± (0.096)
			euclidean	63.3 ± (0.234)	62.3 ± (0.217)	73.1 ± (0.244)	66.5 ± (0.129)	63.5 ± (0.101)
	con		cosine	64.2 ± (0.118)	63.8 ± (0.089)	77.0 ± (0.099)	69.4 ± (0.238)	63.4 ± (0.204)
			euclidean	64.7 ± (0.180)	63.8 ± (0.098)	76.5 ± (0.080)	69.1 ± (0.138)	63.8 ± (0.296)
User2Vec+	cbow	avg	cosine	60.7 ± (0.084)	61.1 ± (0.031)	74.8 ± (0.044)	65.3 ± (0.060)	56.9 ± (0.120)
			euclidean	62.9 ± (0.043)	64.7 ± (0.043)	74.1 ± (0.017)	67.8 ± (0.067)	58.7 ± (0.059)
		tdf	cosine	62.6 ± (0.035)	63.0 ± (0.121)	77.4 ± (0.083)	69.7 ± (0.108)	60.7 ± (0.089)
			euclidean	64.5 ± (0.034)	**65.8 ± (0.026)**	75.8 ± (0.056)	71.3 ± (0.070)	62.1 ± (0.064)
	skipgram	avg	cosine	56.6 ± (0.173)	59.7 ± (0.244)	73.0 ± (0.107)	62.0 ± (0.044)	55.4 ± (0.080)
			euclidean	59.9 ± (0.122)	62.2 ± (0.140)	68.0 ± (0.145)	60.7 ± (0.189)	54.1 ± (0.061)
		tdf	cosine	57.8 ± (0.086)	61.0 ± (0.285)	76.5 ± (0.119)	69.3 ± (0.368)	58.4 ± (0.141)
			euclidean	63.9 ± (0.092)	**65.8 ± (0.129)**	72.3 ± (0.127)	68.7 ± (0.122)	59.5 ± (0.076)

to the performance of the schemes that use the average (AVG, Eq. 4.8) URL representations to represent the users. Another remark is that the models compute the affinity scores between candidate and seed users based on the euclidean distance (EUCLIDEAN, Eq. 4.11) performs better compared to the models that use the cosine similarity (COSINE, Eq. 4.10). Additionally, the performance of the URL2VEC+GAUSSIAN is close to that of URL2VEC+AVG+COSINE. Actually, it does not surprise as both use the empirical mean to generate user representation and the cosine similarity to check the similarity between users.

Let's now have a closer look on the performance of the methods that are based on the USER2VEC model (Sec. 4.2.3). Based on our results, the best performance is achieved by using the user representations

produced by concatenating the representations returned by the other two models, DM and DBOW. Moreover, the USER2VEC+DM model outperforms the USER2VEC+DBOW model. As regards the similarity measures, we don't have a clear winner. Specifically, it seems that euclidean distance performs better when it is combined with the USER2VEC+DBOW model.

Concerning the performance of the methods that are based on the USER2VECC model (Sec. 4.2.4), we can see that CBOW constantly outperforms SKIPGRAM. Specifically, USER2VECC+CBOW+IDF+EUCLIDEAN achieves the best performance. Additionally, similar to the URL2VEC based models, it is more preferable to represent users as the weighted average (IDF, Eq. 4.9) of the representation of their visited URLs, and to use euclidean distance (EUCLIDEAN, Eq. 4.11) to compute the affinity scores between candidate and seed users.

To conclude, our empirical results indicate that the performance of the URL2VEC+WMD is competitive or better to that of USER2VECC+CBOW +IDF+EUCLIDEAN. Nevertheless, the computation of the similarity between users by using WMD is expensive. The same also holds for the USER2VEC based audience expansion schemes, as we need to execute an expensive inference step to get the vector representations of unseen users. Apart from that, the complexity of USER2VEC based schemes is coupled from the size of the training set. This is not acceptable in display advertising, as the number of different users can become extremely large (millions or billions). On the other hand, these issues do not exist on the URL2VEC and USER2VECC based schemes that represent users by just computing the average or weighted average URL representations of their browsing history. Taking all the above into account, we can safely conclude that USER2VECC+CBOW+IDF+EUCLIDEAN is the best audience scheme on display advertising as its performance is quite competitive compared to the other schemes, while on the same the inference of the users representations is not computationally expensive. Also, its performance is competitive to the prediction conversion models presented at Qiu et al., 2020.

Figure 4.1 illustrates the URL representations learned by URL2VEC, USER2VECC+CBOW, and USER2VECC+SKIPGRAM models using the Barnes-Hut t-SNE algorithm (Maaten, 2014). On the other hand, Figure 4.2 visualizes the user representations learned by URL2VEC+IDF, USER2VECC +{CBOW,SKIPGRAM}+IDF, and USER2VEC+CONCAT models on the NEWSPAPER_2 advertiser. As we can see, the seed users (red points) are closer to the positive candidate users (green points) on the embedding space compared to the negative candidate users (blue points). Moreover, we can verify that all these four schemes are able to distinguish the positive from negative candidate users as they create two clusters of candidate users.

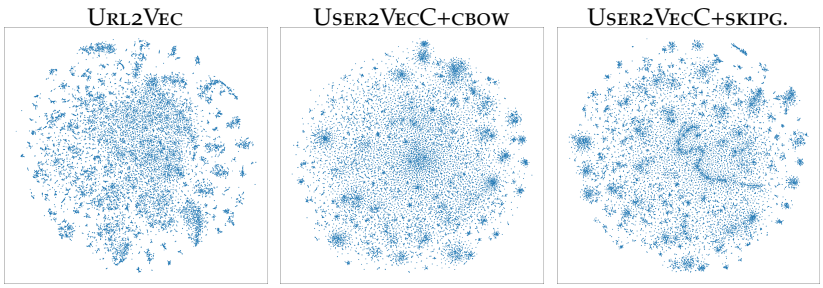


Figure 4.1 – t-SNE visualization of the URL representation vectors ($\boldsymbol{X}$ embedding matrix) learned by URL2VEC, USER2VECC+CBOW, USER2VECC+SKIPGRAM models.

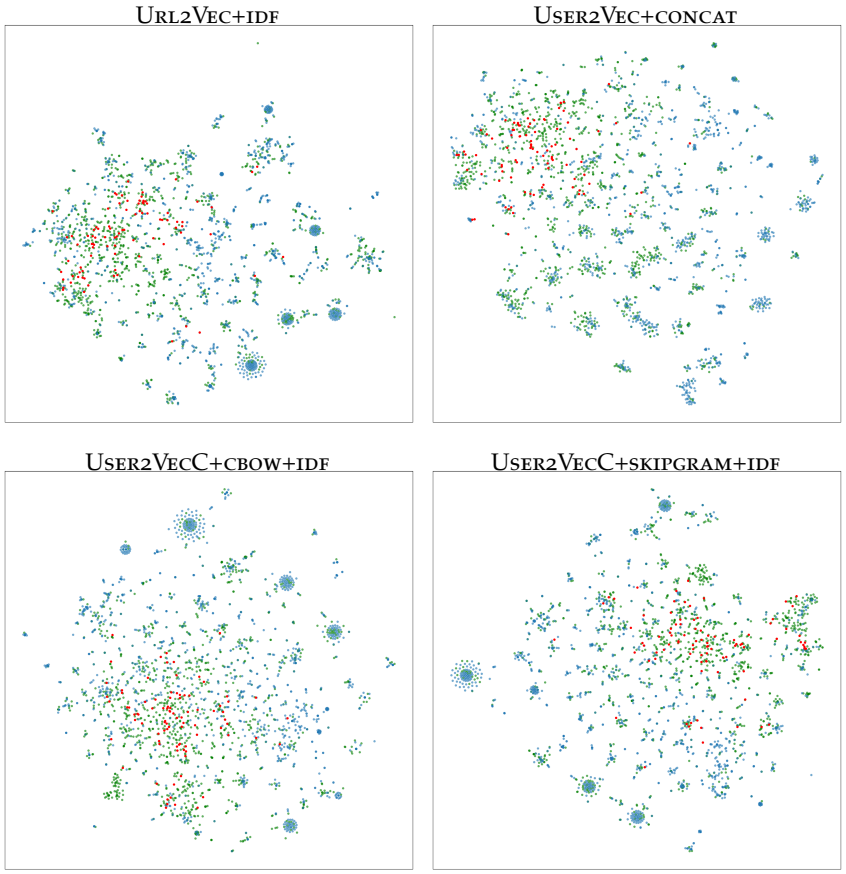


Figure 4.2 – t-SNE visualization of the user representations produced by four representation models. The red points indicate the seed users, the green points indicate the positive candidate users, and the blue points indicate the negative candidate users on the NEWSPAPER_2 dataset.

