Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Modeling on-line user behavior using URL embedding

Diploma thesis

Bc. Jonáš Amrich

Study programme: Open Informatics
Master Specialization: Artificial Intelligence
Supervisor: Ing. Ondřej Vaněk, Ph.D.

Prague, May 2017

Thesis Supervisor:
Ing. Ondřej Vaněk, Ph.D.
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2
160 00 Prague 6
Czech Republic

Declaration

I declare that I worked out the presented thesis independently and I quoted all the used sources of information in accord with Methodical instructions about ethical principles for writing an academic thesis.

In Prague, May 2017

.....
Bc. Jonáš Amrich

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science

DIPLOMA THESIS AGREEMENT

Student: Amrich Jonáš

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: Modeling on-line user behavior using URL embedding

Guidelines:

Clickstream prediction and on-line user modeling based on a sequence of visited URL can be approached using word embedding techniques. The objectives of this thesis is to:

1. Study the current state of the art of word embeddings.
2. Understand current approach in clickstream prediction
3. Design and approach for clickstream prediction using URL embedding techniques.
4. Train a set of predictors on a URL dataset provided by Jumpshot.
5. Evaluate the quality of the predictors.

Bibliography/Sources:

[1] Jumpshot URL dataset. Jumpshot. 2016
[2] Stumme, Gerd, Andreas Hotho, and Bettina Berendt. "Semantic web mining: State of the art and future directions." Web semantics: Science, services and agents on the world wide web 4.2 (2006): 124-143.
[3] Abramson, Myriam, and David W. Aha. "What?s in a URL? Genre Classification from URLs." Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence. 2012.
[4] Wang, Gang, et al. "Unsupervised Clickstream Clustering For User Behavior Analysis." SIGCHI Conference on Human Factors in Computing Systems. 2016.

Diploma Thesis Supervisor: Ing. Ondřej Vaněk

Valid until the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Michal Pěchouček, MSc.
Head of Department

prof. Ing. Pavel Ripka,CSc.
Dean

Prague, February 23, 2017

Abstract

The most common method of capturing online user behavior is through a sequence of clicks, which is referred to as a clickstream. This thesis explores methods for clickstream prediction based on URL embeddings. URL embeddings are continuous representations inspired by word embedding techniques. These embeddings, which are learned from clickstream without supervision, capture semantic and syntactic meanings of URLs and are used for reduction of the state space of predictive models.

We present three methods for generation of URL embeddings and evaluate them using a clickstream dataset of more than 12M clicks. One of our proposed methods shows promising results, improving the baseline result and significantly reducing the state space.

Keywords: Clickstream prediction, Online user behavior, Word embeddings, Machine learning, Neural networks, Web mining.

Abstrakt

Nejčastější způsob zaznamenávání chování online uživatelů je pomocí sekvence jejich kliků, jinak také nazývané clickstream. Tato práce zkoumá metody predikce clickstreamu pomocí distribuovaných reprezentací URL. Distribuované reprezentace URL jsou reprezentace ve spojitém prostoru, inspirované podobnými přístupy v oblasti zpracování přirozeného jazyka. Tyto reprezentace, které jsou natrénované bez učitele z clickstreamových dat, zachycují sémantické a syntaktické vlastnosti URL a jsou použity pro zmenšení stavového prostoru prediktivních modelů.

V této práci představujeme tři metody pro generování distribuovaných reprezentací URL, které jsou vyhodnoceny na datasetu o více než 12 milionech kliků. Jedna z představených metod vykazuje nadějné výsledky, které jsou lepší než referenční model a zároveň mnohonásobně zmenšují stavový prostor.

Keywords: Predikce clickstreamu, Chování online uživatelů, Distribuované reprezentace slov, Strojové učení, Neuronové sítě, Web mining.

Acknowledgements

I would like to thank my supervisor Ondřej Vaněk for his valuable advices and consultations, that made this thesis possible. I would also like to thank Ondra Pluskal for his helpful insights and suggestions, and Hong Tsui from Jumpshot Inc. for great support, provided dataset and computational resources.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

List of Tables

List of Figures

Contents

Chapter 1

Introduction

People spend more and more time online and the range of their digital activities is greater than ever. With this rapid increase in past years and a trend that does not seem to get slower anytime soon, the need for a better understanding, modeling and prediction of human online behavior is an important topic. This need, together with recent advances in computational abilities and datasets with billions of events drives research in this field.

Online behavior is most commonly recorded as a sequence of events that were performed by users, and this sequence is usually referred to as a clickstream; every event in the clickstream usually represents change of a URL. The task of predictive modeling of online user behavior can be therefore formulated as a task of clickstream prediction. The problem can be specified in following way: Given a sequence of URLs visited by a user in a single session, predict the next URL, which will be visited by a user. Alternatively, under the presumption that the set of possible next URLs is known and finite, the task may be formulated as a learning to rank task, ranking URLs according to a probability that it will be the next URL in a session. The output of the model would then be a probability distribution over the set of all possible URLs.

Most of the current approaches, as described in section 3.2, are built on the presumption that clickstream can be modeled as a Markov process, which means that predictions can be made using limited historical information. The most common problem, that has to be addressed, is that there is a tradeoff between lower Markov order models (that look only a few clicks into history) and higher order models. Lower order models are beneficial because of high recall, but they usually have low precision. Higher order models, on the other hand, suffer from low recall but offer higher precision. This is caused by the complexity and sparsity of the state space of higher order models – it is quite uncommon to see the exact sequence of several clicks in both training and testing data.

One of the reasons of such large state space is that most approaches in literature treat URL as an atomic unit. The state space has dimensionality of the vocabulary (set

of possible URLs) to the power of the model's order. One approach for reducing this dimensionality is using models of lower orders, another is to use smaller vocabulary of URLs. This thesis follows the latter approach.

When the task is to understand user behavior, it is also desired to understand individual elements of this behavior. The URL – Uniform Resource Locator – is one of the most important building blocks of the World Wide Web. In the clickstream data, URL acts as an identifier of a particular content, most commonly a webpage, but it can refer also to a multimedia content such as a photo and a video. When modeling the user behavior, we would like to use unique representations of the content that user perceives – such that two identical webpages visited by two users would be represented identically. However, one of the challenges with clickstream data is that a single webpage may be represented by multiple URLs. This can happen for example in the case when a URL contains an identification of user's session or referrer of an ad that led the user to the page.

Another problem of URL representations is that it does not contain any semantic information. While it is legitimate to have two separate representations for e-shop webpages with t-shirts in two different colors, the difference from the perspective of user behavior is negligible. A better representation of the content perceived by a user, that captures semantic information, is therefore an important part of our task.

This thesis explores possible approaches of URL representations inspired by recent achievements in the field of computational linguistics. Word embeddings that capture syntactic and semantic meanings and allow significant reduction of the state space are now widely used in many NLP tasks. Our approach uses similar technique for unsupervised learning of URL embeddings, which are then used as underlying representation for a Markov model.

We present three methods for learning distributed URL representations, which differ in the amount of information that is extracted from clickstream and from URLs. The simplest approach use only behavioral information gathered from clickstream, while other approaches also leverage syntactic information extracted from the structure of URL. The produced URL representations are then hierarchically clustered and these clusters are used as representation for a Markov model. Our results from an evaluation on a large clickstream dataset show that a limited amount of behavioral information from clickstream is not enough for achieving even baseline accuracy level of a regular Markov model (without clustered representations). On the other hand, our second proposed model, based on tokenized URLs, was able to outperform this baseline and achieved modest improvement.

The content of this thesis is structured into six chapters. The description of the task and specification of goals is present in chapter 2. Overview of state of the art methods follows in chapter 3, which presents approaches for both clickstream modeling

and generation of word embeddings. Chapter 4 presents details about proposed approach, with details on implementation in chapter 5. Evaluation of all proposed methods is present in chapter 6. Finally, recapitulation and short elaboration about possible future research directions can be found in chapter 7.

Chapter 2

Thesis goals

The main goal of this thesis is to propose a model for prediction of the next click performed by a user, given clickstream data that represent user's previous behavior. This model has to leverage embeddings (also called distributed representations) of URLs and therefore act as an evaluation method for quality of these representations. URL representations have to be obtained only using the present clickstream data and should represent both semantic and linguistic relations between URLs.

The first step required for designing this predictive model is the state of the art analysis of the field of clickstream prediction. Clickstream prediction is one of the tasks of the field of Web mining, focused on mining and modeling clickstream logs of online user activity. The analysis of various methods used for clickstream prediction together with a summary of several relevant approaches from other disciplines of Web mining are presented in section 3.2.

The similarities between language and clickstream lead to the second part of the state of the art research – the analysis of approaches in word embeddings. Word embeddings, also called "distributed representations" is a term for mapping from words to n-dimensional continuous space, which capture characteristics of individual words and their relations. The short history and overview of methods used for unsupervised learning of these embeddings can be found in section 3.1. The prominent part of this section is focused on novel methods based on neural networks that got lots of research interest shortly after 2010.

As it was already stated in the beginning of this chapter, the main goal of this thesis is to propose an approach for clickstream prediction using distributed URL representations. This approach is built on top of previous research in the field of clickstream prediction and addresses main issues that are common for most of the current methods – namely the complexity of state space of Markov models, its poor generalization to unseen URLs and sessions; and the lack of short-term history and the large number of predictions of

methods based on Association rules [1]. Analysis of strengths and weaknesses of traditional methods can be found in section 3.2. Our proposed method which addresses these weaknesses using URL embeddings is presented in chapter 4.

The proposed model has to be trained on a clickstream dataset which was provided by Jumpshot Inc.[1] This task consists of data cleaning, preprocessing and training both parts of the model. First, a neural network for generation of URL embeddings is trained. Second, we train a Markov model for clickstream prediction, which works on top of clusters of URL embeddings obtained from the first model. The process of training these models is described in sections 4.4.6 and 4.5.

The quality of predictions made by the proposed model is then evaluated and compared with a selected baseline method. The results together with methodology that was used for evaluation can be found in chapter 6.

2.1 URL embeddings

The goals listed in previous paragraphs are necessary steps for the task of learning URL representations that would sufficiently capture semantics of the latent space of URLs and could be used for modeling the Web itself. Clickstream prediction is just one of the tasks that can be solved using these representations – high quality URL embeddings would be beneficial for myriad of tasks that are currently modeled using discrete URL representations. These include tasks like URL categorization, clustering, detection of malicious URLs, focused crawling and many others.

2.2 Clickstream prediction

The task of clickstream prediction is selected as a convenient method for evaluation of the quality of URL embeddings and also as a mean of evaluation of a generative model of user online behavior. While the exact prediction of the next click given only a limited history of clickstream data may be desired in some applications, it is a complicated task and even noisy or imperfect predictions can be highly beneficial to related tasks including recommender systems and personalization, which require models of the user behavior. Other applications include improvements of performance of online services with intelligent caching or pre-fetching of websites or multimedia. Models of user behavior can be also used for anomaly detection, filtering malicious traffic or Sybil detection in online communities. Predictive models can also serve as an input for website creators and help to improve user experience with webpages.

[1] https://www.jumpshot.com/

2.3 Assumptions and limitations

The main simplifying assumption that has been made is that we currently focus on prediction of clicks on a single domain. Disregarding this assumption would not only make the problem by a level of magnitude harder, but it would also complicate comparison with similar works, as most of them hold this assumption. The clickstream data provided by Jumpshot contains data for amazon.com, which is one of the largest e-commerce retailers in the world. The main part of the amazon.com domain is a catalogue of products, which is hierarchically structured. There is also a significant amount of other types of webpages that support Amazon's customer electronics business, serve for managing user profiles or seller's administration. This wide range of webpages together with the fact that amazon.com is one of top 20 most visited domains worldwide[2] makes it an ideal candidate.

Another limitation for this thesis is that both prediction task and embedding generation has to be done using clickstream data only. This may present a small disadvantage in comparison with other methods in literature (see 3.2.3). This limitation is in place mainly for scalability reasons, as Web crawling or other methods for obtaining webpage content are prohibitively expensive in large scale.

Also, as was already stated in the chapter 1, we presume that the set of URLs that can be visited by a user is finite and known in advance.

[2] http://www.alexa.com/topsites

Chapter 3

Related Work overview

This chapter is divided into two main parts, in the first one we will present an overview of the current state of the art of word embedding techniques, with more details about word2vec [2] and related methods based on neural networks, which had significant impact on the most recent research in the field and has been inspiration to many subsequent algorithms. The second part will present research in fields related to clickstream prediction, Web mining, online user behavior and studies of structure of the Web itself.

3.1 Word representations

In recent years, traditional word representations in natural language processing (NLP) are being replaced with word embeddings, which have gained substantial popularity. The word embedding is a mapping from large discrete space where each dimension represents a single word into continuous space with much lower dimensionality. The task of learning word embeddings is unsupervised (with some exceptions) and is based on the presumption that words that occur frequently in the same context are semantically similar. Produced dense vectors then capture both syntactical and semantical similarities between words. Moreover, these representations enable us to do simple vector arithmetic in the semantic space – as is shown in the well-known example *king* - *man* + *woman* = *queen* [2].

This section presents and compares several methods for learning word embeddings and also shortly discusses embeddings of sentences and paragraphs.

3.1.1 Bag-of-words representation

Most machine learning tasks require fixed-length vector representations of inputs. The simplest, however still broadly used representation of text, is a bag-of-words representation [3] (BOW). In bag-of-words representation, every word is represented as a one-hot encoded vector (e.g. vector of a length equal to a number of words in the dictionary which has

zeros on all indices except the index of the word) and document is represented as sum of one-hot vectors of corresponding words.

This approach is used mostly for its simplicity and sufficient results for many tasks. There are however several drawbacks, that led to more sophisticated methods. Bag-of-words representation does not capture ordering of words or its semantics. Another problem is high dimensionality of generated space, which is also almost in all cases very sparse.

