Embedding-based News Recommendation for Millions of Users

Shumpei Okura
Yahoo Japan Corporation
Tokyo, Japan
sokura@yahoo-corp.jp

Yukihiro Tagami
Yahoo Japan Corporation
Tokyo, Japan
yutagami@yahoo-corp.jp

Shingo Ono
Yahoo Japan Corporation
Tokyo, Japan
shiono@yahoo-corp.jp

Akira Tajima
Yahoo Japan Corporation
Tokyo, Japan
atajima@yahoo-corp.jp

ABSTRACT

It is necessary to understand the content of articles and user preferences to make effective news recommendations. While ID-based methods, such as collaborative filtering and low-rank factorization, are well known for making recommendations, they are not suitable for news recommendations because candidate articles expire quickly and are replaced with new ones within short spans of time. Word-based methods, which are often used in information retrieval settings, are good candidates in terms of system performance but have issues such as their ability to cope with synonyms and orthographical variants and define "queries" from users' historical activities. This paper proposes an embedding-based method to use distributed representations in a three step end-to-end manner: (i) start with distributed representations of articles based on a variant of a denoising autoencoder, (ii) generate user representations by using a recurrent neural network (RNN) with browsing histories as input sequences, and (iii) match and list articles for users based on inner-product operations by taking system performance into consideration.

The proposed method performed well in an experimental offline evaluation using past access data on Yahoo! JAPAN's homepage. We implemented it on our actual news distribution system based on these experimental results and compared its online performance with a method that was conventionally incorporated into the system. As a result, the click-through rate (CTR) improved by 23% and the total duration improved by 10%, compared with the conventionally incorporated method. Services that incorporated the method we propose are already open to all users and provide recommendations to over ten million individual users per day who make billions of accesses per month.

KDD'17, August 13–17, 2017, Halifax, NS, Canada.

ISBN 978-1-4503-4887-4/17/08…$15.00
DOI: http://dx.doi.org/10.1145/3097983.3098108

KEYWORDS

News recommendations, Neural networks, Distributed representations, Large-scale services

1 INTRODUCTION

It is impossible for users of news distributions to read all available articles due to limited amounts of time. Thus, users prefer news services that can selectively provide articles. Such selection is typically done manually by editors and a common set of selected stories are provided to all users in outmoded media such as television news programs and newspapers. However, we can identify users before they select articles that will be provided to them on the Internet by using information, such as that in user ID cookies, and personalize the articles for individual users [3, 22].

ID-based methods, such as collaborative filtering and low-rank factorization, are well known in making recommendations. However, Zhong et al. [22] suggested that such methods were not suitable for news recommendations because candidate articles expired too quickly and were replaced with new ones within short time spans. Thus, the three keys in news recommendations are:

- Understanding the content of articles,
- Understanding user preferences, and
- Listing selected articles for individual users based on content and preferences.

In addition, it is important to make recommendations in the real world [14] that respond to scalability and noise in learning data [14]. Applications also need to return responses within hundreds of milliseconds with every user access.

A baseline implementation to cover the three keys would be as follows. An article is regarded as a collection of words included in its text. A user is regarded as a collection of words included in articles he/she has browsed. The implementation learns click probability using co-occurrence of words between candidates of articles and browsing histories as features.

This method has some practical advantages. It can immediately reflect the latest trends because the model is very simple so that the model can be taught to learn and update in short periods of time. The estimation of priority can be quickly calculated using existing search engines with inverted indices on words.

The previous version of our implementation was based on this method for these reasons. However, it had some issues that may have had a negative impact on the quality of recommendations.

The first regarded the representation of words. When a word was used as a feature, two words that had the same meaning were treated as completely different features if the notations were different. This problem tended to emerge in news articles when multiple providers separately submitted articles on the same event.

The second regarded the handling of browsing histories. Browsing histories were handled as a set in this approach. However, they were actually a sequence, and the order of browsing should have represented the transition of user interests. We also had to note large variances in history lengths by users that ranged from private browsing to those who visited sites multiple times per hour.

Deep-learning-based approaches have recently been reported to be effective in various domains. Distributed representations of words thoroughly capture semantic information [11, 16]. Recurrent neural networks (RNNs) have provided effective results as a method of handling input sequences of variable length [9, 15, 17].

If we build a model with a deep network using an RNN to estimate the degree of interest between users and articles, on the other hand, it is difficult to satisfy the response time constraints on accesses in real systems. This paper proposes an embedding-based method of using distributed representations in a three step end-to-end manner from representing each article to listing articles for each user based on relevance and duplication.

- Start with distributed representations of articles based on a variant of the denoising autoencoder (which addresses the first issue in Section 3).
- Generate user representations by using an RNN with browsing histories as input sequences (which addresses the second issue in Section 4).
- Match and list articles for each user based on the inner product of article-user for relevance and article-article for de-duplication (outlined in Section 2).

The key to our method is using a simple inner product to estimate article-user relevance. We can calculate article representations and user representations before user visits in sufficient amounts of time. When a user accesses our service, we only select his/her representations and calculate the inner product of candidate articles and the representations. Our method therefore both expresses complex relations that are included in the user's browsing history and satisfies the response time constraints of the real system.

The proposed method was applied to our news distribution service for smartphones, which is described in the next section. We compared our method to a conventional approach, and the results (see Section 6) revealed that the proposed method outperformed the conventional approach with a real service as well as with static experimental data, even if disadvantages, such as increased learning time and large latency in model updates, are taken into consideration.

2 OUR SERVICE AND PROCESS FLOW

The methods discussed in this paper were designed to be applied to the news distribution service on the homepage of Yahoo! JAPAN on smartphones. The online experiments described in Section 6 were also conducted on this page. This section introduces our service and process flow.