4.3.2 *Ablation study*

As aforementioned, we consider as *seed users* the converted users on $\mathcal{D}_{d+1}$ where the dominant visited URL appears less than 20%

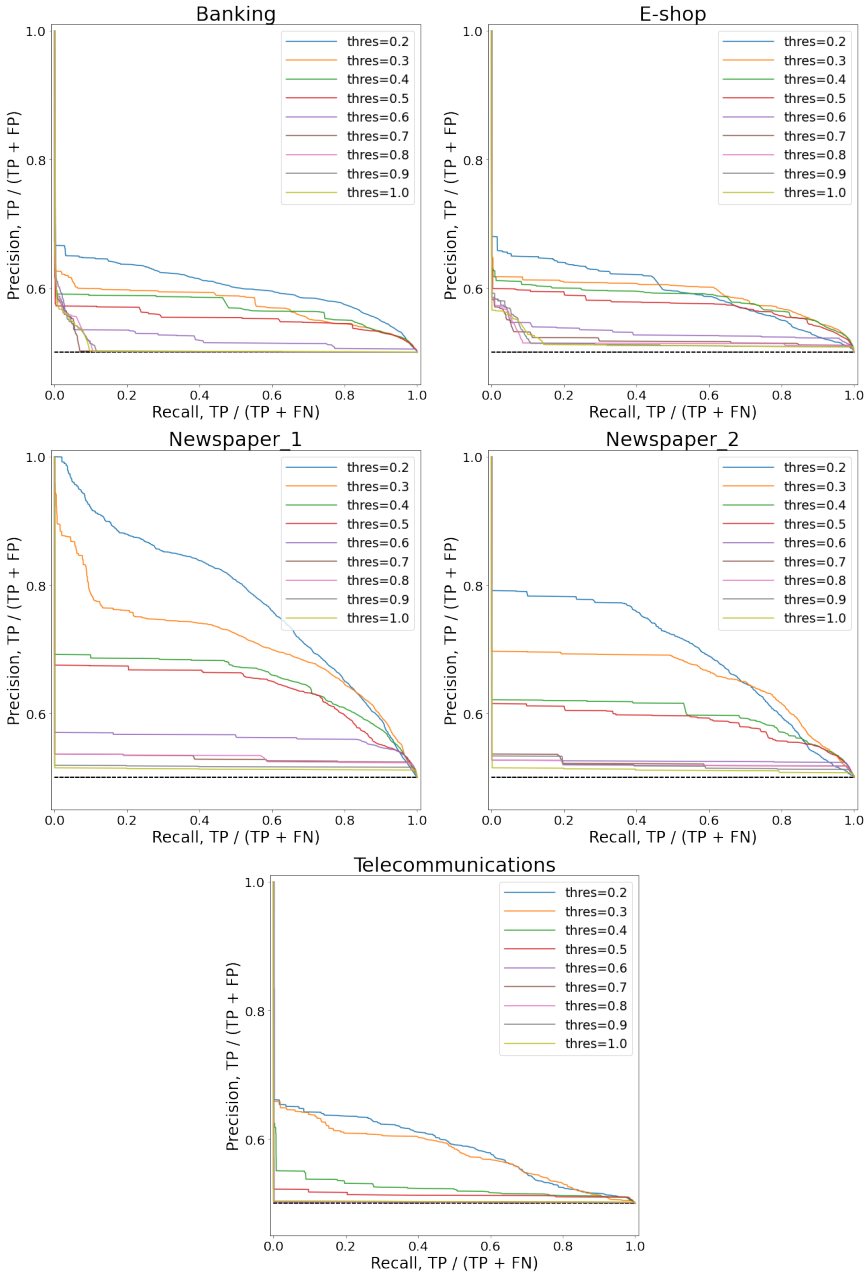


Figure 4.3 – Average precision-recall curves (5 independent runs) of URL2VEC+IDF+COSINE audience expansion model on the five advertisers for different filtering thresholds of seed set $\mathcal{S}$.

of the time in user's browsing. In this way, we filter out the sequences of URLs dominated by a single URL probably due to the automatic refresh of the specific website. For instance, if we set the filtering threshold to 0.9 we will include on the seed set sequences where a single URL is possible to appear on the 90% of the time. Let's now examine the impact of this free hyper-parameter on the performance of our proposed audience expansion schemes. Specifically, Fig. 4.3 presents the average precision-recall curves for the

URL2VEC+IDF+COSINE scheme on the five advertisers for different filtering thresholds, thres $= \{0.2, 0.3, \ldots, 1\}$. It is clear that as we relax this threshold the performance of the URL2VEC+IDF+COSINE is getting worse and worse. Specifically, in the case where we set thres > 0.6, it behaves almost in a random way without being able to identify the positive candidates with success.

4.4 CONCLUSIONS AND FUTURE DIRECTIONS

In this work, we considered the problem of expanded advertisers' audiences in display advertising. In this direction, we introduced and examined different audience expansion schemes that are classified into four main categories. Apart from the audience expansion systems of the first category that represent users as a set of the visited URLs, all the others are based on self-supervision representation models. Specifically, the representation models that used to represent users (URL2VEC, USER2VEC, USER2VECC) are inspired by NLP. With the help of the pre-trained representation models, we can easily compute the affinity scores between candidate and seed users by using any standard computationally efficient similarity metric. In the future, we intend to examine representation models that consider the chronological order of visited URLs (Mikolov, Karafiát, et al., 2010). Additionally, we plan to evaluate the effectiveness of the proposed audience expansion systems on real-world online scenario.

5 CONCLUDING REMARKS

In this chapter, we conclude the dissertation by first summarizing our main contributions described in Chapters 3 and 4. Then we outline the promising future research directions unexplored at the time of writing.

5.1 SUMMARY OF CONTRIBUTIONS

In the context of online advertising, especially in Real-Time Bidding, how to deliver the right ad, to the right person, in the right context (webpage), at the right time is essential for advertisers. Thus, accurately identifying a user's value to the advertiser is of high importance. Under this context, in this dissertation, we focused on developing novel approaches to address two challenging display advertising tasks: *conversion prediction* and *audience expansion*.

CONVERSION PREDICTION Conversion prediction task aims to predict if a user will convert (e. g. , visit website, buy product) or not at a specific moment. In our work (Qiu et al., 2020, see Chapter 3 for details), we considered the case where the objective is to predict whether a user will visit the advertiser's website the next day based on his browsing history. Due to the high cardinality and diversity of URLs, we introduced three self-supervised schemes that are able to learn dense and meaningful URL embeddings. Specifically: 1) Each URL (after removing the http(s) protocol component) is split to tokens by '/', constrained to have maximum 3 tokens. 2) By using only the user browsing history, a skip-gram model is applied to learn URL embedding, where each URL is treated as a word and the user's browsing history as a document. Moreover, the URL embedding is represented as an aggregation of its token embeddings, either by using the embedding of its first token (*Domain_only*), or by averaging (*Token_avg*) or concatenating (*Token_concat*) its token embeddings. 3) In the end, the user's browsing history is given as input into an neural network architecture that consists of i) a URL embedding layer, ii) an aggregation layer, iii) and a dense layer with sigmoid activation function and outputs the user conversion probability. The empirical results showed that our proposed URL embedding models are able to produce meaningful URL representations by grouping together URLs of the same category. Furthermore, we saw that user browsing history provides useful information to predict users' visit on the advertiser's website. Finally, considering the chronological order of the visited

URLs (using RNN-based model in aggregation layer) significantly improves the model's performance.

AUDIENCE EXPANSION Given a set of seed users provided by the advertiser, the objective of the audience expansion task is to discover more users with similar interests. In our work (Tziortziotis et al., 2021, see Chapter 4 for more details), the converted users are considered as the seed users, and by using only their browsing history we try find users with similar behavior. The proposed similarity-based audience expansion schemes rank each candidate user based on its similarity to the seed users, and selects the top-ranked users as the expanded user set. Specifically, the proposed audience expansion schemes can be grouped into four main categories, where except for the first category that each user is represented as a set of the visited URLs, all the others are based on self-supervised representation models for learning user representations. More precisely, user embeddings are learned in a self-supervised way based on user browsing histories where the users and the visited URLs are considered as documents and words, respectively. Then, each user is either considered as a sequence of URL embeddings (URL2Vec model) or as a single vector (User2Vec & User2VecC model). Apart from that, we also introduced a data-drive weight factor (IDF) for each URL in order to intentionally alleviate the frequency redundancy of URLs that affected by the website refreshment. Our experiments shown that the proposed User2vecC+CBOW+IDF+EUCLIDEAN audience expansion scheme is the best choice among all the proposed schemes due to its competitive performance and its fast inference speed.

5.2 FUTURE DIRECTIONS

An essential building block that is shared by our proposed conversion prediction and audience expansion models is the learning of meaning URL embeddings. The process that we have followed to laern URL embedding consists of three main steps: i) URLs tokenization, ii) aggregate URL's token embeddings to form the URL embedding, and iii) then the URLs' embeddings are learned through an auxiliary (classification) task. Next, we present possible ways that could be followed to improve these three steps of the proposed URL representation learning architecture.

- Tokenization
  In our applied URL embedding methods, we split the URL by '/' to get its tokens. This simple tokenization is not robust to the slight change of the token characters. For instance, the domain 'ABC.com' and 'ABC.fr' should share a certain degree of similarity. Nevertheless, our tokenization approach treat these as two completely different tokens. In this direction, it would be may

useful to consider more sophisticated tokenization techniques that extract informative subwords (Sennrich, Haddow, and Birch, 2016; Y. Wu et al., 2016; Kudo and J. Richardson, 2018).