3.1.2 Statistical language model

The basic building block in statistical language modeling is a language model – probability distribution over sequences of words. It can be represented as the conditional probability of the word at the position t given all the previous words. Formally, W denotes the sequence of words:

$$W = (w_1, w_2, \ldots, w_t) \tag{3.1}$$

$$P(W) = P(w_1, w_2, \ldots, w_t) = \prod_{i=1}^{t} P(w_i | w_1, \ldots, w_{i-1}) \tag{3.2}$$

This language model is sometimes called "the perfect language model", as it aims to capture the full information present in the data. The cost for this is the exponentially large number of parameters and therefore this approach is not applicable in practice. For real world applications, one has to employ simplifying assumptions, such as Markov property.

Language model with Markov approximation of $n-1$th order is called n-gram language model. Under this approximation, the probability of word at position t is only conditioned by last n words of the sequence. It can be defined as follows:

$$P(W) = \prod_{i=1}^{t} P(w_i | w_{i-n+1}, \ldots, w_{i-1}) \tag{3.3}$$

The main issue of these statistical language models is the curse of dimensionality – the number of possible word sequences is astronomical and it is common to see sequences different from training data in testing data. Statistical language model also does not take in account any similarities between individual words; all words are treated independently to others. This is addressed by models that operate on continuous variables instead of discrete ones like simple word counts.

3.1.3 Methods based on matrix factorization

One of the applications that require document or word representations that capture semantic relationships is information retrieval. This need is frequently addressed by methods based on Latent Semantic Analysis, which was introduced by Scott Deerwester et al. in 1986 [4].

The input for Latent Semantic Analysis (LSA) is a bag-of-words representation of multiple documents. This sparse document-word matrix is then decomposed using SVD, which results in dense representation of underlying (latent) characteristics of documents.

Representations produced by LSA are significantly better than simple one-hot vectors, however the ordering of words is still not taken in account and the structure of produced vector space is reported to be suboptimal [5]. Also, this method is computationally expensive and does not scale well to larger datasets. (Note that there recently exist methods for computing LSA incrementally [6])

While the LSA method operates on document-word matrix and results in representation of documents, other approaches like "Hyperspace Analogue to Language" [7] are based on word-word matrix. The representations produced by this method suffer from the fact that the co-occurrences matrix is unnormalized and frequent words therefore add noise to semantic meaning of the embeddings.

This is addressed by another notable approach, the GloVe model, which was proposed in 2014 by Pennington et al. [5]. Similarly to other factorization based methods, the GloVe approach is based on matrix decomposition – but in this case the word co-occurrence matrix is normalized and log-smoothed.

3.1.4 Methods based on neural networks

Methods that rely on artificial neural networks are based on the old idea of learning distributed representations that capture hidden semantics of symbolic data, which has been around since 1980s [8]. These representations, or so called "thought vectors" as popularized by one of the co-authors of mentioned paper, Geoffrey Hinton, are in fact a part of neural networks regardless of the task. They are inherent to problems from the field of NLP such as machine translation or speech recognition, but also to image recognition, segmentation, generation and others[1].

One of the pioneers of language models based on neural networks is Yoshua Bengio, who has proposed a "Neural Probabilistic Language Model" [9]. In this case, the model learns both word representations and probability function for sequences of words simultaneously. It therefore generalizes better than n-gram language models, mostly because of

[1] https://deeplearning4j.org/thoughtvectors

the fact that unseen sequences of words can still get high probability when they are composed of words similar to words in training sequences. While this method has achieved decent results and significantly improved at the time state of the art results, scaling to large datasets is still challenging.

All approaches listed so far were based on unsupervised learning, leveraging only unannotated corpora. The method proposed by Collobert and Weston in 2008 [10] can be classified as a semi-supervised multitask learning problem, which provides several advantages, however it may suffer from unavailability of large enough datasets. Their model is a deep convolutional network optionally combined with a classic fully connected network, which is trained on multiple NLP task jointly. These tasks may include part-of-speech tagging, chunking or named-entity recognition.

Later work from the same leading author - "Natural Language Processing (Almost) from Scratch" presented by Collobert et al. in 2011 [11] returns back to unsupervised learning, as it enables usage of datasets that are by magnitude larger. The main difference of this model and previous works based on language model analogy is that it is one of the first approaches that use both preceding and succeeding context of the target word.

Word2Vec model

While the method proposed by Bengio has achieved decent results and significantly improved at the time state of the art results, its main drawback is that it does not scale to larger datasets. This has been addressed by Tomas Mikolov et al. with model called word2vec [2].

Mikolov has proposed two similar architectures, that are essentially feedforward neural networks with one linear projection layer, and a nonlinear output layer. For the first architecture, *Continuous Bag-of-Words model* (cbow), the input consists of words sampled from neighborhood of the target word (four words that occurred before or after the target word), while the output is the target word itself. Words are represented as one-hot encoded vectors of fixed size. Second proposed architecture is called *Continuous Skip-gram model* and the only difference is that the target word is on the input of network, while words from neighborhood are predicted on the output.

Word2vec models are capable of learning representations from datasets that contain billions of words. This is particularly because the projection layer is linear and the only nonlinear, computationally costly functions are on output layer. Moreover, these expensive computations are usually further optimized using hierarchical softmax or candidate sampling [2].

In candidate sampling only a selected sample of target labels (words in case of word2vec) is evaluated in each step. Target positive labels are always evaluated, but from negative

labels (words that are *not* the target word in case of cbow model) only a small randomly drawn subset is evaluated.

Both hierarchical softmax and negative sampling presume that labels would not be assigned randomly, but according to the probability of its occurrence. For hierarchical softmax, most frequent labels should be in top levels of hierarchy, therefore Huffman's encoding is commonly used.

Apart from the improvement in scalability to large datasets, the other benefit of "word2vec" model is that produced representations capture both syntactic and semantic meanings of words. Thanks to the linearity of the model, the relationships between words can be expressed in algebraic operations in the embedded space, as was presented in the beginning of this chapter.

Most recent methods based on neural networks

A recent improvement of the word2vec model was proposed by Bojanowski et al. [12] in 2016. The approach was developed by Facebook Research (Tomas Mikolov has joined Facebook in 2014 and is one of the co-authors of this paper) and is known under the name "fastText". The main idea is to leverage subword information, which improves performance mostly for morphologically rich languages that contain many rarely occurring words. The method produces embeddings for individual character n-grams which are present in the words in training dictionary. The representation for a word is then a simple summation of representations of its n-grams. The proposed approach uses all n-grams with a length from 3 to 6 and also uses special representation for the most frequent words in the vocabulary, due to computational reasons.

3.1.5 Paragraph and document embeddings

Task that directly follows word embedding is distributed representation of larger pieces of text. While this task might seem unrelated to our problem, as there is no ambition to generate embeddings for multiple URLs, one of the possible approaches is to take URL itself as a combination of its tokens.

The first indication of a method that is based on composition of multiple embeddings was the fastText method, which combines representations of character n-grams. While simple addition of individual embeddings which is done in this case may be feasible, there also exist more sophisticated approaches.

One of the possible solutions was introduced by Quoc Le and Tomáš Mikolov in "Distributed Representations of Sentences and Documents" [13]. In this work, which is sometimes referred to as "paragraph2vec", a paragraph is represented using embeddings of

words that it contains, together with a special paragraph vector. This paragraph vector is obtained using second vocabulary of entire paragraphs. Vocabulary of the whole model is then a concatenation of vocabularies of words and paragraphs.

Even newer approach based on a recurrent neural network architecture was presented by Ryan Kiros et al. in an article called "Skip-Thought Vectors" [14]. This approach is inspired by recent achievements in neural machine translation and uses similar model based on recurrent neural network encoder-decoder with attention mechanism. The goal of the decoder is to reconstruct sentences in the surrounding context of an encoded sentence. Authors also propose a method for generalization to unseen words, which uses embeddings generated using second model like word2vec and maps them into embedding space of encoder-decoder model. However, this approach still does not generalize to words that were not seen by the second model.

Another interesting approaches based on more sophisticated neural network architectures include convolutional neural networks proposed by Kalchbrenner et al. [15] and tree-structured long short-term memory networks by Tai et al. [16].

3.1.6 Applications outside of the field of NLP

Our proposed method for URL embeddings is not the first use of originally NLP technique in another field. As presented by Asgari and Mofrad, semantic embeddings are used in biology for representation of sequences of proteins or genes [17]. There are also experiments with representation of rooted subgraphs from large graphs [18], or modeling of mobile applications based on their usage patterns [19]. Combination of clickstream data and word embeddings technique was also presented by Sagar Arora and Deepak Warrier in their work focused on e-commerce personalization [20]. Another similar approach can be also found in the following section – a click model based on distributed representation of user information need [21].

3.2 Clickstream prediction and Web mining

According to a survey presented by Stumme et al. [22], there are three main areas of research in the area of Web mining:

- *content mining*, which explores the actual content of webpages
- *structure mining*, leveraging hypertext structure of the Web
- *usage mining*, inspecting activity of online users

Our task of modeling user behavior and clickstream prediction falls into the usage mining category. However, this division which has been formalized in 1997 by Cooley et al. [23] is getting loose as there are often hybrid approaches that combine both content and usage data. The overview of methods in this section therefore does not respect this division strictly.

Contrary to the field of natural language processing, clickstream prediction or Web usage mining is a less coherent field. This is mainly due to scattered, highly specific tasks that do not allow direct comparison. Another obstacle is also the lack of standardized datasets or competitions. Most large clickstream datasets are proprietary and not available to public due to privacy or commercial concerns. Despite that, there are several significant contributions and inspirational works.

The task of clickstream prediction is traditionally solved using Markov models or Association rules. However, both of these approaches have its drawbacks which will be discussed in the following sections. There is an ongoing research in improving these methods, often inspired by other closely related fields of clickstream mining. We therefore also present a short overview of click models for Web search, which are models that predict the next click on search engine results page (SERP). Another related task that is discussed in this section is user categorization or clustering. This also usually requires modeling user behavior and is mostly intended for a better ad targeting, content recommendation or for the detection of intruders.

3.2.1 Clickstream prediction using Markov models

Markov models are commonly used in predictive modeling and are therefore well suited – and also frequently used – for our task. Markov property assumes that any future state of a stochastic process is dependent only on the present state. In our case, the next URL visited by user would only be dependent on the current URL. However, this assumption is too limiting, therefore we may use, similarly to language models discussed in 3.1.2, Markov approximations of higher orders.

The drawback of using higher order Markov approximations is that the sparsity of state space grows exponentially, and while the precision of these models increases, the recall decreases. Higher order Markov models are also more computationally demanding and require larger amount of training data.

One of the possible approaches for improving accuracy of higher order Markov models is an All-K$^{\text{th}}$-order Markov model, proposed in [24]. The method works such that we generate Markov models of all orders up to given K and combine them in prediction. If a model of order n fails to predict because corresponding n-gram was not present in the training data, we use model of order $n - 1$ and repeat this procedure if needed. The problem of this approach is the complexity of the state space. In the proposed work, this is addressed using three pruning methods: support pruning, confidence pruning and error pruning.

Different method for dealing with high complexity of state space was proposed by Dongshan and Shen [25]. Their "Hybrid-order tree-like Markov model" combines multiple Markov models with a tree-like structure, each of them having different order for improved recall. The tree structure serves as compressed version of user sessions.

3.2.2 Clickstream prediction using Association Rules

Association rule learning, which was introduced in 1993 by Agrawal et al. [1], is a common method used in recommender systems, personalization and similar applications. The main principle is that given a database of transactions that contain discrete items (in our case sessions that contain URLs), association rules are generated from item sets that frequently occur together in transaction. For example, if we see items A, B and C frequently together, we may introduce a rule that suggests C if A and B were already seen in the session.

Association rules have also gained its popularity in clickstream prediction. However, due to their nature, methods based on association rules frequently generate large number of predictions and suffer from inability to select the correct one [26].

One of the first applications was proposed by Mobasher et al. [27]. Their approach is built on Apriori algorithm [28], which is used for discovery of frequent itemsets, which are then stored in custom data structure.

Another possibility is to mine sequential association rules, which were studied by Yong et al. [29]. They showed that using association rules with sequential and temporal constrains may improve the prediction ability of the model.

3.2.3 Combined approaches in clickstream prediction

As was already mentioned, both Markov models and Association rules alone have several drawbacks. Many researchers therefore look for combinations of these methods that would pick advantages of both of them. Popular approaches include clustering webpages or sessions, and adding other information to the model like webpage content or temporal context.

One direction of research are ensemble methods, which combine results of multiple methods. For example, the framework proposed by Khalil et al. [26] combines low order Markov models and Association rules. Markov model is in this case used for prediction of candidates for possible next URL and Association rules are then employed for selection of the most probable one. An ensemble of four models – Markov models, both sequential and classical association rules and clustering is proposed by Kim et al. [30]. Models are ordered by its precision and prediction is selected from the most precise one that covers the presented sample.

Another direction is enhancing classical models with an additional information. Work presented by Vishwakarma et al. [31] improves All-K$^{\text{th}}$-order Markov model with clustered representations of webpages. These clusters are constructed using keywords obtained from the webpage content, and are used for disambiguation in case that Markov model outputs multiple predictions with similar probability.

Another similar approach is enhancement of Markov models with hidden context [32]. The context in this case may either be provided by a domain expert, or induced from training data. The process of context discovery is in the second case formulated as optimization problem which can be directly solved.

3.2.4 Click models for Web search

One of the most active – if not the most active – areas of Web related research is the problem of Web search. Most important components of the search engine are related to information retrieval and learning to rank fields, but inputs from user behavior has recently got its significance. Naturally, research in this area is driven by large search companies – in this section we more closely examine approaches initiated by Yahoo [33], Microsoft [34], [35] and Yandex [21].

There are some differences between regular browsing sessions and search sessions. Most importantly, the core of the search sessions is a user query. Secondly, the clicks that were not performed are equally important as these that were performed. The set of possible clicks is limited to these present on SERP and the session ends with the final outgoing click.

Traditionally, the probability of user clicking on a document presented on search engine results page (SERP) is modeled using probabilistic graphical models. In these models, user's behavior is a sequence of events, which some are observable (user clicked a link) and some may be hidden from the observer (user information need was satisfied). One of the first models that take in account previous clicks of the user was proposed by Dupret et al. [33]. The model is essentially a Bayesian network in which the probability of a click event on the document depends on all previous clicks on other results and on the rank of the document.