Figure 1: Example of Yahoo! JAPAN's homepage on smartphones. This paper discusses methods of providing articles in Personalized module.

Figure 1 has a mockup of our service that was renewed in May 2015. There is a search window and links to other services at the top as a header. The middle part, called the *Topics module*, provides six articles on major news selected by human professionals for a general readership. The bottom part, called the *Personalized module*, provides many articles and advertising that has been personalized for individual users. Users in the Personalized module can see as many articles as they want if they scroll down to the bottom. Typical users scroll down to browse the approximately top 20 articles. This paper describes optimization to provide articles in the Personalized module.

Five processes are executed to select articles for millions of users for each user access.

- **Identify**: Obtain user features calculated from user history in advance.
- **Matching**: Extract articles from all those available using user features.
- **Ranking**: Rearrange list of articles on certain priorities.
- **De-duplication**: Remove articles that contain the same information as others.
- **Advertising**: Insert ads if necessary.

These processes have to be done within hundreds of milliseconds between user requests and when they are displayed because available articles are constantly changing. In fact, as all articles in our service expire within 24 hours from the viewpoint of information freshness, tens of thousands of new articles are posted every day, and the same number of old articles are removed due to expiration. Thus, each process adopts computationally light methods that leverage pre-computed distributed article representations (described in Section 3) and user representations (described in Section 4).

We use the inner product of distributed representations of a user and candidate articles in matching to quantify relevance and select promising candidates. We determine the order of priorities in ranking by considering additional factors, such as the expected number of page views and freshness of each article, in addition to the relevance used for matching. We skip similar articles in a greedy manner in de-duplication based on the cosine similarity of distributed representations. An article is skipped when the maximum value of its cosine similarity with articles with higher priorities is above a threshold. This is an important process in real news distribution services because similar articles tend to have similar scores in ranking. If similar articles are displayed close to one another, a real concern is that user satisfaction will decrease due to reduced diversity on the display. Details on comparison experiments in this process have been discussed in a report on our previous study [12]. Advertising is also important, but several studies [2, 10] have already reported on the relationship between advertising and user satisfaction, so such discussion has been omitted here.

3 ARTICLE REPRESENTATIONS

Section 1 discussed a method of using words as features for an article that did not work well in certain cases of extracting and de-duplicating. This section describes a method of dealing with articles as a distributed representation. We proposed a method in our previous study [12], from which part of this section has been excerpted.

3.1 Generating Method

Our method generates distributed representation vectors on the basis of a denoising autoencoder [19] with weak supervision. The conventional denoising autoencoder is formulated as:

$$\begin{aligned} \tilde{x} &\sim q(\tilde{x}|x) \\ h &= f(W\tilde{x}+b) \\ y &= f(W'h+b') \\ \theta &= \underset{W,W',b,b'}{\arg\min} \sum_{x\in X} L_R(y,x), \end{aligned}$$

where $x \in X$ is the original input vector and $q(\cdot|\cdot)$ is the corrupting distribution. The stochastically corrupted vector, $\tilde{x}$, is obtained from $q(\cdot|x)$. The hidden representation, h, is mapped from $\tilde{x}$ through the network, which consists of an activation function, $f(\cdot)$, parameter matrix W, and parameter vector b. In the same way, the reconstructed vector, y, is also mapped from h with parameters W' and b'. Using a loss function, $L_R(\cdot,\cdot)$, we learn these parameters to minimize the reconstruction errors of y and x.

The h is usually used as a representation vector that corresponds to x. However, h only holds the information of x. We want to interpret that the inner product of two representation vectors ${h_0}^{\mathrm{T}}h_1$ is larger if x_0 is more similar to x_1. To achieve that end, we use a triplet, $(x_0, x_1, x_2) \in X^3$, as input for training and modify the objective function to preserve their categorical similarity as:

$$\begin{aligned} \tilde{x}_n &\sim q(\tilde{x}_n|x_n) \\ h_n &= f(W\tilde{x}_n+b)-f(b) \\ y_n &= f(W'h_n+b') \\ L_T(h_0,h_1,h_2) &= \log(1+\exp({h_0}^{\mathrm{T}}h_2-{h_0}^{\mathrm{T}}h_1)) \end{aligned} \tag{1}$$

$$\theta = \underset{W,W',b,b'}{\arg\min} \sum_{(x_0,x_1,x_2)\in T} \sum_{n=0}^{2} L_R(y_n,x_n) + \alpha L_T(h_0,h_1,h_2),$$

where $T \subset X^3$, such that x_0 and x_1 are in the same category/similar categories and x_0 and x_2 are in different categories. The h in Eq.1 satisfies the property, $x = 0 \Rightarrow h = 0$. This means that an article that has no available information is not similar to any other article. The notation, $L_T(\cdot,\cdot,\cdot)$ is a penalty function for article similarity to correspond to categorical similarity, and α is a hyperparameter for balancing. Figure 2 provides an overview of this method.

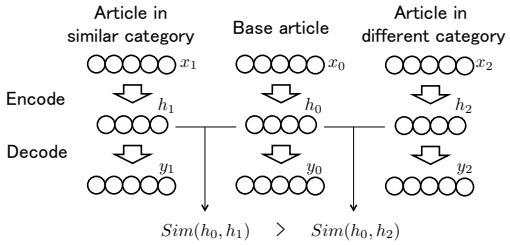

Figure 2: Encoder for triplets of articles

We use the elementwise sigmoid function, $\sigma(x)_i = 1/(1+\exp(-x_i))$, as $f(\cdot)$, elementwise cross entropy as $L_R(\cdot,\cdot)$, and masking noise as $q(\cdot|\cdot)$. We train the model, $\theta = \{W, W', b, b'\}$, by using mini-batch stochastic gradient descent (SGD).