- Token aggregation
  By averaging token embeddings to construct URL embedding, we ignore the sequential property and hierarchy of the tokens appeared on a URL. Methods such as positional embedding (Devlin et al., 2018), RNN based models (David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, 1988; J. Cheng, Dong, and Lapata, 2016) that preserve the sequential property of the tokens may be used. Also, to better capture different meaning of one token in different positions (e. g. , "apple" as domain name refers to the company Apple and "apple" in other position may refer to the fruit), the idea of contextual representation methods (Devlin et al., 2018; Peters et al., 2018) which provide an adaptive meaning of token according to its applied context (e. g. , surrounding tokens) may be useful to construct a better URL embedding.
- Time gap between URL pairs consideration
  The position of URLs is used for labeling the URL pairs. It should be noticed that these labeled URLs are then used for the training of the URL embedding model. Precisely, for a given URL, all its neighbors in the browsing history are considered as positive, without taking into account the time gap between them. However, it is possible that two URLs next to each other are with a long time gap, which indicates that they are not visited together. Therefore, an interesting future direction is the consideration also of the time gap between visited URLs during the labeling process of the URL pairs.

Another interesting direction of future work would be the further improvement of the rest blocks of the proposed model architectures.

CONVERSION PREDICTION Basically, user's browsing history is a chronologically ordered sequence of URLs. We have have examined its sequential aspect by using an RNN-based model that uplifts the conversion prediction model's performance (Chapter 3). Adding the chronological aspect allows us to fully model user's browsing activities. Therefore, using specific time-aware models such as time-LSTM (Y. Zhu et al., 2017) or time-aware attention (P.-C. Chen et al., 2017; Cai et al., 2018) may be beneficial. For instance, D. Gligorijevic, J. Gligorijevic, and Flores, 2020 propose a prediction model with time-aware attention block to model heterogeneous sequences of users' activities.

AUDIENCE EXPANSION As we have already introduced in Sec.1.6.3, seed users typically contain noise (e. g. , outliers, subgroups). Our experiments in Chapter 4 showed that filtering seed users by only

keeping users whose dominant URL appears less than 20% on the total browsing history improve the model's results. Therefore, we believe that a 'clean' and purified seed users set improves model performance. In this direction, Zhuang et al., 2020 propose the usage of a knowledge distillation mechanism (G. Hinton, Vinyals, and Dean, 2015), to eliminate the gap between the seeds and the actual audiences introduced by the provided seed set.

5.3 EPILOGUE

Advertising shows a powerful influence on people's daily life. Throughout this dissertation, we have presented our understanding of this area, Real-time Bidding in particular, and provided our contributions for optimizing customer prospecting through examining challenging real-world problems. Although numerous work has been conducted in this domain, a number of unanswered questions and challenging problems remain. I sincerely hope that the findings in this dissertation will provide some insight into future research and applications that ultimately improves the advertising ecosystem, especially the user experience.

BIBLIOGRAPHY

Jaccard, Paul (1901). « Étude comparative de la distribution florale dans une portion des Alpes et des Jura. » In: *Bulletin del la Société Vaudoise des Sciences Naturelles* 37, pp. 547–579 (cit. on p. 103).

Pearson, K. (1901). « On Lines and Planes of Closest Fit to Systems of Points in Space. » In: *Philosophical Magazine* 2, pp. 559–572 (cit. on p. 39).

Jaccard, Paul (1912). « THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. » In: *New Phytologist* 11.2, pp. 37–50 (cit. on p. 31).

Eckart, C. and G. Young (1936). « The approximation of one matrix by another of lower rank. » In: *Psychometrika* 1.3, pp. 211–218 (cit. on p. 34).

Robbins, Herbert and Sutton Monro (1951). « A Stochastic Approximation Method. » In: *The Annals of Mathematical Statistics* 22.3, pp. 400–407 (cit. on p. 37).

Huffman, David A. (1952). « A Method for the Construction of Minimum-Redundancy Codes. » In: *Proceedings of the IRE*, pp. 1098–1101 (cit. on p. 63).

Harris, Zellig (1954). « Distributional structure. » In: *Word* 10.2-3, pp. 146–162 (cit. on pp. 22, 60).

Bellman, Richard (1957). *Dynamic Programming*. 1st ed. Princeton, NJ, USA: Princeton University Press (cit. on pp. 22, 25, 58).

Luhn, H. P. (1957). « A Statistical Approach to Mechanized Encoding and Searching of Literary Information. » In: *IBM Journal of Research and Development* 1.4, pp. 309–317 (cit. on p. 73).

Cox, David R. (1958). « The Regression Analysis of Binary Sequences (with Discussion). » In: *J Roy Stat Soc B* 20, pp. 215–242 (cit. on p. 41).

Hubel, David H. and Torsten N. Wiesel (1959). « Receptive Fields of Single Neurons in the Cat's Striate Cortex. » In: *Journal of Physiology* 148, pp. 574–591 (cit. on p. 53).

Bellman, Richard (1961). *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press (cit. on pp. 22, 25, 58).

Vickrey, William (1961). « Counterspeculation, Auctions, And Competitive Sealed Tenders. » In: *Journal of Finance* 16.1, pp. 8–37 (cit. on p. 10).

A. N. Tikhonov (1963). « On the solution of ill-posed problems and the method of regularization. » In: *Dokl. Akad. Nauk SSSR* 151.2, pp. 501–504 (cit. on p. 44).

MacQueen, J. (1967). « Some methods for classification and analysis of multivariate observations. » In: (cit. on p. 39).

Golub, G. H. and C. Reinsch (1970). « Singular Value Decomposition and Least Squares Solutions. » In: *Numer. Math.* 14.5, pp. 403–420 (cit. on pp. 34, 60).

Hoerl, A. E. and R. W. Kennard (1970). « Ridge Regression: Biased Estimation for Nonorthogonal Problems. » In: *Technometrics* 12, pp. 55–67 (cit. on p. 45).

Wilson, Dennis L. (1972). « Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. » In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-2.3, pp. 408–421 (cit. on p. 26).

Jones, Karen Sparck (1973). « Index term weighting. » In: *Information Storage and Retrieval* 9.11, pp. 619–633. ISSN: 0020-0271 (cit. on pp. 100, 104).

Stone, M. (1974). « Cross-validatory choice and assessment of statistical predictions. » In: *Roy. Stat. Soc.* 36, pp. 111–147 (cit. on p. 45).

Hartigan, J. A. and M. A. Wong (1979). « A K-Means Clustering Algorithm. » In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 28.1, pp. 100–108 (cit. on p. 30).

Myerson, Roger B. (1981). « Optimal Auction Design. » In: *Math. Oper. Res.* 6.1, pp. 58–73 (cit. on p. 10).

Rabin, M.O. (1981). *Fingerprinting by Random Polynomials*. Center for Research in Computing Technology: Center for Research in Computing Technology. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ. (cit. on p. 14).

Pollay, Richard (Apr. 1986). « The Distorted Mirror: Reflections on the Unintended Consequences of Advertising. » In: *Journal of Marketing* 50, pp. 18–36 (cit. on p. 1).

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1988). « Neurocomputing: Foundations of Research. » In: MIT Press. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6 (cit. on p. 54).

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1988). « Learning Representations by Back-Propagating Errors. » In: *Neurocomputing: Foundations of Research*, pp. 696–699 (cit. on pp. 82, 119).

Sparck Jones, Karen (1988). « A Statistical Interpretation of Term Specificity and Its Application in Retrieval. » In: *Document Retrieval Systems*. GBR: Taylor Graham Publishing, pp. 132–142. ISBN: 0947568212 (cit. on pp. 31, 60, 72).

Hanson, Stephen and Lorien Pratt (1989). « Comparing Biases for Minimal Network Construction with Back-Propagation. » In: *Advances in Neural Information Processing Systems*. Vol. 1. Morgan-Kaufmann (cit. on p. 45).

Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman (1990). « Indexing by latent semantic analysis. » In: *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE* 41.6, pp. 391–407 (cit. on p. 61).

Werbos, Paul (Nov. 1990). « Backpropagation through time: what it does and how to do it. » In: *Proceedings of the IEEE* 78, pp. 1550–1560 (cit. on p. 54).

Geman, Stuart, Elie Bienenstock, and René Doursat (1992). « Neural Networks and the Bias/Variance Dilemma. » In: *Neural Computation* 4.1, pp. 1–58 (cit. on p. 44).

Broder, Andrei Z. (1993). « Some applications of Rabin's fingerprinting method. » In: *Sequences II: Methods in Communications, Security, and Computer Science*. Springer-Verlag, pp. 143–152 (cit. on p. 14).

Matthews, Steven A. (1995). *A Technical Primer on Auction Theory I: Independent Private Values*. Tech. rep. Northwestern University, Center for Mathematical Studies in Economics and Management Science (cit. on p. 11).

Vapnik, Vladimir N. (1995). *The Nature of Statistical Learning Theory*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0387945598 (cit. on pp. 42, 43).