There are several other approaches that leverage user's behavior – those based on probabilistic graphical models can be found in a comprehensive survey by Chuklin et al. [36] – however for our task are more relevant newer approaches based on neural networks.

Methods based on neural networks

The work of Zhang et al. [35] is not directly dealing with click prediction on SERP, rather with click prediction on sponsored search offers. Their model uses both features that are static in time (information about the ad and about the user) and more interestingly sequential features about the past behavior of the user. However, these sequential features does not directly capture visited URLs or performed actions, but time dynamics (like the time since the last click event) of users behavior. Their proposed architecture is a recurrent neural network with time-dependent input, single hidden layer and click probability on output.

Particularly interesting paper that combines click models and distributed representations was presented on last year's Conference on World Wide Web. Borisov et al. have proposed "A Neural Click Model for Web Search" [21], a click model that combines distributed representation of user's information need and representation of the information acquired during the search session. The distributed representation of the query is derived from click patterns that has been observed for the same query in the past. The document representation is based on all click patterns on pages where the document was present, not necessarily generated by the same query. Initially, the representation is derived only from the user's query. When documents are presented to the user, the input of the model is enhanced with document representation and information about user's interest (whether the user clicked at the document or not). Loosely speaking, the representation changes its position in the latent space of click patterns according to user's actions. The architecture proposed by Borisov et al. is a recursive neural network with LSTM units, the model is therefore capable of modeling arbitrarily long search sessions.

3.2.5 Clickstream data for user modeling

Another task which relies on clickstream data is user clustering or classification. This usually does not directly involve clickstream prediction or modeling of individual clicks, it rather uses aggregated clicks or clickstream subsequences for modeling users.

One of the early applications was detecting outliers in user traffic on search pages proposed in [37]. Authors model individual user sessions using Markov chains with states that represent different user actions, such as a click on the search button, a click on a sponsored or organic link or a request for image results. Atypical sessions are then identified according to the probability with which it would be generated by this model.

Another application can be found in security, where user behavior can be used as a part of authentication. Similarly to the previous application in outlier detection, clickstream data is commonly preprocessed and user behavior is modeled using higher level events rather than URLs. More general approach for authentication on mobile devices was presented by Shi et al. [38]. User's behavior is modeled using Gaussian Mixture Models and clickstream data, which is reduced to domain names only is enhanced with other features like GPS position and phone call history.

User behavior data can also be used for detection of fake users in online communities. The task is in this case instance of supervised learning, as the classes of users are explicitly defined. Wang et al. [39] modeled the simplified clickstream (URLs were grouped into 8 event types) using Markov Chains, however the final classification was done on hand-picked features using SVM.

Unsupervised approach from the same leading author was presented in [40]. The raw clickstream data is also transformed and analysis is performed on 33 derived event types. This data is then hierarchically clustered using method based on similarity graphs – the distance (similarity) between two users is defined using count of common n-grams in clickstream sequences.

Clustering of both sessions and users is presented in an article by Volodymyr Melnykov published in 2014 [41]. Proposed approach is based on mixture of low order Markov models which operate on top of clustered URLs. However, the model itself does not perform this clustering – cluster assignments were part of the dataset.

Chapter 4

Approach

The approach for clickstream prediction proposed in this thesis is based on two key components: Markov models that capture short term history, which is essential for accurate prediction, and URL embeddings that capture global behavior of users and are crucial for proper generalization. Markov models are a standard technique used in the field of clickstream prediction. Its known weaknesses were frequently addressed by selecting more general representation of individual URLs. Our proposed method uses clustered URL embeddings as a general representation, which is inspired by recent achievements in linguistics.

In the first part of this chapter we describe general architecture of the proposed method. Then we present individual parts of this architecture and its variations. As there are multiple possible approaches for both clickstream modeling and URL embeddings, we provide more details about them and present short comparison of proposed methods.

4.1 Motivation

The main idea behind the proposed approach is that we can model clickstream data in a similar fashion to natural language. Traditionally, the basic unit of language is word, however as was shown in section 3.1.4, there are approaches that leverage even smaller units like morphemes or characters. Word has its inherent meaning and semantic content, and is usually used to represent some object or concept.

The basic unit in clickstream modeling would be, by this analogy, a single URL. URL by its nature represents an object that can be retrieved using the World Wide Web, but it can also represent user's action or intent - such as intent to read an article or buy a product.

Similarly to words, which are made up from morphemes, URLs have its distinct struc-

ture and are made up from smaller parts, tokens, that are frequently word-like to assure human readability. A single URL is therefore a basic unit of the clickstream, but it can be perceived also as a combination of smaller units. With this approach, one session in clickstream would correspond to a paragraph or a document, with URLs instead of sentences and tokens instead of words.

4.2 Architecture

This thesis compares several different methods, all of them are based on a similar framework. The main motivation of proposed framework is to reduce the complexity of state space, which is the biggest problem of higher order Markov models. This approach is similar to methods discussed in section 3.2.3, which combine Markov models with association rules or clustering by webpage content. However, our approach does not leverage any other information than this already present in the clickstream. Moreover, our approach combines both global information (which is common to methods based on Association rules) and local history of Markov models. Contrary to Association rules, the global information is encoded into continuous space of URL embeddings, which adds more flexibility and better generalization. The description of the proposed framework follows:

1. Clickstream data is preprocessed
2. Distributed URL representations are learned as an unsupervised task. These representations capture both semantic and linguistic similarity between URLs.
3. URLs are clustered according to their embeddings.
4. Markov model is constructed on top of these clusters.
5. Both seen and unseen URLs in prediction time are embedded and their embeddings are assigned to clusters.

This framework permits different approaches to embedding generation and subsequent construction of Markov model. Following sections describe these approaches in greater details.

4.3 Preprocessing

Raw clickstream data is naturally very noisy and require heavy preprocessing. This holds for the data obtained from Jumpshot Inc., which are described more thoroughly in section 6.1. The number one problem that should be addressed by preprocessing is uniqueness of

URLs. From the definition, URL should provide a way of resource identification, however single resource – webpage – can be identified by multiple URLs. That is because of query parameters and fragment part, which are commonly used for user or session identification. On the other hand, query parameters do often contain an important information for the webpage identification, like product id. Our problem is fortunately limited to a single domain, therefore we can identify these "noisy" parameters manually and filter them out.

The distribution of URLs present in the dataset have (even after the first preprocessing step) a very heavy tail – there is a great number of URLs that have been seen only few times in training data. This is troublesome, but expected, especially on domain like amazon.com. Given the fact that every product sold on Amazon has its unique URL, and given the number of products listed there is greater than number of US citizens[1], this would hold with basically any amount of clickstream data. While some of our proposed approaches address the issue of unseen data, we limit this amount in preprocessing to considerable amount for better comparison with baseline methods and mainly for practical computational reasons. We discard all URLs that were seen less than 5 times in the dataset.

Another preprocessing step removes duplicated clicks, which are caused by users who refresh the page or navigate within one page. These duplicated clicks would bring noise to the prediction task and also complicate convergence of the model for embeddings generation. We also remove sessions that contain less than 10 clicks.

The first part of preprocessing, which is common to all approaches can be summarized in the following way:

1. URLs are stripped of referral parameters
2. Least common URLs are removed
3. Duplicated clicks, e.g. situations when next URL is equal to present URL are removed
4. Sessions that contain less than 10 clicks are removed

The input for the model for embeddings generation requires further preprocessing steps, which are slightly different for different architectures, which are presented in section 4.4. Example describing these steps for model EMB2.1 on a real URL is provided in the figure 4.1. First, URLs have to be tokenized, either using tokens extracted from training data, or using all possible character n-grams in the case of n-gram method.

[1] https://export-x.com/2015/12/11/how-many-products-does-amazon-sell-2015/

Next, tokenized URLs are represented as sparse bag-of-words vectors. We have also experimented with an even simpler representation, when the BOW vector is binarized – obtained results were comparable.

Last two steps in the preprocessing pipeline are present mostly from implementation reasons, which are explained in section 5.2. Sparse BOW vectors are padded to fixed length, which may, in rare occurrences, when the number of tokens in single URL is large, lead to loss of some information. The size of fixed length representation was selected such that the fraction of trimmed URLs is less than 1 %. The final step is mini-batching of input for more efficient training (the description of mini-batch training can be found in section 4.4.6).

Preprocessing steps required for the model for embeddings generation therefore are:

1. URLs are tokenized either using extracted vocabulary of tokens or using n-gram method
2. These tokenized URLs are encoded using BOW method
3. For computational reasons, BOW vectors are padded to fixed length
4. Finally, batches of BOW vectors are created for mini-batch learning

4.4 URL Embeddings

In this work, we propose two architectures for generation of URL embeddings. Both architectures are built on top of the skip-gram architecture presented by Mikolov et al. in [2]. This architecture is known for producing high quality embeddings for both frequent and rare words. Also, the model scales well to datasets with billions of words and can be relatively easily implemented in a scalable fashion.

Our proposed architectures differ in the granularity of basic units of clickstream – the first architecture (EMB1) works on URL level, the second one operates on sub-URL level. We propose two additional implementations of the second architecture: EMB2.1, which represent URL as bag of tokens, and EMB2.2, which represent URL as a bag of character n-grams. Both variations of second architecture share the same model implementation and the only difference is in the preprocessing step.

In comparison with existing approaches in NLP, we can take the EMB1 model as the skip-gram model in the original word2vec proposal [2]. The architecture underlying EMB2.2 model can be aligned with the fastText approach [12], while the EMB2.1, even though it uses very same neural network, can be conceptually compared with document embeddings presented in [13].

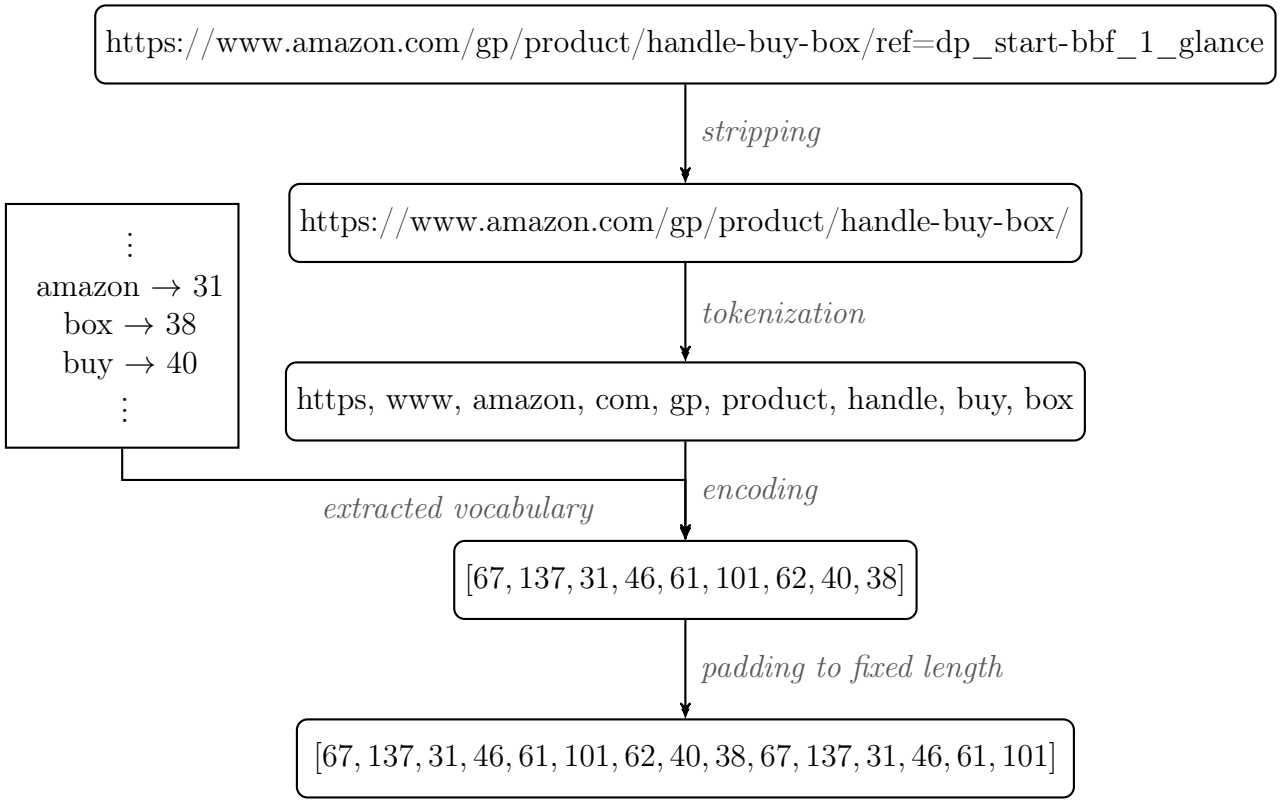

Figure 4.1: Example of preprocessing steps for the EMB2.1 model. A raw URL is stripped of referral parameters, tokenized and encoded with vocabulary of tokens that has been extracted from training data. This sparse bag-of-words representation is then padded to fixed length for efficient training.

The drawback of EMB1 architecture is inability to generalize to unseen URLs, as the vocabulary has to be fixed prior training. Also, this architecture does not handle well URLs that appear rarely in the clickstream. Even with this limitation this architecture allows generalization to unseen sessions, which improves baseline results.

Architectures EMB2 have the ability to generalize to unseen URL under the limitation that it consists of tokens or n-grams that were seen in training. These architectures can also handle rarely appearing URLs, as long as they are composed of more frequently appearing tokens. Training with tokens in model EMB2.1 in general yields better precision, is less computationally demanding and requires smaller training data in comparison with EMB2.2. The n-gram method on the other hand generalizes better and should be more robust. However, the representation is noisier and may require longer training and greater number of training examples.

4.4.1 Neural network design

Both models are feedforward neural networks with one linear projection layer and one non-linear output layer. These networks are also often called shallow networks as they usually consist of few layers with large numbers of neurons, opposed to deep neural networks with many layers of smaller number of neurons.

The input of the network is always the present URL – represented as a simple one-hot encoded vector in case of URL-level model EMB1 or as a bag-of-words vector in case of the second architecture EMB2. The output layer is then probability distribution over candidate URLs – where the selected URL from the session context should have the highest probability.