We construct $\tilde{x}$ in the application phase by using constant decay, instead of stochastic corruption in the training phase, as:

$$\begin{aligned} \tilde{x} &= (1-p)x \\ h &= f(W\tilde{x}+b)-f(b), \end{aligned}$$

where p is the corruption rate in the training phase. Thus, h is uniquely determined at the time of application. Multiplying $1-p$ has the effect of equalizing the input distribution to each neuron in the middle layer between learning with masking noise and that without the application of this noise.

We use the h generated above in three applications as the representation of the article: (i) to input the user-state function described in Section 4, (ii) to measure the relevance of the user and the article in matching, and (iii) to measure the similarity between articles in de-duplication.

4 USER REPRESENTATIONS

This section describes several variations of the method to calculate user preferences from the browsing history of the user. First, we formulate our problem and a simple word-based baseline method and discuss the issues that they have. We then describe some methods of using distributed representations of articles, as was explained in the previous section.

4.1 Notation

Let A be the entire set of articles. Representation of element $a \in A$ depends on the method. The a is a sparse vector in the word-based method described in Section 4.2, and each element of a vector corresponds to each word in the vocabulary (i.e., x in Section 3). However, a is a distributed representation vector of the article (i.e., h in Section 3) in the method using distributed representations described in Sections 4.3 and 4.4.

Browse means that the user visits the uniform resource locator (URL) of the page of an article. Let $\{a_t^u \in A\}_{t=1,\cdots,T_u}$ be the browsing history of user $u \in U$.

Session means that the user visits our recommendation service and clicks one of the articles in the recommended list.

When u clicks an article in our recommendation service (a session occurs), he/she will immediately visit the URL of the clicked article (a browse occurs). Thus, there is never more than one session between browses a_t^u and a_{t+1}^u; therefore, this session is referred to as s_t^u. However, u can visit the URL of an article without our service, e.g., by using a Web search. Therefore, s_t^u does not always exist.

Since a session corresponds to the list presented to u, we express a session, s_t^u, by a list of articles $\{s_{t,p}^u \in A\}_{p \in P}$. The notation, $P \subseteq \mathbb{N}$, is the set of positions of the recommended list that is actually displayed on the screen in this session. Let $P_+ \subseteq P$ be the clicked positions and $P_- = P \setminus P_+$ be non-clicked positions. Although P, P_+, and P_- depend on u and t, we omit these subscripts to simplify the notation. Figure 3 outlines the relationships between these notations.

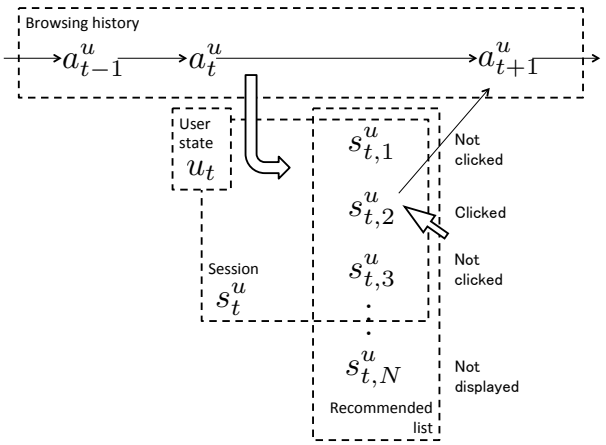

Figure 3: Browsing history and session

Let u_t be the user state depending on $a_1^u, \cdots, a_t^u$, i.e., u_t represents the preference of u immediately after browsing a_t^u. Let $R(u_t, a)$ be the relevance between the user state, u_t, and the article, a, which represents the strength of u's interest in a in time t. Our main objective is to constitute *user-state function* $F(\cdot, \ldots, \cdot)$ and *relevance function* $R(\cdot, \cdot)$ that satisfy the property:

$$u_t = F(a_1^u, \ldots, a_t^u)$$
$$\forall s_t^u \, \forall p_+ \in P_+ \, \forall p_- \in P_- \, R(u_t, s_{t,p_+}^u) > R(u_t, s_{t,p_-}^u). \quad (2)$$

When considering the constrained response time of a real news distribution system with large traffic volumes, $R(\cdot, \cdot)$ must be a simple function that can be quickly calculated. Because candidate articles are frequently replaced, it is impossible to pre-calculate the relevance score, $R(u_t, a)$, for all users $\{u_t | u \in U\}$ and all articles $\{a \in A\}$. Thus, it is necessary to calculate this in a very short time until the recommended list from visiting our service page is displayed. However, we have sufficient time to calculate the user state function, $F(\cdot, \ldots, \cdot)$, until the next session occurs from browsing some article pages.

We restrict relevance function $R(\cdot, \cdot)$ to a simple inner-product relation, $R(u_t, a) = u_t^{\mathrm{T}} a$, for these reasons and only optimize the user state function, $F(\cdot, \ldots, \cdot)$, to minimize the objective:

$$\sum_{s_t^u} \sum_{\substack{p_+ \in P_+ \\ p_- \in P_-}} -\frac{\log(\sigma(R(u_t, s_{t,p_+}^u) - R(u_t, s_{t,p_-}^u)))}{|P_+||P_-|}, \quad (3)$$

where $\sigma(\cdot)$ is the logistic sigmoid function. Equation 4.1 is a relaxation of Eq. 2. As there is a bias, in practice, in the probability of clicks depending on the displayed position when articles are vertically arranged, we use the following objective including the bias term, $B(\cdot, \cdot)$, to correct such effects. Although $B(\cdot, \cdot)$ is a parameter to be determined by learning, its description has been omitted below since it is a common term in the models that follow.

$$\sum_{s_t^u} \sum_{\substack{p_+ \in P_+ \\ p_- \in P_-}} -\frac{\log(\sigma(R(u_t, s_{t,p_+}^u) - R(u_t, s_{t,p_-}^u) + B(p_+, p_-)))}{|P_+||P_-|}.$$

4.2 Word-based Model

We formulate the word-based baseline model introduced in Section 1 as a baseline implementation on the basis of the notations in the previous section.