Breiman, Leo (1996). « Bagging Predictors. » In: *Mach. Learn.* 24.2, pp. 123–140. ISSN: 0885-6125 (cit. on p. 44).

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu (1996). « A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. » In: KDD'96. Portland, Oregon: AAAI Press, pp. 226–231 (cit. on p. 39).

Tibshirani, R. (1996). « Regression Shrinkage and Selection via the Lasso. » In: *Journal of the royal statistical society series b-methodological* 58, pp. 267–288 (cit. on p. 45).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). « Long Short-Term Memory. » In: *Neural Computation* 9.8, pp. 1735–1780 (cit. on pp. 54, 82).

Stehman, Stephen (Oct. 1997). « Selecting and interpreting measures of thematic classification accuracy. » In: *Remote Sensing of Environment* 62, pp. 77–89 (cit. on p. 46).

Hochreiter, Sepp (1998). « The Vanishing Gradient Problem during Learning Recurrent Neural Nets and Problem Solutions. » In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 6.2, pp. 107–116 (cit. on p. 53).

Rubner, Y., C. Tomasi, and L. J. Guibas (1998). « A metric for distributions with applications to image databases. » In: *ICCV* (cit. on p. 107).

Lee, Daniel and H. Seung (Nov. 1999). « Learning the Parts of Objects by Non-Negative Matrix Factorization. » In: *Nature* 401, pp. 788–91 (cit. on p. 60).

Elkan, Charles (2001). « The Foundations of Cost-Sensitive Learning. » In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 973–978 (cit. on p. 26).

Friedman, Jerome H. (2001). « Greedy function approximation: A gradient boosting machine. » In: *The Annals of Statistics* 29, pp. 1189–1232 (cit. on p. 25).

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc. (cit. on p. 39).

King, Gary and Langche Zeng (2001). « Logistic Regression in Rare Events Data. » In: *Political Analysis* 9 (2), pp. 137–163 (cit. on p. 26).

Kristol, David M. (2001). « HTTP Cookies: Standards, Privacy, and Politics. » In: *ACM Trans. Internet Technol.* 1.2, pp. 151–198 (cit. on p. 13).

Laurikkala, Jorma (2001). « Improving Identification of Difficult Small Classes by Balancing Class Distribution. » In: *Artificial Intelligence in Medicine*. Springer Berlin Heidelberg, pp. 63–66 (cit. on p. 26).

Levina, E. and P. Bickel (2001). « The Earth Mover's distance is the Mallows distance: some insights from statistics. » In: *ICCV* (cit. on p. 107).

Rosario, B. (2001). « Latent Semantic Indexing : An Overview 1 Latent Semantic Indexing : An overview INFOSYS 240 Spring 2000 Final Paper. » In: (cit. on p. 61).

Charikar, Moses S. (2002). « Similarity estimation techniques from rounding algorithms. » In: *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, pp. 380–388 (cit. on p. 15).

Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer (2002). « SMOTE: Synthetic Minority over-Sampling Technique. » In: *J. Artif. Int. Res.* 16.1, pp. 321–357. ISSN: 1076-9757 (cit. on p. 26).

Church, Kenneth and Patrick Hanks (July 2002). « Word Association Norms, Mutual Information, and Lexicography. » In: *Computational Linguistics* 16 (cit. on p. 60).

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Janvin (Mar. 2003). « A Neural Probabilistic Language Model. » In: *J. Mach. Learn. Res.* 3.null, pp. 1137–1155. ISSN: 1532-4435 (cit. on p. 64).

Linden, G., B. Smith, and J. York (2003). « Amazon.com recommendations: item-to-item collaborative filtering. » In: *IEEE Internet Computing* 7.1, pp. 76–80 (cit. on p. 38).

Liu, Bing, Yang Dai, Xiaoli Li, Wee Sun Lee, and Philip S. Yu (2003). « Building Text Classifiers Using Positive and Unlabeled Examples. » In: *Proceedings of the Third IEEE International Conference on Data Mining*. ICDM '03. USA: IEEE Computer Society, p. 179 (cit. on p. 29).

Zhang, J. and I. Mani (2003). « KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. » In: *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets* (cit. on p. 26).

Dumais, S.T. (Jan. 2004). « Latent Semantic Analysis. » In: *Annual Review of Information Science and Technology* 38, pp. 188–230 (cit. on pp. 61, 73).

Milgrom, Paul (2004). *Putting Auction Theory to Work*. Churchill Lectures in Economics. Cambridge University Press (cit. on p. 11).

Turney, Peter D. (2004). « Human-Level Performance on Word Analogy Questions by Latent Relational Analysis » (cit. on p. 61).

Han, Hui, Wen-Yuan Wang, and Bing-Huan Mao (2005). « Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. » In: *Advances in Intelligent Computing*, pp. 878–887 (cit. on p. 26).

Morin, Frederic and Yoshua Bengio (2005). « Hierarchical Probabilistic Neural Network Language Model. » In: *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pp. 246–252 (cit. on p. 63).

Rokach, Lior and Oded Maimon (2005). « Clustering Methods. » In: *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, pp. 321–352 (cit. on p. 39).

Zhu, Xiaojin (2005). *Semi-Supervised Learning Literature Survey*. Tech. rep. Computer Sciences, University of Wisconsin-Madison (cit. on p. 40).

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag. ISBN: 0387310738 (cit. on pp. 39, 80, 106).

Cybenkot, G (2006). « Approximation by Superpositions of a Sigmoidal Function *. » In: (cit. on p. 56).

Davis, Jesse and Mark Goadrich (June 2006). « The Relationship Between Precision-Recall and ROC Curves. » In: vol. 06 (cit. on pp. 48, 49).

Fawcett, Tom (2006). « An introduction to ROC analysis. » In: *Pattern Recognition Letters* 27.8, pp. 861–874 (cit. on pp. 47, 48).

Liu, Xu-ying and Zhi-hua Zhou (2006). « The Influence of Class Imbalance on Cost-Sensitive Learning: An Empirical Study. » In: *Sixth International Conference on Data Mining (ICDM'06)*, pp. 970–974 (cit. on p. 26).

Anagnostopoulos, Aris, Andrei Z. Broder, Evgeniy Gabrilovich, Vanja Josifovski, and Lance Riedel (2007). « Just-in-Time Contextual Advertising. » In: *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*. CIKM '07, pp. 331–340 (cit. on p. 15).

Edelman, Benjamin, Michael Ostrovsky, and Michael Schwarz (2007). « Internet Advertising and the Generalized Second-Price Auction: Selling Billions of Dollars Worth of Keywords. » In: *American Economic Review* 97.1, pp. 242–259 (cit. on p. 10).

Mahdian, Mohammad and Kerem Tomak (2007). « Pay-per-action Model for Online Advertising. » In: *ADKDD* (cit. on p. 77).

Richardson, Matthew, Ewa Dominowska, and Robert Ragno (2007). « Predicting Clicks: Estimating the Click-through Rate for New Ads. » In: *Proceedings of the 16th International Conference on World Wide Web WWW*. Association for Computing Machinery, pp. 521–530 (cit. on pp. 25, 79).

Yao, Yuan, Lorenzo Rosasco, and Andrea Caponnetto (Aug. 2007). « On Early Stopping in Gradient Descent Learning. » In: *Constructive Approximation* 26, pp. 289–315 (cit. on pp. 44, 56).

Bottou, Léon and Olivier Bousquet (2008). « The Tradeoffs of Large Scale Learning. » In: *Advances in Neural Information Processing Systems*. Vol. 20. Curran Associates, Inc. (cit. on pp. 43, 44).

He, Haibo, Yang Bai, Edwardo A. Garcia, and Shutao Li (2008). « ADASYN: Adaptive synthetic sampling approach for imbalanced learning. » In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 1322–1328 (cit. on p. 26).

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze (2008). *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press (cit. on p. 72).

Slaney, M. and M. Casey (2008). « Locality-Sensitive Hashing for Finding Nearest Neighbors [Lecture Notes]. » In: *IEEE Signal Processing Magazine* 25.2, pp. 128–131 (cit. on pp. 30, 101).

Eckersley, Peter (2009). *How Unique Is Your Web Browser?* Tech. rep. Electronig Frontier Foundation (cit. on p. 14).

Niu, Xiaofei, Jun Ma, and Dongmei Zhang (2009). « A Survey of Contextual Advertising. » In: *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*. Vol. 7, pp. 505–509 (cit. on p. 15).

Train, Kenneth E. (2009). *Discrete Choice Methods with Simulation*. 2nd ed. Cambridge University Press (cit. on p. 27).

Weinberger, Kilian, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg (2009). « Feature Hashing for Large Scale Multitask Learning. » In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09, pp. 1113–1120 (cit. on p. 25).

Graepel, T., J. Q. Candela, T. Borchert, and R. Herbrich (2010). « Web-scale Bayesian Click-through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. » In: *ICML* (cit. on p. 79).

Ioffe, S. (2010). « Improved Consistent Sampling, Weighted Minhash and L1 Sketching. » In: *2010 IEEE International Conference on Data Mining*, pp. 246–255. DOI: 10.1109/ICDM.2010.80 (cit. on pp. 31, 103).