4.4.2 Context

The main purpose of the model is to generate embeddings that capture semantic relationships between URLs, and produce vectors that are close to each other if the URL has similar interpretation in user session. This is motivated by the linguistic observation that word's semantics is defined by its context.

In NLP, one of the first mentions of the importance of larger context is given by Collobert et al. [11], where authors propose context of size 5. Even larger context is used in the word2vec model, where target words are selected from context of size 10, which means that 20 surrounding words in a sentence are sampled. Extended context is said to improve the quality of vectors; however, it comes at cost of increased computational complexity [2]. It is also worth to note that the probability with which the context word is sampled in word2vec model decays with the distance from the target word.

u_{t-3} `http://amazon.com/gp/site-directory/134-4897415-3048740`
u_{t-2} `http://amazon.com/Mens-Fashion/b/?ie=UTF8&node=7147441011`
u_{t-1} `http://amazon.com/b/?node=16613798011`
u_t `http://amazon.com/Original-Penguin-Stripe-Button/dp/B01N6O61J3/`
u_{t+1} `http://amazon.com/Original-Penguin-Collar/dp/B01N7P567J/`
u_{t+2} `http://amazon.com/gp/product/handle-buy-box/`
u_{t+3} `http://amazon.com/gp/huc/view.html?ie=UTF8`

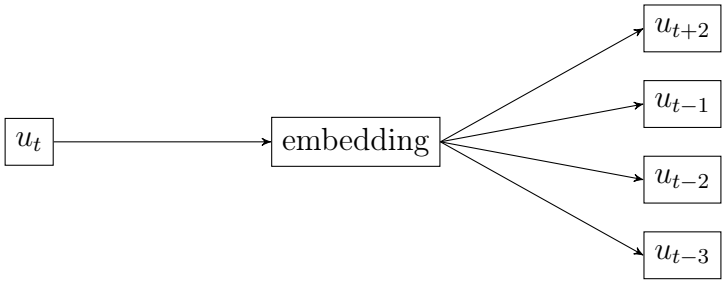

Figure 4.2: Sample clickstream data and an example of skip-gram architecture. URL u_t is on the input of the neural network, and its embedding is used for the task of context prediction. In this example, URLs u_{t+2}, u_{t-1}, u_{t-2} and u_{t-3} were randomly sampled from the context of size 3, with number of skips (sampled URLs from context) equal to 4.

The motivation in our application is analogical to NLP – however given the fact that user sessions are in general noisier than sentences; the fact that our datasets are by magnitude smaller than those used in mentioned works; and due to limited computational resources, we have used modest contexts of sizes 1 and 3. We have also experimented with one-sided context, which uses only one context word w_{t+1} for each target.

4.4.3 URL-level model (EMB1)

The first proposed approach is built on a presumption that URL is an atomic unit of clickstream. The input vocabulary for the neural network is constructed from training data and consists of all unique URLs.

The input of the network is one-hot encoded URL u_t and the output is a randomly selected URL from the context – u_{t+r}, where, in case of context of size 3, $r \in \{-3, -2, -1, 1, 2, 3\}$. Weights in the network are then trained to maximize likelihood $P(u_{t+r}|u_t)$. (Note that the actual optimization criterion is only an approximation of this due to performance reasons, which is described in 4.4.6).

The network has one hidden layer, which is a linear projection using weight matrix W_0, that produces embedded representations e. Matrix W_0 has a dimensionality $|V| \times d$, which yields embeddings of dimension d. The weights and biases of an output layer are denoted

by W_1 and b_1 respectively. It is required to have a proper probability distribution on the output layer, which is accomplished by the softmax function σ. The softmax function scales an input vector of arbitrary real values x to a vector of values in range $[0, 1]$ that sums to one. In the following equation, a softmax value of an element of original vector x on index j is computed as $\sigma(x)_j$.

$$e_t = u_t W_0 \tag{4.1}$$

$$P(u_{t+r} = j|u_t) = \sigma_j(e_t W_1 + b_1) \tag{4.2}$$

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_{k=0}^{|V|} e^{x_k}} \tag{4.3}$$

This model leverages only behavioral information which is present in the clickstream and does not take in account linguistic similarities between individual URLs. This also implies that the model will not be able to deal with URLs that have not been present in the training dataset.

4.4.4 Token-level model (EMB2)

With the paradigm that URLs itself can be decomposed to smaller parts, tokens, the second model leverages both behavioral information from user sessions and linguistic similarities that are present in the URL. This is desired especially in the case when the clickstream is limited to a single domain, which can be assumed to use similar tokens for construction of URLs.

There are two different methods how tokens can be obtained. First one relies on domain knowledge, when we know the structure of the URL and we can parse it into different chunks, usually breaking the full URL on non-alphanumeric characters. However non-alphanumeric characters often carry important information, so it may be favorable to not dispose them. We also do not have a natural upper bound on the number of possible tokens, therefore we have to employ a pruning mechanism in case when URLs contains a large number of generated ids or hashes.

The second option is to use all possible n-grams as tokens; this approach reduces the problem to character-level representation with space of dimensionality equal to the length of alphabet when $n = 1$, and quickly increases with larger n. Our experiments were performed with $n = 3$, as trigrams are the largest n-grams that yield space with smaller dimensionality than pruned tokens.

The embedding of the URL is in this case combination of multiple embeddings of tokens, which the URL consists of. There are several options how these token embeddings

can be combined, most common approaches are mean and weighted sum. All models in presented experiments use weighted sum as a combination function.

Given the fixed length sparse bag-of-words representation u obtained from the preprocessing step, the frequency of tokens is encoded by repeating them – all weights in the weighted sum are equal to one.

In the following equations, the input URL is denoted u_t and consists of p tokens $w_{t,i}$ (either obtained using tokenization in case of EMB2.1 or n-gram tokenization in case of EMB2.2). The probability of the target URL randomly selected from context, u_{t+r}, is computed in the following way:

$$u_t = (w_{t,1}, \ldots, w_{t,p}) \tag{4.4}$$

$$e_t = \sum_{i=0}^{p} w_{t,i} W_0 \tag{4.5}$$

$$P(u_{t+r} = (w_{t+r,1}, \ldots, w_{t+r,p}) | u_t) = \sum_{i=0}^{p} \sigma_i(e_t W_1 + b_1) \tag{4.6}$$

4.4.5 Embeddings

The output layer of neural network gives us probabilities of other URLs being in context of input URL. Embeddings, which are the main product of this model are obtained from the hidden layer. Dimensionality of embeddings is equal to the number of neurons in the hidden layer d and can be therefore adjusted given the size of training data and computational resources. Experiments in this thesis were performed with $d = 512$.

4.4.6 Training

Neural networks are commonly trained using backpropagation, which is a method based on the minimization of the error of the network. The first part of the backpropagation algorithm is the forward calculation, when the input is propagated through the whole network and values on the output layer are compared to true labels. This comparison yields error value, which is used for the computation of the gradient of the loss function. The gradient is then used for updating weights in the network during the backward pass. One of the important properties that is required by the backpropagation is that all functions in neural network has to be differentiable, se we are able to compute its gradient.

The general principle of the gradient descent is simple, however it has its limitations mostly due to the problem of the high dimensionality of the loss function and also due to

the fact that it has many local minima. We therefore have to employ several tricks that are required for a proper convergence. These are described in following sections.

Prior to training, weights of the neural network have to be initialized. The most commonly used way is to use values drawn from a normal or a uniform distribution, depending on the chosen activation function. All our models were initialized using a truncated normal distribution, which is beneficial because all weights are guaranteed to be close to zero. Truncated normal distribution discards all values that are further than 2 standard deviations from mean, and re-generates them. All biases of the hidden layer are initialized to zeroes, which is also a common strategy.

While there are several approaches how to select stopping criteria for neural network training, most of them are valid mostly for supervised tasks. We therefore stop the training when the loss function gets saturated and does not change substantially.

Mini batch training

There are three common strategies for estimation of the true gradient using gradients of training samples: online learning, batch learning and mini-batch learning [42]. In online learning, weights of the neural network are updated after a calculation of a gradient of each training sample. On the other hand, batch learning performs the update of weights using an average gradient of all samples in the training set. The mini-batch method is a compromise – weights are updated after a specific number of samples.

The advantage of mini-batch learning is that a large number of input samples can be processed together, which improves performance especially on devices that allow large parallelization like GPUs. Also, the quality of gradient estimation is higher than in case of online learning, as it averages several training examples. The disadvantage of mini-batch learning is that the variance in examples in a mini batch may lead to lower final accuracy.

Loss function

The training objective for both EMB1 and EMB2 models is to predict the other URL in the context of the session. The standard approach is to normalize outputs of the neural network using softmax function σ and then maximize log-likelihood (for performance and numerical stability reasons). The training objective J_{ML} for model EMB1 is in this case computed as a logarithm of conditional probability of the other URL in context, u_{t+r}, given the input URL u_t, which is described in equation 4.2. The x denotes output of the hidden layer.

$$J_{ML} = \log P(u_{t+r}|u_t) = x_{t+r} - \log \sum_{k=0}^{|V|} e^{x_k} \tag{4.7}$$

However, the computation of full softmax is an expensive operation, which may be prohibitive when the size of vocabulary of possible URLs or tokens is large. The most common method for solving this issue is to approximate the full softmax using a sampling method. Mikolov et al. [2] used negative sampling, we have used similar method called Noise contrastive estimation [43] for our model.

Noise contrastive estimation approximates the task of maximizing log-likelihood with the task of distinguishing samples drawn from the target distribution from samples generated by noise distribution. In our case the samples are tokens from encoded URLs in case of a token-level model or a single sample from vocabulary of possible URLs in case of URL-level model.

Optimizer

The most straightforward approach for computing weight (parameter) updates in back-propagation is using the gradient method, commonly using mini-batch stochastic gradient algorithm (SGD). This means that the update step is computed for every mini-batch of training examples. As a result, the parameters fluctuate around current local minima, which enables the possibility of finding better local minima, but also makes the convergence more complicated. Moreover, SGD is sensitive to the selection of learning rate, which adds another hyperparameter to the model.

After initial experiments with SGD, that proved difficulties with convergence of this method, we have replaced it with Adagrad, which was proposed by Duchi et al. in 2011 [44].

Adagrad is an extension of SGD, that dynamically changes gradient steps based on the data that have been observed in previous iterations. In comparison with SGD, which is independent on the structure of gradient landscape, the Adagrad algorithm changes gradient steps such that it leverages predictive, but rarely seen features instead of those that are seen frequently. This information is used for adaptive modification of the proximal function (which approximates the loss function, which is not fully explored) which results in more robust results and lower sensitivity to selection of learning rate. If not stated otherwise, all models in this thesis were trained with a learning rate 0.01. More details about Adagrad can be found in the original paper or in introductory article by Duchi and Singer[2].

[2]http://www-cs.stanford.edu/~ppasupat/a9online/uploads/proximal_notes.pdf

4.5 Markov models

Markov models, also called Markov chains, are used for modeling stochastic processes that can be simplified such that they satisfy the Markov property. This property says that the probability distribution of the future state depends only on the present state – and is independent on states preceding it. In our case this would mean that the next URL in user's clickstream depends only on a current URL. This dependency can be extended using proper encoding of current state. If we select the representation of current state as n last visited URLs, we get Markov model in which next URL depends on last n URLs. Formally, given a session W of length t, a Markov model of order $n-1$, and a set of all URLs U, the probability of next URL in this session being u is given by:

$$W = (u_1, u_2, \ldots, u_t) \tag{4.8}$$

$$P(u_{t+1} = u|W) = \max_{u \in U} P(u_{t+1} = u|u_{t-n+1}, \ldots, u_t) \tag{4.9}$$

This approach is basically about computing probabilities for all webpages and for all sequences of given length and then selecting the webpage with highest probability. This model has several drawbacks, which are discussed in 3.2.1.

4.5.1 All-K$^{\text{th}}$-order model

One of the most straightforward solutions that improves recall of higher order Markov models is so called All-K$^{\text{th}}$-order model. In this approach, we generate models of orders $1 \ldots k$ and combine them for prediction. This is similar technique as n-gram smoothing in language modeling. The problem can be expressed as:

$$P(u_{t+1} = u|W) = \max_{u \in U} \big(\sum_{i=0}^{k} \lambda_i P_i(u_{t+1} = u|u_{t-i+1}, \ldots, u_t) \big) \tag{4.10}$$

$$s.t. \sum_{i=0}^{k} \lambda_i = 1 \tag{4.11}$$

This has introduced a new parameter λ, which can be set according to different strategies. In NLP, this is usually addressed either by setting all lambdas equal to $1/k$, or by learning optimal lambdas from training data using expectation maximization algorithm. This adds another layer of complexity and while the motivation in natural language is clear, it may not be desired in clickstream prediction.

Our approach is rather simple and follows the All-K$^{\text{th}}$-order model as proposed by

Deshpande et al. [24]. We set all lambdas to zero, except the first one for which we are able to make decision – that effectively means we are going from highest order model to lowest one and stop when we are able to make prediction. Formally:

$$\lambda_i = \begin{cases} 1 & \text{if } i = \text{argmax}_i\, i(P_i(u_{t+1} = u | u_{t-i+1}, \ldots, u_t) > 0) \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

4.5.2 Markov model on clustered embeddings

Even though the All-K$^{\text{th}}$-order approach partially solves the problem of state-space complexity, it does not improve predictions for previously unseen URLs and does not take in account any semantic information about URLs. We therefore propose a model, which is not based on discrete URL representations. For our purpose, a URL is in the Markov model represented with its cluster. This cluster is obtained from its embedding, which was generated in the previous step. Details on clustering methods are provided in section 4.6. It is worth to note that cluster representations figure only in the user's session, predictions made by our model are plain URLs. The problem for a Markov model of order $n-1$ can be reformulated as follows:

$$c_i = cluster(u_i) \tag{4.13}$$

$$W = (c_1, c_2, \ldots, c_t) \tag{4.14}$$

$$P(u_{t+1} = u | W) = \max_{u \in U} P(u_{t+1} = u | c_{t-n+1}, \ldots, c_t) \tag{4.15}$$

4.6 Clustering

The purpose of clustering of URL representations is to reduce dimensionality of the state space and to allow generalization to unseen URLs. Our goal is to cluster together URLs that has identical semantical meaning – are interchangeable in user's sessions and usually are seen preceding same URLs. These can be either a different URL representing identical webpage (e.g. when URLs differ only in session identifier as discussed in section 1) or similar webpages that usually lead to common successor (e.g. pages describing two different shipping methods leading to order confirmation).