Recall the three steps in the baseline implementation.

- An article is represented by a collection of words included in its text,
- A user is represented by a collection of words included in articles he/she has browsed, and
- The relevance between the user and article is expressed by a linear function on the co-occurrences of words between them.

This model can be regarded as a special case of the notation in the previous section, if article representation a and user function F are defined as:

$$V : \text{Vocabulary set}$$
$$a, u_t \in \{0, 1\}^V$$
$$(a)_v = \begin{cases} 1 & \text{(if the article contains the word } v) \\ 0 & \text{(otherwise)} \end{cases}$$
$$(u_t)_v = (F(a_1^u, \ldots, a_t^u))_v = \alpha_v \max_{1 \le t' \le t} (a_{t'}^u)_v, \quad (4)$$

where $(x)_v$ is the v-th element of x. Then, the relevance function becomes a simple linear model with parameters $\{\alpha_v\}$ as:

$$\begin{aligned} R(u_t, a) &= u_t^{\mathrm{T}} a \\ &= \sum_{v \in V} (u_t)_v (a)_v \\ &= \sum_{v \in V} \alpha_v 1_{v \in u_t \cap a}. \end{aligned}$$

There are two major issues with this model, as was explained in Section 1.

The first is the sparseness of the representation. The $R(u_t, a)$ in this model increases if and only if u_t and a contain the same words. Thus, even if a user is interested in similar articles, the relevance scores may greatly differ depending on whether they have been written using the same words as the articles in his/her browsing histories.

The second issue is intensity. Because the browsing history is regarded as a set, the information about the browsing order and frequency are lost. Thus, the model is very vulnerable to noise that is mixed into the browsing history.

We describe other models by using distributed representations by taking into account these issues in the subsections below.

4.3 Decaying Model

We introduce a simple model with distributed representations to solve these issues. Two changes in this model from that of the baseline are:

- It uses the distributed representation constructed in Section 3 as the representation vector, a, of an article, instead of a bag-of-words (BoW) representation (which addresses the first issue).
- It uses the weighted average to aggregate browsing histories, instead of the maximum value. More specifically, we increase the weights for recent browses and reduce the weights for the previous days' browses (which addresses the second issue).

In summary, u_t can be expressed as:

$$u_t = \alpha \odot \frac{1}{\sum_{1 \le t' \le t} \beta^{t-t'}} \sum_{1 \le t' \le t} \beta^{t-t'} a^u_{t'},$$

where α is a parameter vector that has the same dimension as a^u_t, $\odot$ is the elementwise multiplication of two vectors, and $0 < \beta \le 1$ is a scalar value that is a hyperparameter that represents the strength of time decay. If β is 1, aggregation is the simple average, so that the browsing order is not taken into consideration. Training parameters are only α in this model, which are similar to those in the baseline model.

4.4 Recurrent Models

4.4.1 Simple Recurrent Unit. Although the decaying model in the previous subsection took into account issues with the word-based model, it had limitations such as being linear with respect to frequency and that the forgetting effect was limited to exponential decay with hyperparameters.

More generally, u_t should be determined by a previous state, u_{t-1}, and previous browse a^u_t as:

$$u_t = f(a^u_t, u_{t-1}).$$

Therefore, we try to learn this function by using an RNN. A simple RNN is formulated by:

$$u_t = \phi(W^{in} a^u_t + W^{out} u_{t-1} + b),$$

where $\phi(\cdot)$ is the activation function; therefore, we subsequently use hyperbolic tangent function $\tanh(\cdot)$. Training parameters are square matrices W^{in}, W^{out}, bias vector b, and initial state vector u_0 in this model, where u_0 is a common initial value that does not depend on u.

We learn these parameters by end-to-end mini-batch SGD with the objective function in Eq. 4.1. However, when the input sequence is too long, simple RNN makes it too difficult to learn this due to gradient vanishing and explosion problems [5]. Additional structures that are attached to the hidden layer are able to alleviate these problems in such cases.

The following subsections introduce two models that use such structures.

4.4.2 Long-short Term Memory Unit. Long-short term memory (LSTM) [6] is a well-known structure for vanishing and exploding gradients [5]. We formulate an LSTM-based model as:

$$\begin{aligned} gi_t &= \sigma(W^{in}_{gi} a^u_t + W^{out}_{gi} u_{t-1} + W^{mem}_{gi} h^u_{t-1} + b_{gi}) \\ gf_t &= \sigma(W^{in}_{gf} a^u_t + W^{out}_{gf} u_{t-1} + W^{mem}_{gf} h^u_{t-1} + b_{gf}) \\ enc_t &= \phi(W^{in}_{enc} a^u_t + W^{out}_{enc} u_{t-1} + b_{enc}) \quad (5) \\ h^u_t &= gi_t \odot enc_t + gf_t \odot h^u_{t-1} \quad (6) \\ go_t &= \sigma(W^{in}_{go} a^u_t + W^{out}_{go} u_{t-1} + W^{mem}_{go} h^u_t + b_{go}) \\ dec_t &= \phi(W^{mem}_{dec} h^u_t + b_{dec}) \quad (7) \\ u_t &= go_t \odot dec_t, \quad (8) \end{aligned}$$

where $\sigma(\cdot)$ is the elementwise logistic sigmoid function, and h^u_t is a hidden memory state. Figure 4 has a network image of the structure of the LSTM-based model.