Mikolov, Tomas, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur (2010). « Recurrent neural network based language model. » In: *INTERSPEECH* (cit. on p. 116).

Nair, Vinod and Geoffrey E. Hinton (2010). « Rectified Linear Units Improve Restricted Boltzmann Machines. » In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, pp. 807–814. ISBN: 9781605589077 (cit. on pp. 38, 81).

Pergelova, Albena, Diego Prior, and Rialp Josep (Sept. 2010). « Assessing Advertising Efficiency. » In: vol. 39, pp. 39–54 (cit. on p. 1).

Rendle, Steffen (2010). « Factorization Machines. » In: *The 10th IEEE International Conference on Data Mining ICDM*. IEEE Computer Society, pp. 995–1000 (cit. on p. 26).

Chen, Ye, Pavel Berkhin, Bo Anderson, and Nikhil R. Devanur (2011). « Real-time Bidding Algorithms for Performance-based Display Ad Allocation. » In: *KDD* (cit. on p. 78).

Gerber, Alan S., James G. Gimpel, Donald P. Green, and Daron R. Shaw (2011). « How Large and Long-lasting Are the Persuasive Effects of Televised Campaign Ads? Results from a Randomized Field Experiment. » In: *American Political Science Review* 105.1, pp. 135–150 (cit. on p. 1).

Google (2011). *The arrival of real-time bidding* (cit. on p. 77).

Loh, Wei-Yin (Jan. 2011). « Classification and Regression Trees. » In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1, pp. 14–23 (cit. on p. 57).

McMahan, Brendan (2011). « Follow-the-Regularized-Leader and Mirror Descent: Equivalence Theorems and L1 Regularization. » In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Vol. 15. Proceedings of Machine Learning Research. PMLR, pp. 525–533 (cit. on p. 60).

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. Vol. 385. Studies in Computational Intelligence. Springer. ISBN: 978-3-642-24796-5 (cit. on p. 78).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). « ImageNet Classification with Deep Convolutional Neural Networks. » In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc. (cit. on pp. 26, 38, 52).

Tao, Jin, Marco Cuturi, and Akihiro Yamamoto (2012). « A Distance Between Text Documents based on Topic Models and Ground Metric Learning. » In: *JSAI* (cit. on p. 107).

Bengio, Y., A. Courville, and P. Vincent (2013). « Representation Learning: A Review and New Perspectives. » In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8, pp. 1798–1828 (cit. on p. 79).

Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton (2013). « Speech recognition with deep recurrent neural networks. » In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649 (cit. on p. 53).

McMahan, H. Brendan et al. (2013). « Ad Click Prediction: a View from the Trenches. » In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (cit. on pp. 60, 79).

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). « Efficient Estimation of Word Representations in Vector Space. » In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings* (cit. on pp. 22, 24, 28, 30, 40, 57, 62, 73, 100).

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). « Distributed Representations of Words and Phrases and their Compositionality. » In: *NIPS* (cit. on pp. 22, 62, 63, 78, 79, 83, 84, 100, 105, 106).

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). *On the difficulty of training Recurrent Neural Networks*. arXiv: 1211.5063 [cs.LG] (cit. on p. 53).

Yuan, Shuai, Jun Wang, and Xiaoxue Zhao (2013). « Real-Time Bidding for Online Advertising: Measurement and Analysis. » In: *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising*. ADKDD '13 (cit. on p. 3).

Zeiler, Matthew D. and Rob Fergus (2013). « Visualizing and Understanding Convolutional Networks. » In: *CoRR* abs/1311.2901. arXiv: 1311.2901 (cit. on p. 57).

Chapelle, O. (2014). « Modeling delayed feedback in display advertising. » In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (cit. on p. 27).

Chapelle, O., E. Manavoglu, and R. Rosales (2014). « Simple and Scalable Response Prediction for Display Advertising. » In: *ACM Trans. Intell. Syst. Technol.* 5.4 (cit. on p. 79).

Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv: 1406.1078 (cit. on pp. 24, 27, 53, 54).

Dauphin, Yann, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio (2014). *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*. arXiv: 1406.2572 [cs.LG] (cit. on p. 55).

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). *Generative Adversarial Networks*. arXiv: 1406.2661 [stat.ML] (cit. on p. 39).

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). *Neural Turing Machines*. arXiv: 1410.5401 [cs.NE] (cit. on p. 68).

He, X. et al. (2014). « Practical Lessons from Predicting Clicks on Ads at Facebook. » In: *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising, ADKDD* (cit. on pp. 25, 79).

Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom (June 2014). « A Convolutional Neural Network for Modelling Sentences. » In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 655–665 (cit. on p. 52).

Karpathy, Andrej, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei (2014). « Large-Scale Video Classification with Convolutional Neural Networks. » In: CVPR '14. IEEE Computer Society, pp. 1725–1732 (cit. on p. 52).

Kim, Yoon (2014). *Convolutional Neural Networks for Sentence Classification*. arXiv: 1408.5882 [cs.CL] (cit. on p. 52).

Kingma, Diederik P and Max Welling (2014). *Auto-Encoding Variational Bayes*. arXiv: 1312.6114 [stat.ML] (cit. on p. 39).

Le, Quoc and Tomas Mikolov (2014). « Distributed Representations of Sentences and Documents. » In: *ICML* (cit. on pp. 22, 31, 73, 74, 100, 108).

Maaten, Laurens van der (2014). « Accelerating t-SNE using Tree-Based Algorithms. » In: *Journal of Machine Learning Research* 15.93, pp. 3221–3245 (cit. on p. 113).

Mordelet, F. and J.-P. Vert (2014). « A bagging SVM to learn from positive and unlabeled examples. » In: *Pattern Recognition Letters* 37, pp. 201–209 (cit. on p. 29).

Oentaryo, Richard et al. (2014). « Detecting Click Fraud in Online Advertising: A Data Mining Approach. » In: *J. Mach. Learn. Res.* 15.1, pp. 99–140. ISSN: 1532-4435 (cit. on pp. 21, 27).

Oentaryo, Richard J., Ee-Peng Lim, Jia-Wei Low, David Lo, and Michael Finegold (2014). « Predicting Response in Mobile Advertising with Hierarchical Importance-aware Factorization Machine. » In: *WSDM* (cit. on p. 79).

Pennington, Jeffrey, Richard Socher, and Christopher Manning (Jan. 2014). « Glove: Global Vectors for Word Representation. » In: vol. 14, pp. 1532–1543 (cit. on pp. 30, 61).

Qu, Yan, Jing Wang, Yang Sun, and Hans Marius Holtan (2014). *Systems and methods for generating expanded user segments*. US Patent 8,655,695 (cit. on p. 102).

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). « Dropout: A Simple Way to Prevent Neural Networks from Overfitting. » In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958 (cit. on p. 56).

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). « Sequence to Sequence Learning with Neural Networks. » In: *CoRR* abs/1409.3215. arXiv: 1409.3215 (cit. on pp. 53, 67).

Yuan, Shuai, Jun Wang, Bowei Chen, Peter Mason, and Sam Seljan (2014). « An empirical study of reserve price optimisation in real-time bidding. » In: *Proceedings of the 20th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, KDD, pp. 1897–1906 (cit. on p. 11).

Zhang, Y., H. Dai, C. Xu, J. Feng, T. Wang, J. Bian, B. Wang, and T.-Y. Liu (2014). « Sequential Click Prediction for Sponsored Search with Recurrent Neural Networks. » In: *AAAI* (cit. on p. 79).

Chapelle, Olivier, Eren Manavoglu, and Romer Rosales (2015). « Simple and Scalable Response Prediction for Display Advertising. » In: *ACM Trans. Intell. Syst. Technol.* 5.4 (cit. on p. 25).

Choromanska, Anna, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun (2015). *The Loss Surfaces of Multilayer Networks*. arXiv: 1412.0233 [cs.LG] (cit. on p. 55).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). « Deep Residual Learning for Image Recognition. » In: *CoRR* abs/1512.03385. arXiv: 1512.03385 (cit. on pp. 38, 49, 52, 55).

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (2015). *Distilling the Knowledge in a Neural Network*. arXiv: 1503.02531 [stat.ML] (cit. on pp. 31, 120).

Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv: 1502.03167 [cs.LG] (cit. on p. 56).

Kusner, Matt, Yu Sun, Nicholas Kolkin, and Kilian Weinberger (2015). « From Word Embeddings To Document Distances. » In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. Proceedings of Machine Learning Research. PMLR, pp. 957–966 (cit. on pp. 31, 100, 107).

Liu, Qiang, Feng Yu, Shu Wu, and Liang Wang (2015). « A Convolutional Click Prediction Model. » In: *CIKM* (cit. on pp. 26, 79).

Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning (2015). *Effective Approaches to Attention-based Neural Machine Translation*. arXiv: 1508.04025 [cs.CL] (cit. on p. 68).

Manic, Marius (2015). « The Rise of native advertising. » In: *Bulletin of the Transilvania University of Brasov. Series V : Economic Sciences*, pp. 53–58 (cit. on p. 15).

Simonyan, Karen and Andrew Zisserman (2015). « Very Deep Convolutional Networks for Large-Scale Image Recognition. » In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (cit. on p. 52).