Granularity of clustering is one of the most important hyperparameters of the model. It can be seen that if number of clusters $|C|$ is equal to the number of URLs $|U|$, the clustered model performs equally to the regular Markov model. Also, when $|C| \to 1$, the performance would be similar to the performance of Markov model of order 0.

Our representations of URLs are embedded in high dimensional space, which complicates clustering due to the curse of dimensionality. One of the challenges is that with the increasing dimensionality, the Euclidean distance between points is less informative, as the difference in the distance between different pairs of points is vanishing.

One of the commonly used solutions for this problem is using cosine similarity instead of Euclidean distance. An intuitive explanation of cosine similarity may be that we first project the points to unit hypersphere, and then measure the angle between them. Formally, given input vectors A and B:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} \tag{4.16}$$

While there are several clustering methods that allow parametrization of the number of clusters, the natural choice for this application is hierarchical clustering. We use agglomerative clustering, which begins with all samples assigned to its own cluster and merges them subsequently. In each step, the algorithm merges two clusters with minimal average distance between samples. The clusters are therefore built in a "bottom up" fashion.

Chapter 5

Implementation

Most parts of this thesis were implemented as a modular Python framework, which incorporates methods for preprocessing, model training and evaluation. Exploratory analysis and prototyping was done using Jupyter notebooks [45], which provide enough interaction and flexibility which is important for this task.

Python was selected for its expressivity and because of the vital scientific ecosystem, which includes number of libraries that provide out-of-the-box implementations of common algorithms. Python is nowadays also one of the main languages used for training neural networks, as its the main API for Tensorflow framework [46] and many others. In addition, Python packages NumPy [47] and SciPy [48] provide fundamental matrix types and include convenient implementations of many numerical routines. These packages are built on low level linear algebra libraries like BLAS, which guarantees fast and efficient computations with larger amounts of data.

5.1 Data acquisition and preprocessing

The very first part of data acquisition, selection and sampling of the data for use in this thesis was done in Apache Hadoop ecosystem using Hive[1] and Spark [49]. This initial step yielded the dataset, which is described in section 6.1. Hive provides SQL-like interface to large datasets stored on HDFS, which is convenient, as user does not have to write low level map-reduce tasks. Hive is used for the initial filtering of the Jumpshot's internal dataset, namely the selection of date frame, country and particular domain. This raw clickstream dataset is then processed using Spark, which is a framework for distributed computing, data manipulation and machine learning. The dataset from the previous step is reindexed and divided into user sessions. The resulting dataset is small enough to fit on a single machine and is therefore saved in a csv format for an easy manipulation.

[1] https://hive.apache.org/

The next step is data filtering and preprocessing, as discussed in section 4.3. The implementation of the preprocessing pipeline respects the scikit-learn [50] philosophy, and consists of several custom transformers together with some that were already implemented in other libraries. These transformers operate on a dataset which is streamed from the disk, therefore are memory efficient and scalable. The modular nature of the preprocessing pipeline also allows fast prototyping and enables an easy comparison of different preprocessing methods. Apart from the scikit-learn library, which was used as a framework holding all modules together, we have also used the Gensim library [6] for dictionary operations and pandas [51] as a convenient method for work with tabular data and its serialization.

5.2 URL Embeddings

The neural network, which is used for learning URL embeddings was implemented in the Tensorflow [46] framework. Tensorflow is one of the most popular open source libraries for training neural networks and similar models based on computational graphs. Tensorflow is developed by Google Brain team and its first version was released in November 2015. The framework quickly became popular and is now used by many top technological companies and has an active community.

Despite its wide popularity, Tensorflow has its weaknesses, mainly unstable API, which is a consequence of the fact that there was not a stable release until February 2017. The second weakness is that Tensorflow operates on a quite low level of abstraction and while this is beneficial for great flexibility and allows us to change and implement specific details of the architecture, its effect is a steeper learning curve and slower prototyping. These weaknesses are addressed by Keras [52], which is a popular high-level library build on top of Tensorflow (or Theano [53], which is another frequently used library for tensor computations) backend. While we have initially experimented with Keras implementation, we have quickly discovered its limitations – mostly because of need of more advanced layers and loss functions.

Tensorflow is a quite novel project and while it contains number of ready-made components, there are still parts that are not fully completed. This is unfortunately true for some methods and components that are required by our proposed model, we had therefore employ several workarounds to overcome these difficulties. One of the main difficulties is that Tensorflow's sparse embedding, which is required for mapping from bag-of-words representation to vector representation, does require fixed number of tokens. This can be fortunately addressed in a way when we set the fixed number of tokens high enough and pad these representations by repeating the individual tokens. This way we can also pre-

serve the observed frequencies of tokens, however this approach adds unnecessary noise to the model. Similar complications arise with the noise contrastive estimation loss function, which does not allow variable number of target classes.

5.2.1 Training on GPU

One of the benefits of Tensorflow library is ability to change between CPU and GPU implementations seamlessly. While prototyping work can be done locally on CPU, training on larger datasets was done on GPUs which were provided by either Metacentrum or Jumpshot. The amount of data combined with the size of the network is prohibitive for training on CPUs, we have therefore utilized graphics cards with CUDA support, namely nVidia Tesla K20Xm 6GB and nVidia Tesla K80 24GB. Still, even with this computational power, training of neural networks for URL embeddings took 5-20 hours, depending on number of steps.

5.3 Markov models

The implementation of Markov models of various orders is quite straightforward and was done using SciPy library [48]. The SciPy library provides fast and memory efficient methods for work with large sparse matrices, which are essential for this method.

Our approach for clickstream modeling is built on top of clustered URL representations. We have experimented with several different clustering methods, and used implementations that are present in the scikit-learn library.

Chapter 6

Evaluation

This chapter is divided into three parts. The first part consists of a description of the dataset which was used for training and evaluation of proposed models. The second part contains definitions needed for a proper evaluation of the quality of models. The third part presents actual experiments that were performed as a practical part of this thesis.

Proposed models are compared with selected baseline methods – regular Markov models of varying orders. The baseline model acts as an upper bound in terms of training accuracy, as its inner working is based on memorizing the training data. The goal of our approach is to explore the possibility whether this baseline model can be outperformed in terms of generalization, e.g. whether our proposed model can achieve higher accuracy on the test set.

6.1 Dataset

The dataset provided by Jumpshot Inc. is a random sample of clickstream data acquired from their panelists. Clicks that are included in this sample are limited to a single domain – amazon.com – and are restricted to IP addresses from the US only. The sample was collected in January 2016 and contains sessions of both PC and mobile users. Due to the private nature of data, the dataset unfortunately can not be made public. The clickstream data contains three columns: session identifier, order of a click in a session and URL.

The original dataset contains around 60M clicks, which were reduced to slightly more than 12M after preprocessing steps as described in the section 4.3. This dataset is referred to as DATA-12M and includes about 600k unique URLs in about 1M sessions. The histogram of session lengths is displayed in the figure 6.1.

The second dataset was created for the purpose of in-memory training of Markov models. It is a subset of the full dataset and consists of 500k clicks. This dataset is referred to as DATA-500K.

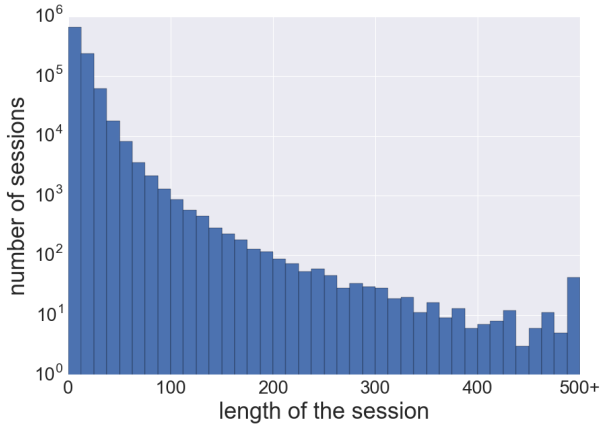

Figure 6.1: Histogram of session lengths in the raw dataset. Y-axis is in logarithmic scale, sessions longer than 500 clicks were merged into the last bin.

6.2 Methodology

6.2.1 Accuracy

The quality of clickstream prediction is evaluated on related tasks – exact prediction TOP1, where is measured whether the model is able to predict the exact next URL; and TOP5 prediction, where we measure whether any of top 5 guesses of the model is the true next URL. The performance of models is measured using accuracy score, which is defined as the fraction of correct predictions in the set of n samples. We denote the true value (the actual next URL) y_i and predicted values $\hat{y}_{i,j}$, where i stands for index of sample and j is index of prediction, when ordered from the most probable to least probable. The accuracy of the exact prediction and the TOP5 prediction is then defined as follows:

$$\text{accuracy}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_{i,1} = y_i) \tag{6.1}$$

$$\text{top5 accuracy}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} \max_{j \in 1 \ldots 5} (\hat{y}_{i,j} = y_i) \tag{6.2}$$

6.2.2 Cross-validation

Estimating model performance on real-world data is a sensitive task. The common way is to use two sets of data - training set and testing set. It is also sometimes beneficial to use a third, validation, set for hyperparameters selection. As the significance of hyperparameters in our approach is low and we do not employ methods for hyperparameter tuning like gridsearch, conducted experiments were done with only train and test sets of data. Under the assumption that samples in our dataset are drawn from the real-world

distribution i.i.d. (independent and identically distributed), it can be said that the real model error corresponds to the error measured on the test set.

For a more accurate estimation of the test set error, it is recommended to compute cross-validated metrics. In this thesis, experiments were performed using k-fold cross-validation, which works as follows: First, dataset is split into k folds. Next, k iterations of training and testing procedures are performed – the model is evaluated with different fold every time and remaining $k-1$ folds are used for training.

If not stated otherwise, cross-validated results are presented with 95% confidence intervals.

6.2.3 t-SNE

For visualization, embeddings were reduced to 2-dimensional space using t-distributed stochastic neighborhood embedding (t-SNE) [54]. t-SNE is a nonlinear dimensionality reduction technique that preserves locality. That means that points that are nearby in original space are close to each other in projected space. This makes it popular choice for visualizations of high-dimensional data, however interpretation of projected data may be sometimes misleading [55]. t-SNE is a nonlinear technique and performs different transformations in different regions, therefore projections that look similar in reduced space does not have to be as similar in original space.

6.3 Generalization of higher order Markov models

6.3.1 Simple Markov model

The baseline model for our task of clickstream prediction is a Markov model with one-hot representations of URLs. The first experiment compares Markov models of different orders, as defined in section 4.5.

All Markov models were evaluated using the DATA-500K dataset, train/test accuracy is computed using k-fold cross-validation with $k = 5$. Due to memory requirements of models of larger orders, results were computed for orders up to 4.

Results are presented in the figure 6.2 with accuracy plotted against y axis and Markov order on x axis. We have observed the same behavior of Markov models as reported in other literature (see section 3.2.1), most notably poor generalization of higher order Markov models and expected overfitting on training set.

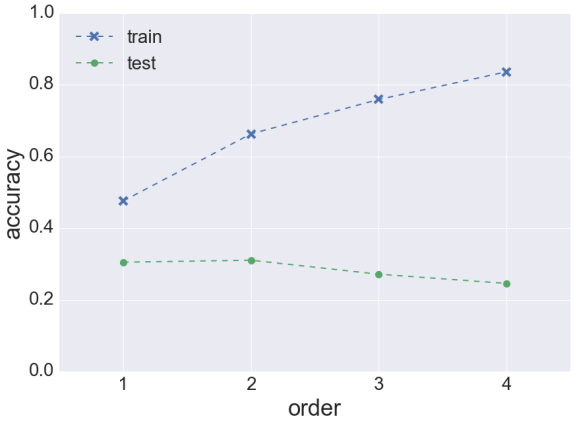

Figure 6.2: Accuracy of Markov models of increasing orders.

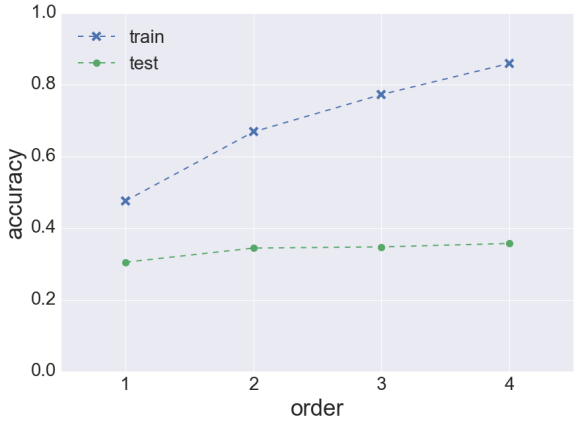

Figure 6.3: Accuracy of All-K$^{\text{th}}$-order Markov models of increasing orders.

train set				
Markov order	1	2	3	4
Simple	0.4761 ± 0.0061	0.6631 ± 0.0031	0.7591 ± 0.0012	0.8363 ± 0.0023
All-K$^{\text{th}}$	0.4761 ± 0.0061	0.6687 ± 0.0026	0.7729 ± 0.0020	0.8597 ± 0.0012
test set				
Simple	0.3053 ± 0.0431	$\mathbf{0.3107 \pm 0.0482}$	0.2718 ± 0.0502	0.2461 ± 0.0574
All-K$^{\text{th}}$	0.3053 ± 0.0431	0.3447 ± 0.0467	0.3479 ± 0.0471	$\mathbf{0.3576 \pm 0.0537}$

Table 6.1: Accuracy of Markov models of increasing orders.

6.3.2 All-Kth-order Markov model

First improvement of the Markov model is so called All-Kth-order model, which combines predictions of Markov models of orders up to k, as described in section 4.5.1. The setup was identical as in the previous experiment.

Results are presented in the figure 6.3 and in the table 6.1. The accuracy of higher order models have improved, but it comes with performance and memory costs – the model have to train and store probabilities for all four sub-models.