The center flows are the main flows from input (browsed article) to output (user state). The input, a^u_t, is encoded from article vector space to hidden space (Eq. 5), merged into the previous hidden state (Eq. 6), and decoded to the article vector space (Eq. 7, Eq. 8) as the user state.

In addition, this unit has three gates, called input gate (gi_t), forget gate (gf_t), and output gate (go_t). We assumed each gate would conceptually play the following roles in this case. The input gate filters unnecessary inputs to construct a user state, such as that caused by sudden interest. The forget gate represents the decline in interest by the user. It can represent a more complex forgetting effect than the exponential decay that is used in the decaying model. The output gate filters components that should not be focused on in the next session.

Training parameters are weight matrices $W^{\cdot}_{\cdot}$, bias vectors $b_{\cdot}$ and initial state vectors u_0 and h^u_0 in this model, where u_0 and h^u_0 are common initial values that do not depend on u.

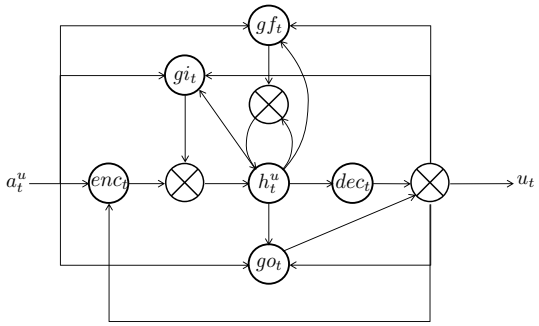

Figure 4: LSTM-based model

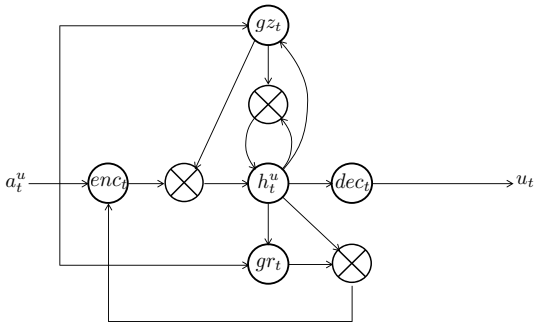

Figure 5: GRU-based model

4.4.3 Gated Recurrent Unit. Gated recurrent unit (GRU) [1] is another structure to avoid gradient vanishing and explosion problems [5]. We formulate a GRU-based model as:

$$
\begin{aligned}
gz_t &= \sigma(W_{gz}^{in} a_t^u + W_{gz}^{mem} h^{t-1} + b_{gz}) \\
gr_t &= \sigma(W_{gr}^{in} a_t^u + W_{gr}^{mem} h^{t-1} + b_{gr}) \\
enc_t &= \phi(W_{enc}^{in} a_t^u + W_{enc}^{out}(gr_t \odot h^{t-1}) + b_{enc}) \\
h_t^u &= gz_t \odot enc_t + (1 - gz_t) \odot h_{t-1}^u \qquad (9) \\
dec_t &= \phi(W_{dec}^{mem} h_t^u + b_{dec}) \qquad (10) \\
u_t &= dec_t .
\end{aligned}
$$

We describe these formulations using symbols that correspond to those of the LSTM-based model as much as possible. More precisely, this model is constructed using one GRU layer and one fully connected layer because Eq. 10 is not contained in the original GRU configuration. Figure 5 outlines a network image of the GRU-based model structure.

Except for the omission of some arrows, this structure is similar to that of the LSTM-based model. However, there is an important difference between Eqs. 6 and 9. The gz_t gate in this model plays the role of two gates, i.e., gi_t and gf_t in the LSTM-based model. As a result, the following difference in the upper limit of norm $||h_t^u||_\infty$ occurs.

$$
\sup_u ||h_t^u||_\infty = \begin{cases} ||h_0^u||_\infty + t \sup_x |\phi(x)| & \text{in LSTM} \quad (11) \\ \max(||h_0^u||_\infty, \sup_x |\phi(x)|) & \text{in GRU} . \quad (12) \end{cases}
$$

Equation 11 can be a large value for a very long input sequence; however, Eq. 12 never exceeds the constant. Therefore, we think the GRU-based model has a higher aptitude to solve the gradient explosion problem than the LSTM-based model.

The LSTM-based model occasionally failed in training due to gradient explosion when we did not use gradient clipping [13] in the experiments that are described in the next section. However, the GRU-based model did not cause gradient explosion without any supplementary processing.

5 OFFLINE EXPERIMENTS

This section discusses the effectiveness of the distributed representation-based method with the models in the previous section by offline evaluation using past serving logs. We compared the three word-based models that were variants of the model introduced in Section 4.2 and five distributed representation-based models introduced in Sections 4.3 and 4.4. These models are summarized in Table 1.

5.1 Training Dataset

First, we sampled approximately 12 million users who had clicked at least one article from the service logs of Yahoo! JAPAN's homepage on smartphones between January and September 2016. We extracted logs for each user over a two-week period chosen at random to include at least one click. This method of extraction was used to mitigate the impact of epidemic articles within a specific period.

As a result, there were about 166 million sessions, one billion browses, and two million unique articles in the training data. We also created another dataset for the same period and used it as a validation dataset to optimize the hyperparameters.

5.2 Test Dataset

We sampled 500,000 sessions, in which users clicked articles above position 20 on October 2016. We extracted browse logs in the previous two weeks for each session. We used the article data from positions 1 to 20 for evaluation regardless of whether they were actually displayed on the screen. This was based on our observation with timeline-based user-interfaces. Users scrolled from top to bottom and tended to leave our service when they clicked one article. That is, if we only used data actually displayed for evaluation, the method of arranging the actual displayed order in reverse was evaluated as being disproportionately better.