Su, Wanhua, Yan Yuan, and Mu Zhu (2015). « A Relationship between the Average Precision and the Area Under the ROC Curve. » In: *ICTIR* (cit. on p. 111).

Ta, A. (2015). « Factorization machines with follow-the-regularized-leader for CTR prediction in display advertising. » In: *IEEE Big Data* (cit. on p. 79).

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). *Layer Normalization*. arXiv: 1607.06450 [stat.ML] (cit. on p. 56).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2016). *Neural Machine Translation by Jointly Learning to Align and Translate*. arXiv: 1409.0473 [cs.CL] (cit. on pp. 38, 67, 68).

Barkan, Oren and Noam Koenigstein (2016). *Item2Vec: Neural Item Embedding for Collaborative Filtering* (cit. on p. 22).

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomás Mikolov (2016). « Enriching Word Vectors with Subword Information. » In: *CoRR* abs/1607.04606. arXiv: 1607.04606 (cit. on p. 63).

Cheng, Heng-Tze et al. (2016). « Wide & Deep Learning for Recommender Systems. » In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. DLRS 2016, pp. 7–10 (cit. on p. 26).

Cheng, Jianpeng, Li Dong, and Mirella Lapata (2016). « Long Short-Term Memory-Networks for Machine Reading. » In: *CoRR* abs/1601.06733. arXiv: 1601.06733 (cit. on pp. 24, 68, 119).

Covington, Paul, Jay Adams, and Emre Sargin (2016). « Deep Neural Networks for YouTube Recommendations. » In: *RecSys* (cit. on pp. 30, 101).

Grbovic, M., N. Djuric, V. Radosavljevic, F. Silvestri, R. Baeza-Yates, A. Feng, E. Ordentlich, L. Yang, and G. Owens (2016). « Scalable Semantic Matching of Queries to Ads in Sponsored Search Advertising. » In: *SIGIR* (cit. on p. 79).

Grover, Aditya and Jure Leskovec (2016). « Node2vec: Scalable Feature Learning for Networks. » In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (cit. on p. 22).

Hidasi, Balázs, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk (2016). *Session-based Recommendations with Recurrent Neural Networks*. arXiv: 1511.06939 [cs.LG] (cit. on p. 56).

Johnson, Melvin et al. (2016). « Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. » In: *CoRR* abs/1611.04558. arXiv: 1611.04558 (cit. on p. 38).

Juan, Yuchin, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin (2016). « Field-Aware Factorization Machines for CTR Prediction. » In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. New York, NY, USA: Association for Computing Machinery, pp. 43–50 (cit. on pp. 25, 26).

Konečný, Jakub, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon (2016). « Federated Learning: Strategies for Improving Communication Efficiency. » In: *NIPS Workshop on Private Multi-Party Machine Learning* (cit. on p. 15).

Kotila, Rumin and Dhar (2016). *Compendium of Ad Frau d Knowledge for Media Investors*. https://fr.slideshare.net/JeffMartinez4/compendium-of-ad-fraud-knowledge-for-media-investors. Accessed: 2021-06-14 (cit. on p. 19).

Liu, Haishan, David Pardoe, Kun Liu, Manoj Thakur, Frank Cao, and Chongzhe Li (2016). « Audience Expansion for Online Social Network Advertising. » In: *KDD* (cit. on pp. 30, 99, 101).

Ma, Q., E. Wagh, J. Wen, Z. Xia, R. Ormandi, and D. Chen (2016). « Score Look-Alike Audiences. » In: *ICDMW* (cit. on pp. 30, 101).

Ma, Qiang, Musen Wen, Zhen Xia, and Datong Chen (2016). « A Sub-linear, Massive-scale Look-alike Audience Extension System A Massive-scale Look-alike Audience Extension. » In: *BigMine workshop at KDD* (cit. on pp. 30, 99, 101).

Paes Leme, Renato, Martin Pal, and Sergei Vassilvitskii (2016). « A Field Guide to Personalized Reserve Prices. » In: WWW '16. International World Wide Web Conferences Steering Committee (cit. on p. 11).

Qu, Yanru, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang (2016). « Product-Based Neural Networks for User Response Prediction. » In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 1149–1154 (cit. on p. 26).

Sennrich, Rico, Barry Haddow, and Alexandra Birch (Aug. 2016). « Neural Machine Translation of Rare Words with Subword Units. » In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1715–1725 (cit. on pp. 23, 119).

Silver, David et al. (2016). « Mastering the Game of Go with Deep Neural Networks and Tree Search. » In: *Nature* 529.7587, pp. 484–489 (cit. on p. 49).

Vasile, Flavian, Elena Smirnova, and Alexis Conneau (2016). « Meta-Prod2Vec - Product Embeddings Using Side-Information for Recommendation. » In: *Proceedings of the 10th ACM Conference on Recommender Systems, RecSys* (cit. on p. 22).

Wu, Yonghui et al. (2016). « Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. » In: *CoRR* abs/1609.08144 (cit. on pp. 23, 38, 119).

Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy (2016). « Hierarchical Attention Networks for Document Classification. » In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pp. 1480–1489 (cit. on p. 68).

Berg, Rianne van den, Thomas N. Kipf, and Max Welling (2017). *Graph Convolutional Matrix Completion*. arXiv: 1706.02263 [stat.ML] (cit. on p. 56).

Chen, Po-Chun, Ta-Chung Chi, Shang-Yu Su, and Yun-Nung Chen (2017). « Dynamic time-aware attention to speaker roles and contexts for spoken language understanding. » In: *2017 IEEE Automatic Speech*

Recognition and Understanding Workshop (ASRU), pp. 554–560 (cit. on p. 119).

Chen, Minmin (2017). « Efficient Vector Representation for Documents through Corruption. » In: *ICLR* (cit. on pp. 31, 73, 74, 100, 108).

Guo, Huifeng, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He (2017). « DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction. » In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI'17. AAAI Press, pp. 1725–1731 (cit. on p. 27).

Justesen, Niels, Philip Bontrager, Julian Togelius, and Sebastian Risi (2017). « Deep Learning for Video Game Playing. » In: *CoRR* abs/1708.07902. arXiv: 1708.07902 (cit. on p. 49).

Kokolakis, Spyros (2017). « Privacy attitudes and privacy behaviour: A review of current research on the privacy paradox phenomenon. » In: *Computers & Security* 64, pp. 122–134 (cit. on p. 14).

Li, Juanjuan, Xiaochun Ni, Yong Yuan, Rui Qin, Xiao Wang, and Fei-Yue Wang (2017). « The impact of reserve price on publisher revenue in real-time bidding advertising markets. » In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (cit. on p. 11).

Narayanan, Annamalai, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal (2017). « graph2vec: Learning Distributed Representations of Graphs. » In: *arXiv e-prints* (cit. on p. 22).

Nikolentzos, Giannis, Polykarpos Meladianos, François Rousseau, Yannis Stavrakas, and Michalis Vazirgiannis (2017). « Multivariate Gaussian Document Representation from Word Embeddings for Text Categorization. » In: *EACL* (cit. on pp. 100, 106, 110).

Szwabe, Andrzej, Pawel Misiorek, and Michal Ciesielczyk (2017). « Logistic Regression Setup for RTB CTR Estimation. » In: *Proceedings of the 9th International Conference on Machine Learning and Computing, ICMLC* (cit. on p. 26).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). « Attention Is All You Need. » In: *CoRR* abs/1706.03762. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (cit. on pp. 26, 27, 66, 68–70).

Vries, Lisette de, Sonja Gensler, and Peter S.H. Leeflang (2017). « Effects of Traditional Advertising and Social Messages on Brand-Building Metrics and Customer Acquisition. » In: *Journal of Marketing* 81.5, pp. 1–15 (cit. on p. 2).

Wang, J., W. Zhang, and S. Yuan (2017). *Display Advertising with Real-Time Bidding (RTB) and Behavioural Targeting*. Now Publishers Inc. (cit. on pp. 77, 99).

Xiao, Jun, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua (2017). « Attentional Factorization Machines: Learning

the Weight of Feature Interactions via Attention Networks. » In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI'17. AAAI Press, pp. 3119–3125 (cit. on p. 26).

Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals (2017). *Understanding deep learning requires rethinking generalization*. arXiv: 1611.03530 [cs.LG] (cit. on p. 57).

Zhao, Bendong, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu (2017). « Convolutional neural networks for time series classification. » In: *Journal of Systems Engineering and Electronics* 28.1, pp. 162–169 (cit. on p. 52).

Zhu, Yu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai (2017). « What to Do Next: Modeling User Behaviors by Time-LSTM. » In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 3602–3608 (cit. on p. 119).

Cai, Xiangrui, Jinyang Gao, Kee Yuan Ngiam, Beng Chin Ooi, Ying Zhang, and Xiaojie Yuan (2018). « Medical Concept Embedding with Time-Aware Attention. » In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI'18. Stockholm, Sweden: AAAI Press, pp. 3984–3990. ISBN: 9780999241127 (cit. on p. 119).