6.4 Markov models on clustered embeddings

The predictive quality of URL embeddings was evaluated on both tasks – TOP1 and TOP5 prediction, with embeddings clustered into 27354 clusters (unless stated otherwise) using hierarchical clustering with cosine similarity as a linkage criteria. All experiments were performed with Markov model of the second order.

The embeddings were trained on the dataset DATA-12M, Markov models were then evaluated on dataset DATA-500K. Results for all models are presented in the table 6.2.

6.4.1 EMB1

First embeddings were produced using the EMB1 model, which is built on top of one-hot representations of URLs. The context window was set to 3, dimensionality of embeddings to 512. Training was done using Adagrad optimizer with learning rate 0.001 on batches of size 256.

The visualization of generated embeddings can be found in appendix in figure A.4. The figure shows 2-dimensional t-SNE visualization of a random sample of 2048 URLs. Also, corresponding learning curve which shows dependency of the value of loss function on the mini-batch step is also presented in appendix in figure A.1.

Embeddings produced by this model have several drawbacks, which results in poor accuracy of the Markov model. As can be seen from the embeddings visualization, the produced space has practically no structure. Apart from a random noise, we can observe one cluster that contains the most visited URLs related to customer account and checkout process – which are tasks that most users have to perform.

Learning curve (figure A.1) indicates that the model has not enough capacity to capture information from training data. This can be possibly caused by the fact that the dataset contains too many URLs that appear rarely and that the total number of URLs in vocabulary is too large in comparison with the size of hidden layer.

As can be seen in the table 6.2, the accuracy of the Markov model on top of these embeddings is much lower than the baseline on both training and testing data.

6.4.2 EMB2.1

Embeddings produced by the model EMB2.1 combine both linguistic and behavioral information, because the embedding of an URL is a combination of embeddings of its tokens, as described in section 4.4.4. Embeddings in this experiment were trained using binary BOW representation of tokens and context of size 3. Dimensionality of embeddings was set to 512. The neural network was trained on mini-batches of size 256, using Adagrad optimizer with learning rate 0.01.

The visualization of produced embeddings, which is presented in figure A.5 shows that the space is highly structured and contains large number of clusters. Similarly to the previous visualization, we can see a sample of 2048 URL embeddings reduced to 2-dimensional space by t-SNE. The learning curve, which is present in figure A.2 shows that the value of loss function is much lower than in case of the previous model.

The Markov model with clustered representations generated by this model is able to outperform the baseline model on both tasks TOP1 and TOP5. The improvement is more significant on the task TOP5, which is expected, as embeddings improve the model with global information, which mainly improves recall of the model.

Embeddings produced by this model were used for two following experiments, comparison of TOP1 and TOP5 tasks and analysis of models with different numbers of clusters, which are presented in sections 6.5 and 6.6, respectively.

6.4.3 EMB2.2

Architecture EMB2.2 is built on top of bag-of-trigrams representation of URL. The neural network was trained using Adagrad optimizer with learning rate 0.001 and mini-batch size 128. Target URLs were selected from context of size 3.

The space produced by embeddings, as seen in figure A.6, which was created in a same way as previous two visualizations, is comparable to the space produced by embeddings generated by model EMB2.1. While one can find subtle differences between these two, it is important to note that some of the structural differences may be caused by t-SNE projection and that the real situation in the 512-dimensional space may be different.

The learning curve, presented in figure A.3, indicates better ability to capture information from training data than in case of EMB1 model, however the resulting value of loss function after training is still higher than in case of model EMB2.1.

Results on the clickstream prediction are slightly better than results of EMB1 model,

	train		test	
	top1	top5	top1	top5
baseline	0.6631 ± 0.0031	0.8679 ± 0.0036	0.3107 ± 0.0482	0.4319 ± 0.0536
emb1 (27k)	0.2910 ± 0.0073	0.5000 ± 0.0065	0.0603 ± 0.0083	0.1653 ± 0.0305
emb2.1 (27k)	0.5723 ± 0.0044	0.8066 ± 0.0041	$\mathbf{0.3216 \pm 0.0483}$	0.4607 ± 0.0535
emb2.1 (2k)	0.3506 ± 0.0116	0.5997 ± 0.0130	0.2854 ± 0.0435	$\mathbf{0.4842 \pm 0.0543}$
emb2.2 (27k)	0.3304 ± 0.0071	0.5715 ± 0.0024	0.1642 ± 0.0045	0.3427 ± 0.0316

Table 6.2: Markov models on clustered embeddings. The baseline model is in this case simple Markov model of second order. Other models are also Markov models of second order, but with clustered representations of URLs. Number of clusters is indicated in parenthesis.

however the accuracy is still lower than baseline. Inspection of produced clusters has shown that the distribution is different compared to those produced by EMB2.1, and that the fraction of large clusters is higher. This may be caused due to worse ability to distinguish similar URLs of the model based on character trigrams. The number of individual trigram embeddings that form the final embedding of URLs is larger than in the case of tokens and the selected combination method, as described in 4.4.4, may introduce larger amount of noise to the model.

6.5 The task of TOP5 prediction

As was already stated in section 2.2, the task of exact clickstream prediction is quite complicated. Also, in many applications our goal is to recommend more than one possible succeeding URL, therefore we may be interested in less strict task of TOP5 prediction.

We have compared both Markov models on simple URL representations and models on clustered representations. In both cases the model has order 2, and was trained on the dataset DATA-500K. Results were obtained using k-fold cross-validation with $k = 5$.

Train accuracy for the regular Markov model in task of TOP1 prediction is 0.6631 ± 0.0031, test accuracy is 0.3110 ± 0.0474. The model is identical to the one which is presented in section 6.3.1, results are therefore same – minus small variance due to different train test splits – to results in column 2 of table 6.1. If we lower our requirements for the model and measure its performance on the TOP5 task, train accuracy goes up to 0.8679 ± 0.0036, while test accuracy increases to 0.4319 ± 0.0536. This trend can be seen in the figure 6.4.

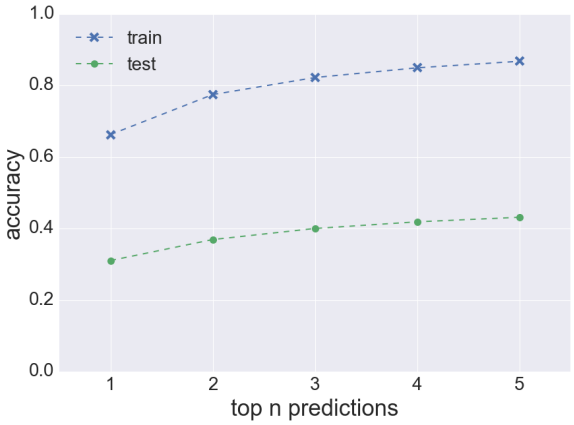

Figure 6.4: Accuracy of simple Markov model on TOPN prediction tasks.

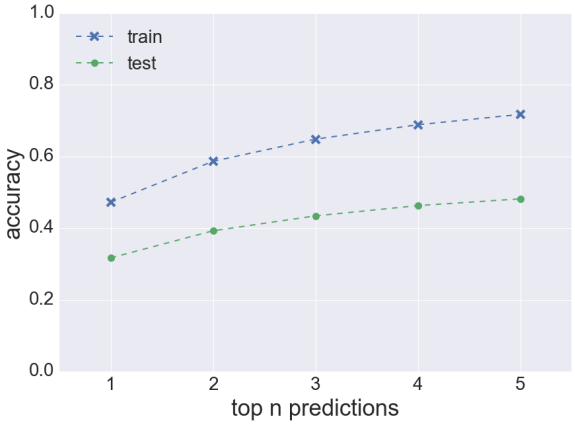

Figure 6.5: Accuracy of Markov model on TOPN prediction tasks, using clustered representations generated with EMB2.1 model.

6.6 Markov models with different numbers of clusters

As was stated in the section 4.6, clustering granularity is one of the key factors that affect model performance. We have evaluated accuracy of a Markov model of second order with clustered representations. This experiment was performed on dataset DATA-500K, with representations produced by model EMB2.1 with the same setting as in the experiment 6.4.2. Clusters were obtained using hierarchical clustering and using cosine similarity as a linkage criteria. The number of clusters was selected as a fraction of number of URLs – ranging from 1/2 to 1/50. We have measured accuracy of the model on both TOP1 and TOP5 tasks. Presented results are mean from k-fold cross-validation with $k = 5$.

Results are shown in the figure 6.8. It can be seen that when the number of clusters is high, therefore each cluster contains small number of URLs, the model overfits to training data. As the number of clusters is decreasing, the performance on training set decreases as well. The difference in accuracy on testing set is surprisingly low – the best result achieved on testing set is 0.3216 ± 0.0483 with 27354 clusters (1/2 * number of URLs) for the task of TOP1 prediction and 0.4842 ± 0.0543 with just 1823 clusters (1/25 * number of URLs) for the task TOP5. This result indicates that URL embeddings have captured significant amount of information needed for clickstream modeling.

Histograms of cluster sizes, that can be seen in figure 6.6, show that even when the number of clusters is large, there is still significant number of clusters that contain more than 300 URLs.

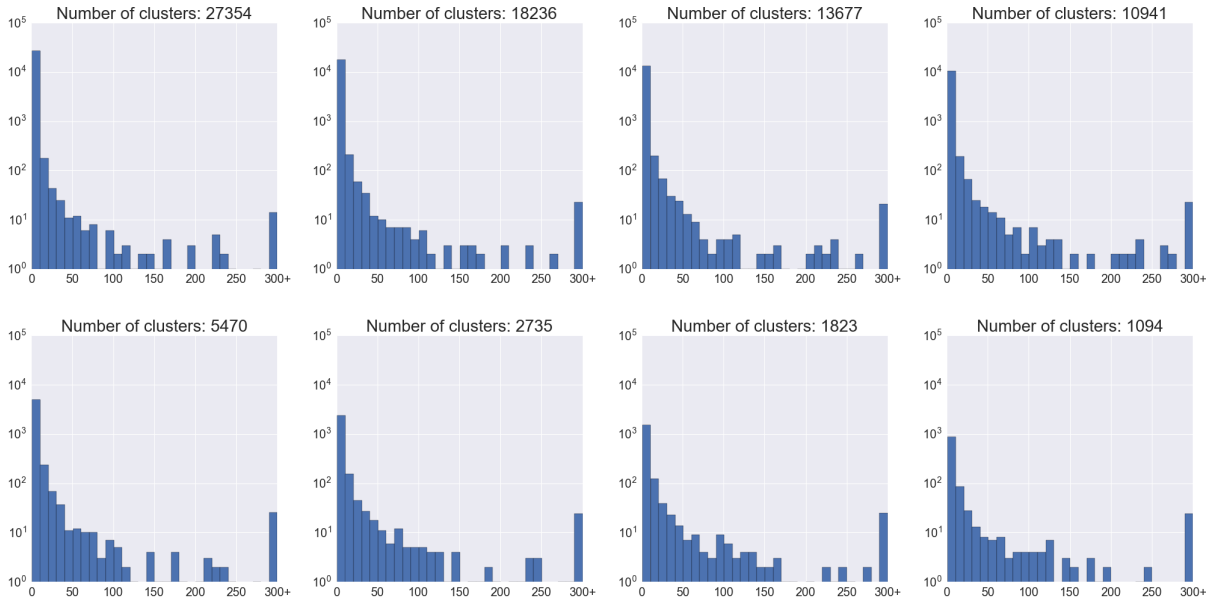

Figure 6.6: Histogram of cluster size for different numbers of clusters. The x axis shows number of URLs in a cluster. Clusters of size larger than 300 were merged into the last bin. The y axis is presented in logarithmic scale and denotes the number of clusters in the bin.

	train		test	
n. clusters	TOP1	TOP5	TOP1	TOP5
1094	0.2783 ± 0.0068	0.5603 ± 0.0142	0.2318 ± 0.0250	0.4706 ± 0.0532
1823	0.3506 ± 0.0116	0.5997 ± 0.0130	0.2854 ± 0.0435	$\mathbf{0.4842 \pm 0.0543}$
2735	0.3651 ± 0.0106	0.6191 ± 0.0118	0.2870 ± 0.0435	0.4838 ± 0.0542
5470	0.4224 ± 0.0105	0.6649 ± 0.0094	0.3112 ± 0.0503	0.4830 ± 0.0546
10941	0.4722 ± 0.0087	0.7174 ± 0.0065	0.3172 ± 0.0505	0.4819 ± 0.0537
13677	0.4925 ± 0.0079	0.7351 ± 0.0059	0.3192 ± 0.0503	0.4775 ± 0.0550
18236	0.5228 ± 0.0067	0.7629 ± 0.0051	0.3216 ± 0.0484	0.4717 ± 0.0536
27354	0.5723 ± 0.0044	0.8066 ± 0.0041	$\mathbf{0.3216 \pm 0.0483}$	0.4607 ± 0.0535

Table 6.3: Accuracy of Markov model on different numbers of clusters of URL embeddings. Numbers of clusters are selected as a fraction of the number of URLs, ranging from 1/50 (1094 clusters) to 1/2 (27354 clusters). Embeddings were generated with EMB2.1 model with same configuration as in experiment in section 6.4.2.