5.3 Offline Metrics

We evaluated the rankings provided by each model by using three popular metrics, i.e., the area under the ROC curve (AUC), mean reciprocal rank (MRR), and normalized discounted cumulative gain (nDCG), which regarded clicks as positive labels.

Let S be the set of sessions for the evaluation, $c_{s,i}$ be *one* when the article at position i is clicked, and zero otherwise. Each metric is formulated as:

$$
\begin{aligned}
\text{AUC} &= \frac{1}{|S|} \sum_{s \in S} \frac{|\{(i,j) | i < j, c_{s,i} = 1, c_{s,j} = 0\}|}{|\{i | c_{s,i} = 1\}||\{j | c_{s,j} = 0\}|} \\
\text{MRR} &= \frac{1}{|S|} \sum_{s \in S} \frac{1}{\min_{c_{s,i}=1} i} \\
\text{nDCG} &= \frac{1}{|S|} \sum_{s \in S} \frac{\sum_i c_{s,i} / \log_2(i+1)}{\max_\pi \sum_i c_{s,\pi(i)} / \log_2(\pi(i)+1)},
\end{aligned}
$$

where π is an arbitrary permutation of positions.

The AUC is the metric directly related to the objective of training (see Eq.2). The MRR and nDCG are popular ranking metrics; the former focuses on the first occurrence of positive instances, and the latter evaluates ranks for all the positive instances. Each metric is a value between zero and one, which is calculated as the average score for each session; i.e., the larger the better.

5.4 Models and Training

We evaluated the eight models listed in Table 1.

Table 1: Model descriptions

Name	Description	Section
BoW	The simplest word-based model	4.2
BoW-Ave	Word-based model that uses average function instead of max in Eq.4	4.2 $+\alpha$
BoW-Dec	Word-based model that uses decayed average function with $\beta = 0.9$ similar to that introduced in Section 4.3	4.2 $+\alpha$
Average	Decaying model with $\beta = 1$ (no decaying)	4.3
Decay	Decaying model with $\beta = 0.9$	4.3
RNN	Recurrent model using simple RNN unit	4.4.1
LSTM	Recurrent model using LSTM-based unit	4.4.2
GRU	Recurrent model using GRU-based unit	4.4.3

As was described in Section 4.2, word-based models can be regarded as simple linear logistic regressions for pairwise data with sparse feature vectors, like the ranking support vector machine (SVM) [7]. Thus, we used LIBLINEAR [4] for training L2-regularized logistic regression (solver type 0).

We used mini-batch SGD with RMS-prop to adjust the learning rate for the other models. The mini-batch size was 20 and the initial learning rate was 0.005. The numbers of dimensions of the distributed representations of article a and user state u_t were 500. The number of dimensions of internal state h_t^u was 200 in *LSTM* and *GRU*. These parameters were determined by using the development dataset.

We used the gradient clipping technique [13] to avoid gradient explosion in *RNN* and *LSTM*. *GRU* did not cause gradient explosion when this technique was not used in these experiments.

5.5 Experimental results

Table 2 lists all metrics used in the experiments. We split the test dataset into ten sub-datasets and calculated each metric per sub-dataset. The values in Table 2 are the averages for each metric of the sub-datasets and each metric's 99% confidence intervals that were estimated based on the t-distribution.

GRU had the best score for all metrics with a sufficient margin, and it yielded a significant difference against all other models according to a paired t-test with $p < 0.01$.

Because *Average* exhibited a significant improvement from *BoW-Ave*, distributed representations of articles should work better than BoW representations.

Decay and *BoW-Dec* were worse than *Average* and *BoW-Ave*. This suggests that browsing-order information cannot be expressed with simple attenuation. *RNN* also exhibited a slight improvement on AUC against *Average*. However, *LSTM* and *GRU* were significantly better than *Average*. We believe this was because it was able to express more complex relations for the order of browsing sequences by using the gate structures in these models.

Table 2: Results from offline experiments. Values indicate average of metrics and 99% confidence intervals in ten split test sets.

	AUC	MRR	nDCG
BoW	0.582 ± 0.003	0.300 ± 0.003	0.446 ± 0.002
BoW-Ave	0.579 ± 0.004	0.310 ± 0.003	0.452 ± 0.002
BoW-Dec	0.560 ± 0.004	0.297 ± 0.004	0.442 ± 0.003
Average	0.608 ± 0.003	0.313 ± 0.003	0.457 ± 0.002
Decay	0.596 ± 0.003	0.302 ± 0.002	0.449 ± 0.001
RNN	0.612 ± 0.004	0.309 ± 0.004	0.455 ± 0.003
LSTM	0.648 ± 0.004	0.344 ± 0.004	0.481 ± 0.003
GRU	**0.652 ± 0.003**	**0.347 ± 0.004**	**0.484 ± 0.003**

6 DEPLOYMENT

We began using *GRU* on December 2016 for the main traffic in our service (which we denoted the Proposed bucket). However, we continued to apply the conventional *BoW* model to 1 % of users to enable comparisons of performance (which we denoted the Control bucket). This section reports the results obtained from comparisons after deployment.

6.1 Settings

Our system presents articles to users who visit our service page according to a three step procedure.

- The user state, u_t, of each model should be calculated from the browsing history in advance.
- When user u visits our service, we calculate the relevance scores, $R(u_t, a) = u_t^{\mathrm{T}} a$, for all articles a that were newly published within a certain period of time.
- We present the articles to the user in order from the highest relevance score by applying de-duplication.

The representation of u_t and a for relevance scores depends on the bucket. We applied de-duplication by using the distributed representations described in Section 2 to both Control and Proposed buckets. As the effects of de-duplication on users are not discussed here, refer to our past paper [12] for details on an experiment on these effects.