Chan, P. P. K., X. Hu, L. Zhao, D. S. Yeung, D. Liu, and L. Xiao (2018). « Convolutional Neural Networks based Click-Through Rate Prediction with Multiple Feature Sequences. » In: *IJCAI* (cit. on p. 79).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). « BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. » In: *CoRR* abs/1810.04805. arXiv: 1810.04805 (cit. on pp. 22, 30, 40, 49, 55, 64, 66, 70, 71, 119).

Kang, Wang-Cheng and Julian McAuley (2018). *Self-Attentive Sequential Recommendation*. arXiv: 1808.09781 [cs.IR] (cit. on p. 56).

Kudo, Taku (2018). « Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. » In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 66–75 (cit. on p. 23).

Kudo, Taku and John Richardson (2018). « SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. » In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, pp. 66–71 (cit. on pp. 23, 119).

Le, H., Q. Pham, D. Sahoo, and S. C. H. Hoi (2018). « URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection. » In: *CoRR* (cit. on pp. 24, 79).

Leevy, Joffrey L., T. Khoshgoftaar, Richard A. Bauder, and Naeem Seliya (2018). « A survey on addressing high-class imbalance in big data. » In: *Journal of Big Data* 5, pp. 1–30 (cit. on p. 26).

Li, Chunyuan, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski (2018). *Measuring the Intrinsic Dimension of Objective Landscapes*. arXiv: 1804.08838 [cs.LG] (cit. on p. 57).

Liu, Peter J., Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer (2018). « Generating Wikipedia by Summarizing Long Sequences. » In: *ICLR* (cit. on p. 66).

Pan, Junwei, Jian Xu, A. L. Ruiz, W. Zhao, Shengjun Pan, Yu Sun, and Q. Lu (2018). « Field-weighted Factorization Machines for Click-Through Rate Prediction in Display Advertising. » In: *Proceedings of the 2018 World Wide Web Conference* (cit. on p. 26).

Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018). *Deep contextualized word representations*. arXiv: 1802.05365 [cs.CL] (cit. on pp. 30, 64, 65, 119).

Pham, Thuy Thi Thanh, Van Nam Hoang, and Thanh Ngoc Ha (2018). « Exploring Efficiency of Character-Level Convolution Neuron Network and Long Short Term Memory on Malicious URL Detection. » In: *Proceedings of the 2018 VII International Conference on Network, Communication and Computing (ICNCC)* (cit. on p. 24).

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second. The MIT Press (cit. on p. 40).

Tolomei, Gabriele, M. Lalmas, A. Farahat, and Andrew Haines (2018). « You must have clicked on this ad by mistake! Data-driven identification of accidental clicks on mobile ads with applications to advertiser cost discounting and click-through rate prediction. » In: *International Journal of Data Science and Analytics* 7, pp. 53–66 (cit. on p. 27).

Yoshikawa, Y. and Yusaku Imai (2018). « A Nonparametric Delayed Feedback Model for Conversion Rate Prediction. » In: *ArXiv* abs/1802.00255 (cit. on p. 27).

Yuan, Huaping, Zhenguo Yang, X. Chen, Yukun Li, and W. Liu (2018). « URL2Vec: URL Modeling with Character Embeddings for Fast and Accurate Phishing Website Detection. » In: *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pp. 265–272 (cit. on p. 24).

Zhou, Guorui, Xiaoqiang Zhu, Chengru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai (2018). « Deep Interest Network for Click-Through Rate Prediction. » In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge*

Discovery & Data Mining, KDD. ACM, pp. 1059–1068 (cit. on pp. 26, 79).

Arora, Sanjeev, Hrishikesh Khandeparkar, Mikhail Khodak, Orestis Plevrakis, and Nikunj Saunshi (2019). « A Theoretical Analysis of Contrastive Unsupervised Representation Learning. » In: *CoRR* abs/1902.09229. arXiv: 1902.09229. URL: http://arxiv.org/abs/1902.09229 (cit. on p. 40).

Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal (2019). *Reconciling modern machine learning practice and the bias-variance trade-off*. arXiv: 1812.11118 [stat.ML] (cit. on pp. 43, 56).

Chen, Qiwei, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou (2019). « Behavior Sequence Transformer for E-commerce Recommendation in Alibaba. » In: *CoRR* abs/1905.06874. arXiv: 1905.06874 (cit. on p. 38).

Dada, Emmanuel Gbenga, Joseph Stephen Bassi, Haruna Chiroma, Shafi'i Muhammad Abdulhamid, Adebayo Olusola Adetunmbi, and Opeyemi Emmanuel Ajibuwa (2019). « Machine learning for email spam filtering: review, approaches and open research problems. » In: *Heliyon* 5.6. ISSN: 2405-8440 (cit. on p. 38).

deWet, Stephanie and Jiafan Ou (2019). « Finding Users Who Act Alike: Transfer Learning for Expanding Advertiser Audiences. » In: *KDD* (cit. on pp. 30, 99, 101).

Doan, Khoa D., Pranjul Yadav, and Chandan K. Reddy (2019). « Adversarial Factorization Autoencoder for Look-Alike Modeling. » In: *CIKM* (cit. on pp. 101, 102).

Grigorescu, Sorin Mihai, Bogdan Trasnea, Tiberiu T. Cocias, and Gigel Macesanu (2019). « A Survey of Deep Learning Techniques for Autonomous Driving. » In: *CoRR* abs/1910.07738. arXiv: 1910.07738 (cit. on p. 38).

Hendrycks, Dan, Mantas Mazeika, Saurav Kadavath, and Dawn Song (2019). « Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty. » In: *CoRR* abs/1906.12340. arXiv: 1906.12340 (cit. on p. 40).

Jiang, Jinling, Xiaoming Lin, Junjie Yao, and Hua Lu (2019). « Comprehensive Audience Expansion based on End-to-End Neural Prediction. » In: *eCom at SIGIR* (cit. on pp. 29, 102).

Junczys-Dowmunt, Marcin (2019). « Microsoft Translator at WMT 2019: Towards Large-Scale Document-Level Neural Machine Translation. » In: *Proceedings of the Fourth Conference on Machine Translation, WMT 2019, Florence, Italy, August 1-2, 2019 - Volume 2: Shared Task Papers, Day 1*. Ed. by Ondrej Bojar et al., pp. 225–233 (cit. on p. 66).

Liu, Bin, Ruiming Tang, Yingzhi Chen, Jinkai Yu, Huifeng Guo, and Yuzhou Zhang (2019). « Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction. » In: *The World Wide Web Conference* (cit. on p. 26).

Liu, Yudan, Kaikai Ge, Xu Zhang, and Leyu Lin (2019). « Real-time Attention Based Look-alike Model for Recommender System. » In: *KDD* (cit. on pp. 30, 99, 101).

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). « Language Models are Unsupervised Multitask Learners. » In: (cit. on p. 66).

Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar (2019). *On the Convergence of Adam and Beyond*. arXiv: 1904.09237 [cs.LG] (cit. on p. 36).

Shao, Taihua, Yupu Guo, Honghui Chen, and Zepeng Hao (2019). « Transformer-Based Neural Network for Answer Selection in Question Answering. » In: *IEEE Access*, pp. 26146–26156 (cit. on p. 66).

Sun, Fei, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang (2019). *BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer*. arXiv: 1904.06690 [cs.IR] (cit. on p. 56).

Zeithammer, Robert (2019). « Soft Floors in Auctions. » In: *Manag. Sci.* 65.9, pp. 4204–4221 (cit. on p. 11).

Zhang, Zheng (2019). « Explorations in Word Embeddings : graph-based word embedding learning and cross-lingual contextual word embedding learning. » PhD thesis. Université Paris-Saclay. URL: https://tel.archives-ouvertes.fr/tel-02366013 (cit. on p. 60).

Bekker, Jessa and Jesse Davis (2020). « Learning from positive and unlabeled data: a survey. » In: *Mach. Learn.* 109.4, pp. 719–760 (cit. on p. 29).

Blaise, Agathe, Mathieu Bouet, Vania Conan, and Stefano Secci (2020). « BotFP: FingerPrints Clustering for Bot Detection. » In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–7 (cit. on p. 21).

Brown, Tom B. et al. (2020). *Language Models are Few-Shot Learners*. arXiv: 2005.14165 [cs.CL] (cit. on pp. 49, 55).

Cunningham, Padraig and Sarah Jane Delany (2020). *k-Nearest Neighbour Classifiers: 2nd Edition (with Python examples)*. arXiv: 2004.04523 [cs.LG] (cit. on p. 57).

Gligorijevic, Djordje, Jelena Gligorijevic, and Aaron Flores (2020). « Prospective Modeling of Users for Online Display Advertising via Deep Time-Aware Model. » In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2461–2468 (cit. on p. 119).

Google (2020). *Evaluation of Cohort Algorithms for the FLoC API*. https://github.com/google/ads-privacy/blob/master/proposals/FLoC/FLOC-Whitepaper-Google.pdf. Accessed: 2021-06-14 (cit. on p. 15).

IAB (2020). *IAB Europe's Guide to Ad Fraud*. https://iabeurope.eu/wp-content/uploads/2020/12/IAB-Europe-Guide-to-Ad-Fraud-1.pdf. Accessed: 2021-06-14 (cit. on p. 19).