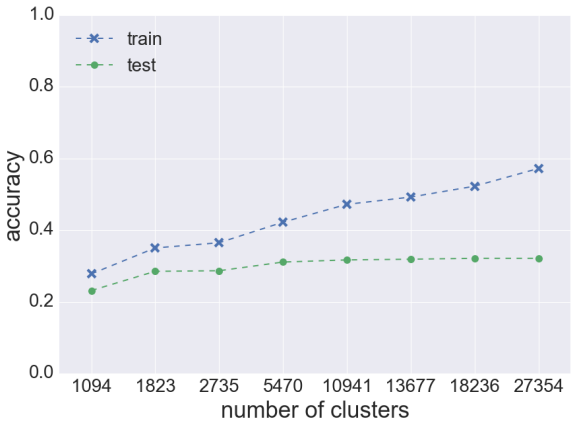

Figure 6.7: Accuracy of Markov model on different numbers of clusters for the task of TOP1 prediction

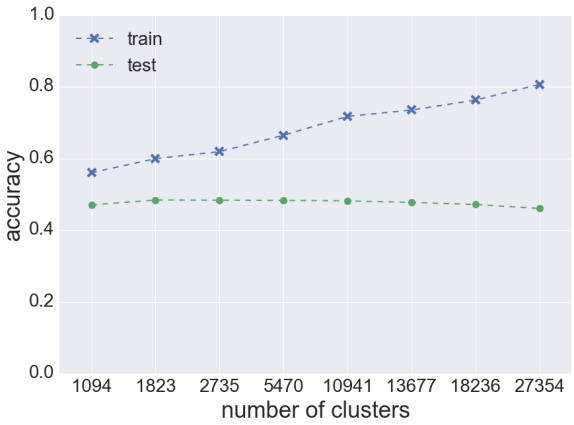

Figure 6.8: Accuracy of Markov model on different numbers of clusters for the task of TOP5 prediction

Chapter 7

Conclusion

The goal of this thesis was to propose and evaluate an approach for clickstream prediction using URL embeddings. The first part presents current approaches in both clickstream prediction and in word embeddings generation. We have studied several approaches for clickstream prediction based on Markov models, Association rules and other techniques; as well as approaches used for other tasks closely related to modeling user behavior. In the survey of word embedding techniques, we have described several methods based on matrix factorization and shallow neural networks, and also explored more complex task of document embeddings.

The second part then presents proposed method for generation of URL embeddings and clickstream prediction. We have proposed and evaluated three approaches for generation of URL embeddings, which are based on the idea of similarity between clickstream and language. These embeddings were then used in the task of clickstream prediction using Markov models.

While the performance of two of proposed approaches have not reached the baseline, the third proposed approach was able to deliver more than 10% improvement of the baseline model (increasing accuracy from 0.43 to 0.48), while using 25 times more compact representation. These results implies that produced URL embeddings capture significant amount of information about user behavior.

Even though our results are not significant enough for the sole task of clickstream prediction, the performance of generated URL embeddings is promising and encourages applications in related tasks and further research in this area.

7.1 Future work

This thesis presents initial exploration of an application of URL embeddings based on the idea of clickstream - language similarity. While achieved results look promising, they are

far away from the possible production use. We consider three possible directions of future research. First, the quality of embeddings can be evaluated on other related tasks like categorization and others that were discussed in section 2.1. Second, while this thesis is only limited to a single domain, generalization to arbitrary number of domains would be important for real world use. Last, but not least, newer approaches in deep learning favor so called end-to-end learning, where the whole task is performed by single neural network. It would be interesting to see whether this principle can replace proposed combination of neural network and Markov model.