We can identify the user from browser cookies in addition to the ID for login. Therefore, we can extract some histories for most users including those who are not logged in. However, there are some users who have no history such as new users or who have been privately browsing. Although we can provide articles to them with other methods by using popularity, diversity, and freshness, instead of relevance scores, all results on them have been excluded from the following reports.

6.2 Online Metrics

The list below represents the four online metrics we used.

Sessions The average number of times one user utilized our service per day.

Table 3: Average metric lift rates in 7th week and those by user segments.

Metric	ALL	Heavy	Medium	Light
Sessions	+2.3%	+1.1%	+1.0%	+1.8%
Duration	+7.8%	+4.9%	+13.3%	+17.4%
Clicks	+19.1%	+14.3%	+26.3%	+42.3%
CTR	+23.0%	+18.7%	+29.8%	+45.1%

Duration The average time (in seconds) that the user spent with our service per session. It was the total time the user took looking at the recommendation list and the time it took him/her to read the article after clicking on it.

Clicks The average number of clicks per session (which corresponded to $|P_+|$ in Section 4).

Click through rate (CTR) Clicks/number of displayed articles. These were decreased if article retrieval became inefficient and users spent more time exploring the recommendation list.

A session in this section means the collection of accesses by a user. It is regarded as another session if there is a gap of more than 10 min between accesses.

Our most important metric is the total duration per user, which is the product of *Sessions* and *Duration*.

6.3 Experimental Results

The results obtained from comparison are summarized in Figure 6 and Table 3.

Figure 6 plots the daily transition in the lift rate of each metric (i.e., that with the Proposed and metric with Control). All metrics improved significantly with the Proposed bucket. Duration, Clicks, and CTR indicated a certain rate of improvement from the first day the new model was applied. However, Sessions demonstrated a relatively small rate of improvement on the first day but gradually improved a great deal. This meant that a good recommendation model first increases clicks, and multiple click experiences encourage users to gradually use the service more frequently.

Table 3 summarizes the average metric lift rates in the seventh week and those by user segments split according to the frequency of visits. The definitions for segments are three fold. The user composition ratios are roughly $Heavy : Medium : Light = 3 : 2 : 1$.

- Heavy: Users who have visited for more than five days during the previous week.
- Medium: Users who have visited for between two and five days during the previous week.
- Light: Users who have visited for less than two days during the previous week.

Improvements in the metrics were confirmed for all segments. Light users demonstrated particularly large improvement rates. Therefore, we derived the following hypotheses. Users who had little history were strongly influenced by feature sparsity if we used BoW representations. Also, some noisy browsing caused by sudden interest seriously affected the accuracy of recommendations. Our method with the GRU-based model might successfully prevent these problems with distributed representations and the gate structures of GRU. This is why the Proposed bucket worked better than the Control bucket with the word-based model, especially for light users.

Table 4: Average daily percentage of user composition in 7th week for both buckets.

Bucket	ALL (%)	Heavy (%)	Medium (%)	Light (%)
Control	100.0	50.0	33.9	16.1
Proposed	100.0	51.4	33.6	15.0
0th week (ref.)	100.0	49.7	34.3	16.0

In addition to the improvement in metrics within each segment, we also observed a shift in users from Light to Medium, and Medium to Heavy, as listed in Table 4. This is why the overall rate of increase in Sessions was higher than the increase in each segment in Table 3.

6.4 Deployment challenges to large-scale deep-learning-based services

It is very important to keep updating models with the latest data when applying machine learning to actual news distribution systems. In fact, the Control bucket described in the previous section uses the latest article data and session data every three hours to update the model. The Proposed bucket using the GRU-based model, on the other hand, could not update the model as frequently due to four reasons.

- It takes a long time to learn the model. In fact, the model actually used in main traffic is calculated over a week using GPU.
- If we update the article-representation model (discussed in Section 3), we have to re-calculate representations for all available articles and re-index to the article search engine.
- If we update the user-representation model (discussed in Section 4), we have to re-calculate the user state from the first browse in the past. In practice, there is no choice but to give up on re-calculating old histories over a certain period of time.
- The data store holding the user representations and the search index holding the article representations must be synchronously updated.

Therefore, we prepared two identical systems and switched them alternately to update the model once every two weeks. Of course, adding fresh articles and updating user states by users' browsing are executed more frequently because they require fewer calculations than those for updating models.

As word-based models in our experience have responded sensitively to buzz-words, they deteriorate as soon as a few days after stops have been updated; however, the distributed representation-based model maintains sufficient accuracy at this frequency of model updates. When we left the GRU-based model for about three months, it had deteriorated to the same accuracy as the word-based model with updates every three hours in experiments without model updates.

7 RELATED WORK

This section presents related work to our proposed approach.

We generated distributed representations of articles by using a denoising autoencoder with weak supervision, as described was