Kidger, Patrick and Terry Lyons (2020). *Universal Approximation with Deep Narrow Networks*. arXiv: 1905.08539 [cs.LG] (cit. on p. 56).

Laperdrix, Pierre, Nataliia Bielova, Benoit Baudry, and Gildas Avoine (2020). « Browser Fingerprinting: A Survey. » In: *ACM Trans. Web* 14.2 (cit. on p. 14).

Lee, Jason D., Qi Lei, Nikunj Saunshi, and Jiacheng Zhuo (2020). *Predicting What You Already Know Helps: Provable Self-Supervised Learning*. arXiv: 2008.01064 [cs.LG] (cit. on p. 40).

Leijnen, Stefan and Fjodor Veen (May 2020). « The Neural Network Zoo. » In: *Proceedings* 47, p. 9 (cit. on p. 52).

Lindsay, Grace W. (2020). « Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future. » In: *Journal of Cognitive Neuroscience*, pp. 1–15 (cit. on p. 53).

Liu, Xiaodong, Kevin Duh, Liyuan Liu, and Jianfeng Gao (2020). « Very Deep Transformers for Neural Machine Translation. » In: *CoRR* abs/2008.07772 (cit. on p. 66).

Liu, Zhining, Xiao-Fan Niu, Chenyi Zhuang, Yize Tan, Yixiang Mu, Jinjie Gu, and Guannan Zhang (2020). « Two-Stage Audience Expansion for Financial Targeting in Marketing. » In: *CIKM* (cit. on pp. 99, 102).

Qiu, Yang, Nikolaos Tziortziotis, Martial Hue, and Michalis Vazirgiannis (2020). « Predicting conversions in display advertising based on URL embeddings. » In: *AdKDD* (cit. on pp. 24, 28, 102, 105, 109, 113, 117).

Staib, Matthew, Sashank J. Reddi, Satyen Kale, Sanjiv Kumar, and Suvrit Sra (2020). *Escaping Saddle Points with Adaptive Gradient Methods*. arXiv: 1901.09149 [cs.LG] (cit. on p. 55).

Su, Y., L. Zhang, Quanyu Dai, B. Zhang, Jinyao Yan, D. Wang, Yongjun Bao, Sulong Xu, Y. He, and W. Yan (2020). « An Attention-based Model for Conversion Rate Prediction with Delayed Feedback via Post-click Calibration. » In: *IJCAI* (cit. on p. 27).

Tan, Mingxing and Quoc V. Le (2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv: 1905.11946 [cs.LG] (cit. on pp. 49, 55).

Yang, Zitong, Yaodong Yu, Chong You, Jacob Steinhardt, and Yi Ma (2020). *Rethinking Bias-Variance Trade-off for Generalization of Neural Networks*. arXiv: 2002.11328 [cs.LG] (cit. on p. 57).

Zhuang, Chenyi et al. (2020). « Hubble: An Industrial System for Audience Expansion in Mobile Marketing. » In: *KDD* (cit. on pp. 30, 99, 102, 120).

Cheng, Yuan and Yanbo Xue (2021). « Looking at CTR Prediction Again: Is Attention All You Need? » In: *CoRR* abs/2105.05563 (cit. on p. 27).

Gharibshah, Zhabiz and Xingquan Zhu (2021). « User Response Prediction in Online Advertising. » In: *ACM Comput. Surv.* 54.3 (cit. on p. 28).

Google (2021). *Building a privacy-first future for web advertising*. https://blog.google/products/ads-commerce/2021-01-privacy-sandbox/. Accessed: 2021-06-14 (cit. on p. 15).

Liu, Xiao, Fanjin Zhang, Zhenyu Hou, Zhaoyu Wang, Li Mian, Jing Zhang, and Jie Tang (2021). *Self-supervised Learning: Generative or Contrastive*. arXiv: 2006.08218 [cs.LG] (cit. on p. 40).

Tziortziotis, Nikolaos, Yang Qiu, Martial Hue, and Michalis Vazirgiannis (2021). « Audience expansion based on user browsing history. » In: *2021 International Joint Conference on Neural Networks (IJCNN)* (cit. on pp. 24, 31, 118).

Wu, Chuhan, Fangzhao Wu, Tao Qi, and Y. Huang (2021). « FeedRec: News Feed Recommendation with Various User Feedbacks. » In: *ArXiv* abs/2102.04903 (cit. on pp. 18, 27).

Xing, Ying, Hui Shu, Hao Zhao, Dannong Li, and Li Guo (Apr. 2021). « Survey on Botnet Detection Techniques: Classification, Methods, and Evaluation. » In: *Mathematical Problems in Engineering* 2021, pp. 1–24 (cit. on p. 21).

Zhang, Weinan, Jiarui Qin, Wei Guo, Ruiming Tang, and Xiuqiang He (2021). *Deep Learning for Click-Through Rate Estimation*. arXiv: 2104.10584 [cs.IR] (cit. on pp. 26, 28).

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` based on `Latex` language. Most of the graphics in this dissertation are generated using the `Matplotlib` library for the Python programming language and online diagram software `diagrams.net`.

ECOLE DOCTORALE

Titre : Méthodes d'optimisation de la prospection client dans l'affichage publicitaire automatisé par enchères en temps réel

Mots clés : plongement d'URL, prédiction de conversion, extension d'audience, enchères en temps réel, réseau de neurones artificiels, apprentissage automatique

Résumé : L'affichage publicitaire en ligne est devenu de plus en plus populaire ces dernières années grâce à l'automatisation du processus d'achat des inventaires. Spécifiquement, les enchères en temps réel (Real-time bidding en anglais, ou RTB) permettent l'échange automatisé d'impressions publicitaires entre les annonceurs et les éditeurs via des enchères en temps réel, au niveau de l'utilisateur. L'objectif principal des campagnes RTB est d'aider les annonceurs à cibler la bonne personne, dans le bon contexte, avec la bonne publicité et au bon moment. Par conséquent, l'identification précise de la 'valeur' d'un utilisateur pour l'annonceur est très importante. Dans ce contexte, nous examinons deux problèmes complexes de l'affichage publicitaire: la prédiction de conversion et l'extension d'audience. Dans les deux tâches, nous considérons uniquement l'historique de navigation de l'utilisateur comme caractéristiques, collectées à partir de données réelles.
Nous examinons d'abord le problème de la prédiction de conversion, où notre objectif est de prédire si un utilisateur se convertira (c'est-à-dire, s'il visitera le site web de l'annonceur, s'il achètera son produit, etc.) ou non vers un annonceur donné. Inspirés par le traitement du langage naturel, nous introduisons trois modèles auto-supervisés de plongement d'URL afin de produire des représentations d'URL sémantiquement significatives. Ensuite, nous avons examiné trois différentes fonctions de projection pour représenter les utilisateurs qui prennent en entrée les représentations d'URL déjà apprises. Enfin, après avoir calculé les représentations des utilisateurs, nous utilisons le modèle de régression logistique pour prédire les probabilités de conversion.
Nous étudions ensuite la tâche de l'extension d'audience dont l'objectif est d'aider les annonceurs à découvrir des audiences présentant des attributs similaires à un public cible (seed users) intéressé par les produits ou services des annonceurs. Dans cette direction, nous proposons différents schémas de l'extension d'audience basés sur la similarité (similarity-based), se concentrant sur l'apprentissage de plongement d'utilisateur de haute qualité de manière auto-supervisée. Les schémas proposés sont capables d'identifier les utilisateurs ayant des intérêts de navigation similaires à ceux des utilisateurs de référence fournis par l'annonceur.

Title : Methods for optimizing customer prospecting in automated display advertising with Real-Time Bidding

Keywords : URL embedding, conversion prediction, audience expansion, real-time bidding, neural networks, machine learning

Abstract : Online display advertising has become more and more popular in recent years thanks to the automation of the ad buying process. Specifically, Real-Time Bidding (RTB) allows the automated trading of ad impressions between advertisers and publishers through real-time auctions, at a per-user level. The primary goal of RTB campaigns is to help advertisers target the right person, in the right context, with the right ad, and at the right time. Therefore, the accurate identification of the 'value' of a user for the advertiser is of high importance. Under this context, we examine two challenging display advertising problems: the conversion prediction and the audience expansion. In both tasks, we consider only the user's browsing history as features, collected from real logged data.
We first examine the conversion prediction problem, where our objective is to predict whether a user will convert (visit the website, buy a product, etc.) or not to a given advertiser. Inspired by natural language processing, we introduce three self-supervised URL embedding models in order to compute semantically meaningful URL representations. Then, we have examined three different mapping functions to represent users that take as input the already learned URL representations. Finally, having computed users' representations, we are using the standard logistic regression model to predict conversion probabilities.
We then investigate the audience expansion task, whose goal is to help advertisers discover audiences with similar attributes to a target audience interested in advertisers' products or services. In this direction, we propose different (similarity-based) audience expansion schemes focusing on the learning of a high-quality user embedding in a self-supervised way. The proposed schemes are able to identify users with similar browsing interests to those of the seed users provided by the advertiser.

Institut Polytechnique de Paris
91120 Palaiseau, France