Appendix A

Appendix

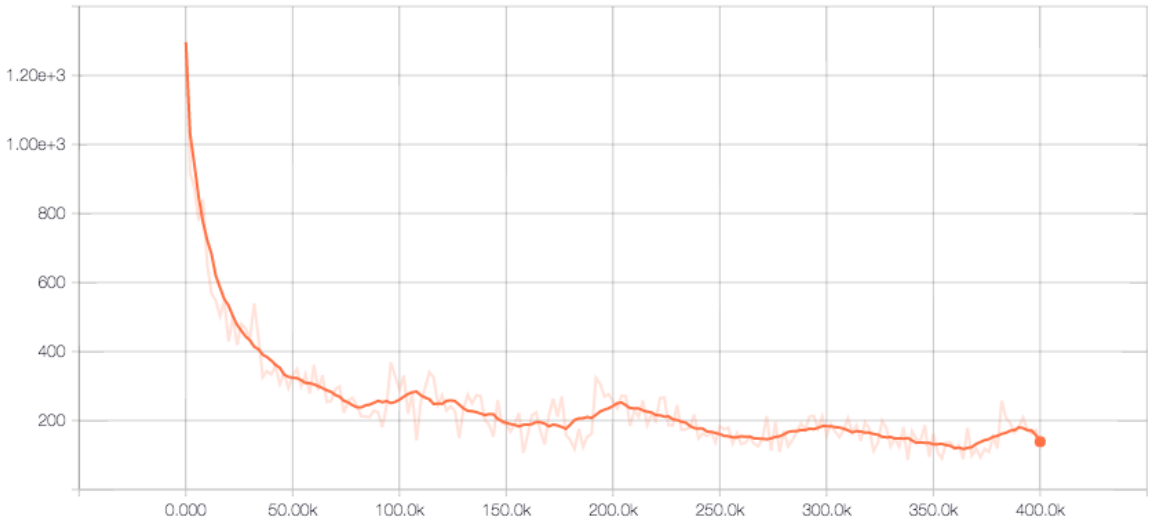

Figure A.1: Learning curve of EMB1 model

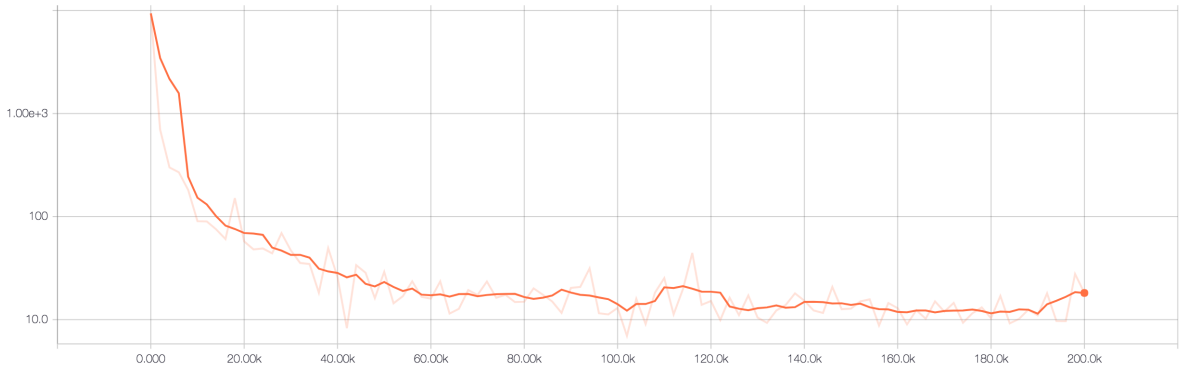

Figure A.2: Learning curve of EMB2.1 model

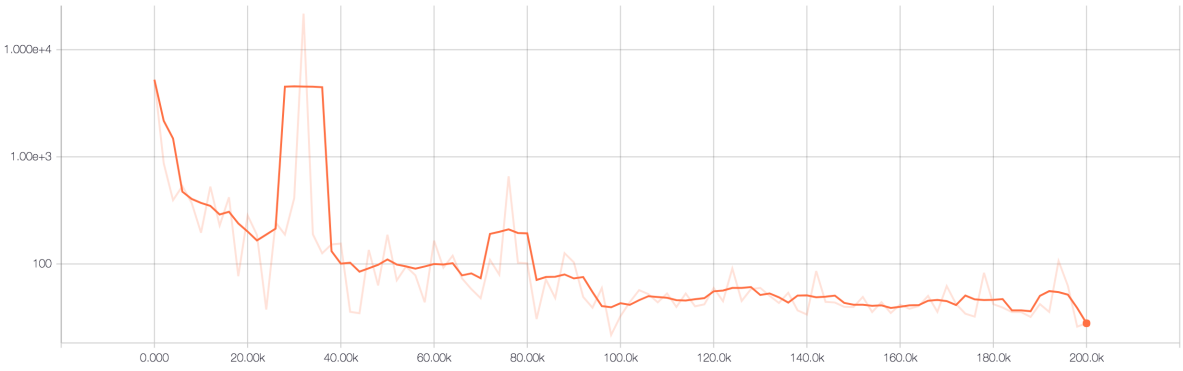

Figure A.3: Learning curve of EMB2.2 model

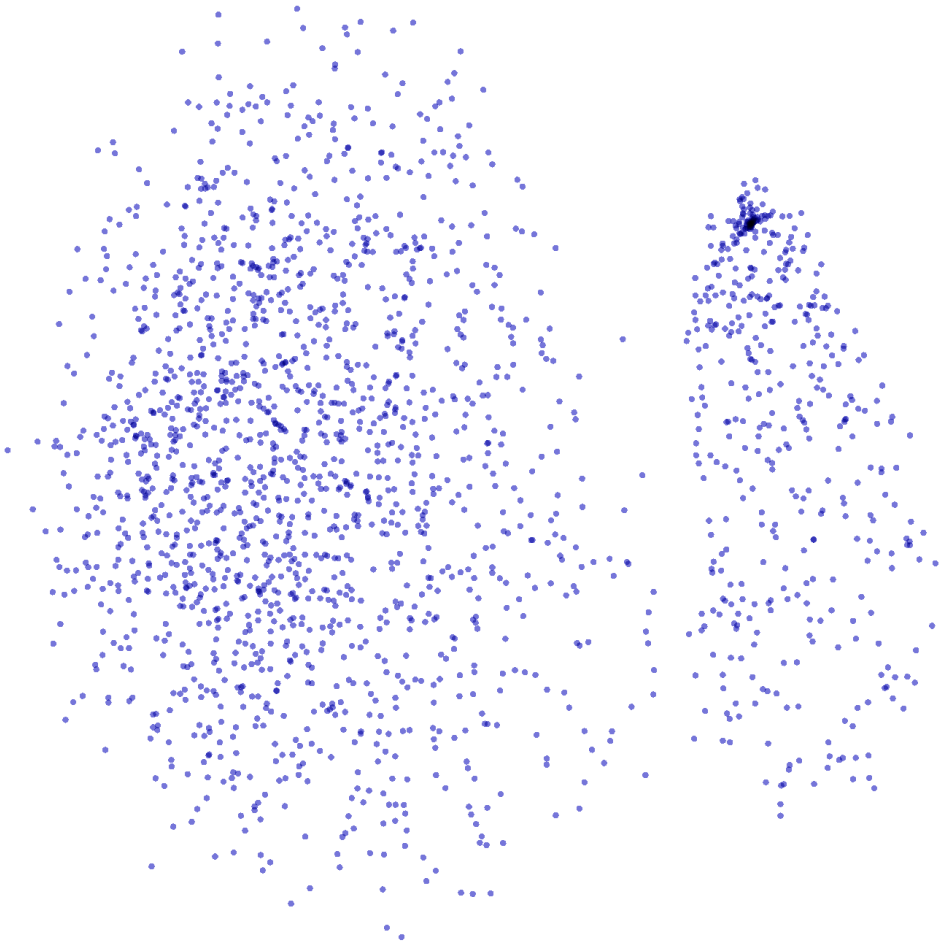

Figure A.4: t-SNE projection of embeddings generated with EMB1 model

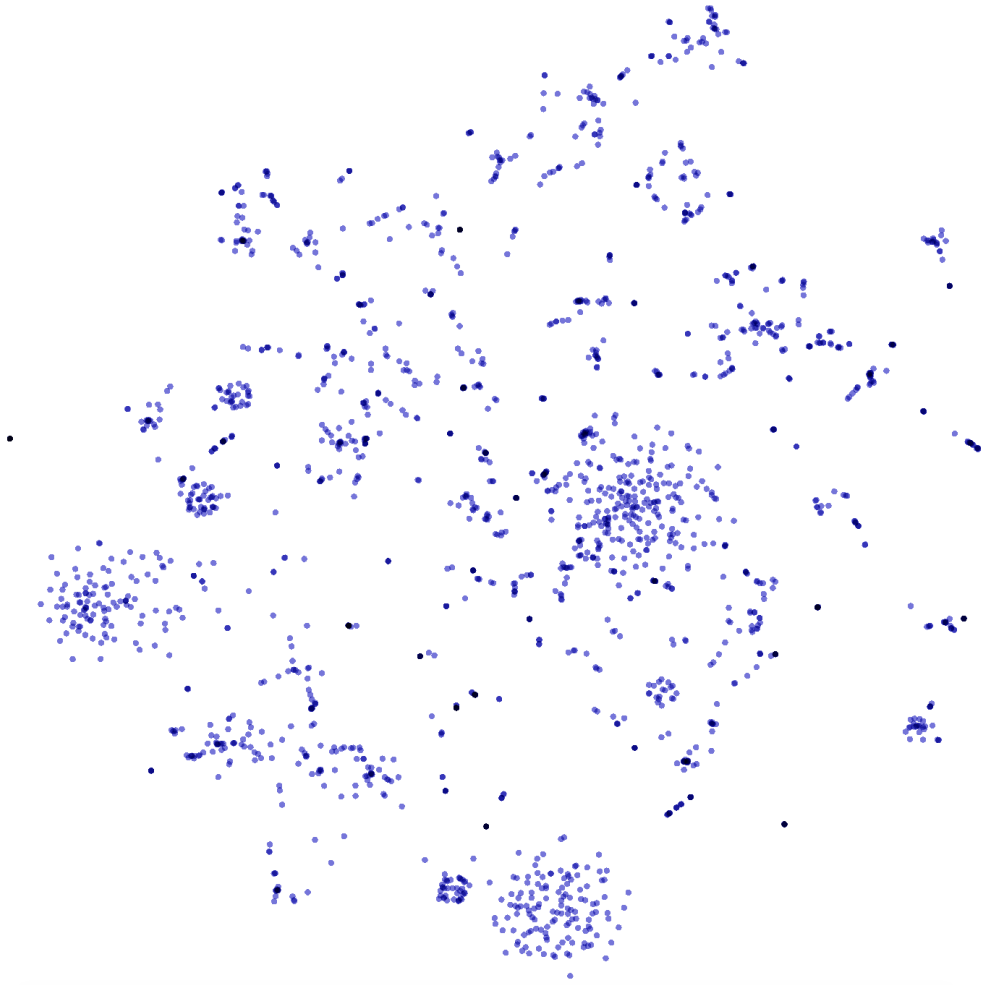

Figure A.5: t-SNE projection of embeddings generated with EMB2.1 model

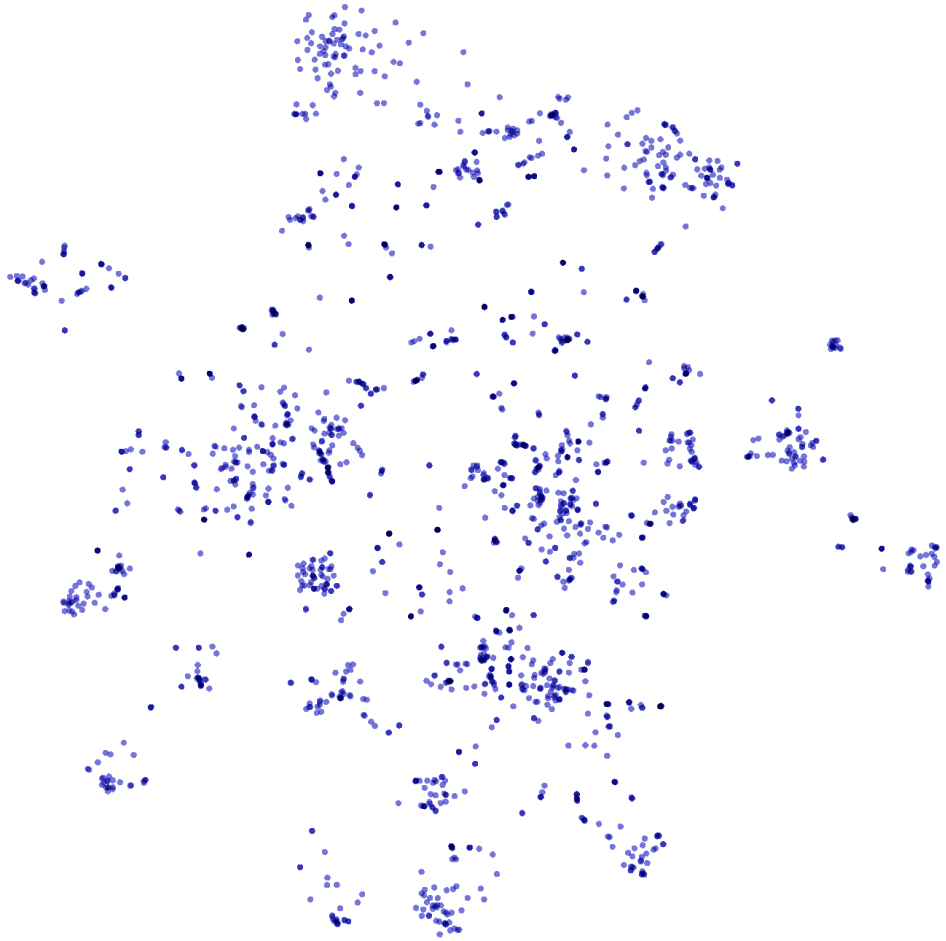

Figure A.6: t-SNE projection of embeddings generated with EMB2.2 model

Bibliography

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases", in *Proceedings of the 1993 acm sigmod international conference on management of data - sigmod '93*, vol. 22, New York, New York, USA: ACM Press, 1993, pp. 207–216, ISBN: 0897915925. DOI: 10.1145/170035.170072. [Online]. Available: http://portal.acm.org/citation.cfm?doid=170035.170072.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space", 2013. arXiv: 1301.3781. [Online]. Available: http://arxiv.org/abs/1301.3781.

[3] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing", *Communications of the acm*, vol. 18, no. 11, pp. 613–620, 1975, ISSN: 00010782. DOI: 10.1145/361219.361220. [Online]. Available: http://portal.acm.org/citation.cfm?doid=361219.361220.

[4] S. Deerwester, S. T. Dumais, G. W. Furnas, and T. K. Landauer, "Indexing by Latent Semantic Analysis", *Journal of the american society for information science sep*, vol. 41, no. 6, 1986.

[5] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation", *Emnlp*, vol. 14, 2014. [Online]. Available: http://nlp.stanford.edu/pubs/glove.pdf.

[6] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora", 2010. [Online]. Available: https://radimrehurek.com/gensim/lrec2010_final.pdf.

[7] K. Lund and C. Burgess, "Producing high-dimensional semantic spaces from lexical co-occurrence", *Behavior research methods, instruments, & computers*, vol. 28, no. 2, pp. 203–208, 1996, ISSN: 0743-3808. DOI: 10.3758/BF03204766. [Online]. Available: http://www.springerlink.com/index/10.3758/BF03204766.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors.", *Nature*, 1986.

[9] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, J. Kandola, T. Hofmann, T. Poggio, and J. Shawe-Taylor, "A Neural Probabilistic Language Model", *Journal of machine learning research*, vol. 3, pp. 1137–1155, 2003.

[10] R. Collobert and J. Weston, "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning", *Proceedings of the 25th international conference on machine learning*, 2008. [Online]. Available: http://www.thespermwhale.com/jaseweston/papers/unified_nlp.pdf.

[11] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) from Scratch", *Journal of machine learning research*, vol. 12, no. Aug, pp. 2493–2537, 2011, ISSN: ISSN 1533-7928. [Online]. Available: http://www.jmlr.org/papers/v12/collobert11a.html.

[12] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information", 2016. arXiv: arXiv:1607.04606v1. [Online]. Available: https://arxiv.org/pdf/1607.04606.pdf.

[13] Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents", *Proceedings of the 31st international conference on machine learning*, 2014. arXiv: arXiv:1405.4053v2.

[14] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, "Skip-Thought Vectors", *Advances in neural information processing systems 28*, 2015. [Online]. Available: http://papers.nips.cc/paper/5950-skip-thought-vectors.pdf.

[15] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences", *Proceedings of the 52nd annual meeting of the association for computational linguistics*, 2014. [Online]. Available: https://arxiv.org/pdf/1404.2188.pdf.

[16] K. S. Tai, R. Socher, and C. D. Manning, "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks", *Arxiv preprint arxiv:1503.00075*, 2015. [Online]. Available: https://arxiv.org/pdf/1503.00075.pdf.

[17] E. Asgari and M. R. K. Mofrad, "Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics", 2015. DOI: 10.1371/journal.pone.0141287. [Online]. Available: http://llp.http://dx.doi.org/10.7910/DVN/JMFHTN..

[18] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, "subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs", *Arxiv preprint arxiv:1606.08928*, 2016. DOI: 10.1145/1235. [Online]. Available: https://arxiv.org/pdf/1606.08928.pdf.

[19] Q. Ma, S Muthukrishnan, and W. Simpson, "App2vec: Vector modeling of mobile apps and applications", in *Advances in social networks analysis and mining (asonam), 2016 ieee/acm international conference on*, IEEE, 2016, pp. 599–606.

[20] S. Arora and D. Warrier, "Decoding Fashion Contexts Using Word Embeddings", *Machine learning meets fashion, kdd2016 workshop*, 2016.

[21] A. Borisov, I. Markov, M. de Rijke, and P. Serdyukov, "A Neural Click Model for Web Search", *Proceedings of the 25th international conference on world wide web*, pp. 531–541, 2016. DOI: 10.1145/2872427.2883033. [Online]. Available: http://dx.doi.org/10.1145/2872427.2883033..

[22] G. Stumme, A. Hotho, and B. Berendt, "Semantic Web Mining State of the art and future directions", *Web semantics : Science , services and agents on the world wide web*, vol. 4, pp. 124–143, 2006.

[23] R. Cooley, B. Mobasher, and J. Srivastava, "Web mining: information and pattern discovery on the World Wide Web", in *Proceedings ninth ieee international conference on tools with artificial intelligence*, IEEE Comput. Soc, 1997, pp. 558–567, ISBN: 0-8186-8203-5. DOI: 10.1109/TAI.1997.632303. [Online]. Available: http://ieeexplore.ieee.org/document/632303/.

[24] M. Deshpande and G. Karypis, "Selective Markov Models for Predicting Web-Page Accesses", 2000. [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/a439247.pdf.

[25] X. Dongshan and J. Shen, "A new markov model for web access prediction", 2002.

[26] F. Khalil, J. Li, and H. Wang, "A Framework of Combining Markov Model with Association Rules for Predicting Web Page Accesses", 2006.

[27] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa, "Effective Personalization Based on Association Rule Discovery from Web Usage Data", *Proceedings of the 3rd international workshop on web information and data management*, 2001.

[28] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", *Proc. 20th int. conf. very large databases, vldb*, 1994.

[29] W. Yong, L. Zhanhuai, and Z. Yang, "Mining Sequential Association-Rule for Improving WEB Document Prediction", in *Sixth international conference on computational intelligence and multimedia applications (iccima'05)*, IEEE, 2005, pp. 146–151, ISBN: 0-7695-2358-7. DOI: 10.1109/ICCIMA.2005.38. [Online]. Available: http://ieeexplore.ieee.org/document/1540717/.

[30] D.-H. Kim, V. Atluri, M. Bieber, N. Adam, and Y. Yesha, "A clickstream-based collaborative filtering personalization model", in *Proceedings of the 6th annual acm international workshop on web information and data management - widm '04*, New York, New York, USA: ACM Press, 2004, p. 88, ISBN: 1581139780. DOI: 10.1145/1031453.1031470. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1031453.1031470.

[31] S. Vishwakarma, S. Lade, M. K. Suman, and D. Patel, "Web page access prediction based on integrated approach", *Int. j. engg. res. & sci. & tech*, 2013. [Online]. Available: http://www.ijerst.com/ijerstadmin/upload/IJEETC_527a74b158854.pdf.

[32] J. Kiseleva, H. T. Lam, M. Pechenizkiy, and T. Calders, "Predicting Current User Intent with Contextual Markov Models", in *2013 ieee 13th international conference on data mining workshops*, IEEE, 2013, pp. 391–398, ISBN: 978-1-4799-3142-2. DOI: 10.1109/ICDMW.2013.143. [Online]. Available: http://ieeexplore.ieee.org/document/6753947/.

[33] G. Dupret and B. Piwowarski, "A User Browsing Model to Predict Search Engine Click Data from Past Observations", *Proceedings of the 31st annual international acm sigir conference on research and development in information retrieval*, 2008.

[34] F. Guo, C. Liu, A. Kannan, T. Minka, M. Taylor, Y.-M. Wang, and C. Faloutsos, "Click Chain Model in Web Search", *Proceedings of the 18th international conference on world wide web*, 2009. [Online]. Available: http://www2009.eprints.org/2/1/p11.pdf.

[35] Y. Zhang, H. Dai, C. Xu, J. Feng, T. Wang, J. Bian, B. Wang, and T.-Y. Liu, "Sequential click prediction for sponsored search with recurrent neural networks", *Arxiv preprint arxiv:1404.5772*, 2014.

[36] A. Chuklin, I. Markov, and M. de Rijke, "Click Models for Web Search", *Synthesis lectures on information concepts, retrieval, and services*, vol. 7, no. 3, pp. 1–115, 2015, ISSN: 1947-945X. DOI: 10.2200/S00654ED1V01Y201507ICR043. [Online]. Available: http://www.morganclaypool.com/doi/10.2200/S00654ED1V01Y201507ICR043.

[37] N. Sadagopan and J. Li, "Characterizing Typical and Atypical User Sessions in Clickstreams", *Proceedings of the 17th international conference on world wide web*, 2008.

[38] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, "Implicit Authentication through Learning User Behavior", *International conference on information security*, 2010.

[39] G. Wang, T. Konolige, C. Wilson, X. Wang, H. Zheng, B. Y. Zhao, and U. C. S. Barbara, "You are how you click: Clickstream analysis for Sybil detection", *Usenix security 2013*, p. 15, 2013.

[40] G. Wang, X. Zhang, S. Tang, H. Zheng, and B. Y. Zhao, "Unsupervised Clickstream Clustering for User Behavior Analysis", *Proceedings of the 2016 chi conference on human factors in computing systems - chi '16*, pp. 225–236, 2016. DOI: 10.1145/2858036.2858107.

[41] V. Melnykov, "Model-based biclustering of clickstream data", *Computational statistics and data analysis*, vol. 93, pp. 31–45, 2014, ISSN: 01679473. DOI: 10.1016/j.csda.2014.09.016.

[42] S. Ruder, "An overview of gradient descent optimization algorithms", 2016. [Online]. Available: https://arxiv.org/pdf/1609.04747.pdf.

[43] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models", *Aistats*, vol. 1, p. 6, 2010. [Online]. Available: http://proceedings.mlr.press/v9/gutmann10a/gutmann10a.pdf.

[44] J. Duchi, J. B. Edu, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of machine learning research*, vol. 12, pp. 2121–2159, 2011. [Online]. Available: http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf.

[45] F. Pérez and B. E. Granger, "IPython: A System for Interactive Scientific Computing", *Computing in science and engineering*, vol. 9, no. 3, 2007. [Online]. Available: http://ipython.org.

[46] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", 2016. arXiv: 1603.04467. [Online]. Available: http://tensorflow.org/.

[47] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation", *Computing in science & engineering*, vol. 13, no. 2, pp. 22–30, 2011. DOI: 10.1109/MCSE.2011.37. [Online]. Available: http://ieeexplore.ieee.org/document/5725236/.

[48] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python*, 2001–. [Online]. Available: http://www.scipy.org/.

[49] M. Zaharia, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, and S. Venkataraman, "Apache Spark", *Communications of the acm*, vol. 59, no. 11, pp. 56–65, 2016, ISSN: 00010782. DOI: 10.1145/2934664. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3013530.2934664.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python", *Journal of machine learning research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf.

[51] W. McKinney, *Data Structures for Statistical Computing in Python*, 2010. [Online]. Available: http://pandas.pydata.org/.

[52] F. Chollet and Others, *Keras*, 2015. [Online]. Available: https://github.com/fchollet/keras.

[53] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. B. Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. De Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. E. Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. Mcgibbon, R. Memisevic, B. Van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. Van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. De Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang, "Theano: A Python framework for fast computation of mathematical expressions", 2016. arXiv: arXiv:1605.02688v1. [Online]. Available: https://arxiv.org/pdf/1605.02688.pdf.

[54] L. Van Der Maaten and G. Hinton, "Visualizing Data using t-SNE", *Journal of machine learning research*, vol. 9, pp. 2579–2605, 2008.

[55] M. Wattenberg, F. Viégas, and I. Johnson, "How to Use t-SNE Effectively", *Distill*, vol. 1, no. 10, e2, 2016, ISSN: 2476-0757. DOI: 10.23915/distill.00002. [Online]. Available: http://distill.pub/2016/misread-tsne.