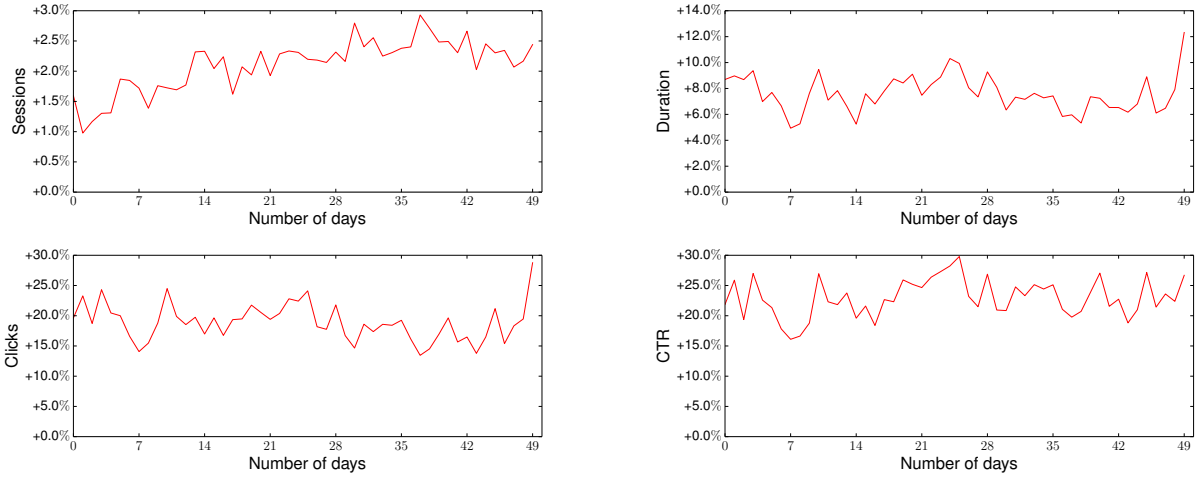

Figure 6: Transition in lift rate for metrics.

in Section 3 and our previous paper [12]. There have been many studies on generating distributed representations of sentences. The conventional denoising auto-encoder [19], its variant [20], and Paragraph Vector [11] are well-known methods of unsupervised learning. Representations of articles generated with these methods may be used for our problem setting. While there have been reports [11, 16] that the representations generated with these methods provide helpful features for sentiment classification tasks, it has not been clarified whether these representations would be suitable for our purposes because these methods are unsupervised approaches.

Of course, fully supervised methods can be applied if we obtain a sufficient amount of human-annotated data for article pairs, but this is too costly since news articles and the proper nouns they contain change over time. Therefore, our proposed method uses a weakly supervised method with an objective function that includes loss terms that correspond to the similarities between categories of articles, as well as reconstruction terms for the denoising autoencoder of words. As these categories, such as politics, sports, and entertainment, are provided by news publishers, they are easy to obtain. In addition, as distributed representations, which are generated by optimizing objective functions that consist of two terms, are expected to represent reasonable granularity of similarity, they inherit the properties of both words and categories.

There have been some studies [18, 21] on obtaining the representations of user interests from the sequence of users' behaviors on the Web. Zhang et al. [21] proposed a framework based on an RNN for click prediction of sponsored search advertising. Tagami et al. [18] applied Paragraph Vector [11] to users' Web browsing sequences to obtain common features from user-related prediction tasks. RNN and its variants have recently been widely used to obtain sequential data in various research fields, such as speech recognition [15], machine translation [17], and image captioning [9]. Learning with standard RNNs often suffers from vanishing and exploding gradient problems. Thus, some architectures such as LSTM [6] and GRU [1] have been employed to overcome these problems. Jozefowicz et al. [8] empirically evaluated various RNN architectures and reported that GRU outperformed LSTM on almost all tasks they evaluated. Our experimental results presented in Section 5.5 also showed that the GRU-based model produced better results than the LSTM-based model.

8 CONCLUSION

This paper proposed an embedding-based method to use distributed representations in a three step end-to-end manner: (i) start with distributed representations of articles based on a variant of a denoising autoencoder, (ii) generate user representations by using an RNN with browsing histories as input sequences, and (iii) match and list articles for individual users based on inner-product operations by considering system performance. We found that our method was effective even in a real news distribution system with large-scale traffic because it was designed with an awareness of implementation.

The method we propose has already been fully incorporated in all the traffic of Yahoo! JAPAN's homepage on smartphones, and it recommends various articles to millions of users every day. We plan to further improve our recommendation service continuously in the future and apply this approach to other domains such as advertisements.

REFERENCES

[1] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

[2] Henriette Cramer. 2015. Effects of Ad Quality & Content-Relevance on Perceived Content Quality. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*.

[3] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google News Personalization: Scalable Online Collaborative Filtering. In *Proceedings of the 16th International Conference on World Wide Web*.

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

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

[6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* (1997).

[7] Thorsten Joachims. 2002. Optimizing Search Engines Using Clickthrough Data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[8] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning*.

[9] Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[10] Mounia Lalmas, Janette Lehmann, Guy Shaked, Fabrizio Silvestri, and Gabriele Tolomei. 2015. Promoting Positive Post-Click Experience for In-Stream Yahoo Gemini Users. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[11] Quoc Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of The 31st International Conference on Machine Learning*.

[12] Shumpei Okura, Yukihiro Tagami, and Akira Tajima. 2016. Article Deduplication Using Distributed Representations. In *Proceedings of the 25th International Conference Companion on World Wide Web*.

[13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2014. On the difficulty of training recurrent neural networks. In *Proceedings of The 30th International Conference on Machine Learning*.

[14] Jay Adams Paul Covington and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA.

[15] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Hasim Sak. 2015. Convolutional, long short-term memory, fully connected deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*.

[16] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

[17] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems 27*.

[18] Yukihiro Tagami, Hayato Kobayashi, Shingo Ono, and Akira Tajima. 2015. Modeling User Activities on the Web Using Paragraph Vector. In *Proceedings of the 24th International Conference on World Wide Web*.

[19] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*.

[20] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research* (2010).

[21] Yuyu Zhang, Hanjun Dai, Chang Xu, Jun Feng, Taifeng Wang, Jiang Bian, Bin Wang, and Tie-Yan Liu. 2014. Sequential click prediction for sponsored search with recurrent neural networks. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*.

[22] Erheng Zhong, Nathan Liu, Yue Shi, and Suju Rajan. 2015. Building Discriminative User Profiles for Large-scale Content Recommendation. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.