Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Word Prediction Model using an elaborated RNN integrated with N-gram for efficient Text Input

Yukino Ikegami
IO Inc., Tokyo, Japan
Setsuo Tsuruta
Tokyo Denki University
Andrea Kutics
International Christian University
Ernesto Damiani
Khalifa University of Science and Technology
Rainer Knauf (rainer.knauf@tu-ilmenau.de)
Ilmenau University of Technology

Research Article

Keywords: Input Method Editor, word prediction, hybrid language model, client-server model, Recurrent Neural Networks, Back Propagation Through Time

Posted Date: August 14th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3246800/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Manuscript title:
Word Prediction Model using an elaborated RNN integrated with N-gram for efficient Text Input

Authors:
- **Yukino Ikegami**, IO Inc., Tokyo, Japan, yknikgm@gmail.com
- **Setsuo Tsuruta**, School of Information Environment, Tokyo Denki University, Tokyo, Japan, tsuruta@mail.dendai.ac.jp
- **Andrea Kutics**, International Christian University, Tokyo, Japan, matz@icu.ac.jp, ORCID 0000-0002-9313-1119
- **Ernesto Damiani**, Cyber-Physical Systems Center, Khalifa University, Abu Dhabi, ernesto.damiani@ku.ac.ae, ORCID 0000-0002-9557-6496
- **Rainer Knauf**, Ilmenau University of Technology, Ilmenau, Germany, rainer.knauf@tu-ilmenau.de, ORCID 0000-0001-8795-6360

Rainer Knauf is the corresponding author.

Abstract:
Smartphone users are beyond two billion worldwide. Heavy users of the texting application rely on input prediction to reduce typing effort. In languages based on the Roman alphabet, many techniques are available. However, Japanese text is based on multiple character sets such as Kanji (Chinese-like word symbols), Hiragana and Katakana syllable sets. For its time/labor intensive input, next word prediction is crucial. It is still an open challenge. To tackle this, a hybrid language model is proposed. It integrates a Recurrent Neural Network (RNN) with an n-gram model. RNNs are powerful models for learning long sequences for next word prediction. N-gram models are best at current word completion. Our RNN language model (RNN-LM) predicts the next words. According the "price" of the performance gain paid by a higher time complexity, our model best deploys on a client-server architecture. Heavily-loaded RNN-LM deploys on the server while the n-gram model on the client. Our RNN-LM consists of an input layer equipped with word embedding, an output layer, and hidden layers connected with LSTMs (Long Short-Term Memories). Training is done via BPTT (Back Propagation Through Time). For robust training, BPTT is elaborated by learning rate refinement and gradient norm scaling. To avoid overfitting, the dropout technique is applied except for LSTM. Our novel model is compact (2 LSTMs, 650 units per layer), indeed. Due to synergetic elaboration, it shows 10% lower perplexity than Zaremba's excellent conventional models in our Japanese text prediction experiment. Our model has been incorporated into IME (Input Method Editor) we call Flick. On the Japanese text input experiment, Flick outperforms Mozc (Google Japanese Input) by 16% in time and 34% in the number of keystrokes.

Keywords:
Input Method Editor, word prediction, hybrid language model, client-server model, Recurrent Neural Networks, Back Propagation Through Time

Conflict of Interest:
The authors declare that they have no conflict of interest.

Funding:
This work was supported
- by JSPS Fund KAKENHI, grant number 15K00349, and
- by the European Commission Fund HORIZON under the project TOREADOR, grant number H2020-688797.

Word Prediction Model using an elaborated RNN integrated with N-gram for efficient Text Input

Abstract— **Smartphone users are beyond two billion worldwide. Heavy users of the texting application rely on input prediction to reduce typing effort. In languages based on the Roman alphabet, many techniques are available. However, Japanese text is based on multiple character sets such as Kanji, Hiragana and Katakana. For its time intensive input, next word prediction an open challenge. To tackle this, a hybrid language model is proposed. It integrates a Recurrent Neural Network (RNN) with an n-gram model. RNNs are powerful models for learning long sequences for next word prediction. N-gram models are best at current word completion. Our RNN language model predicts next words. According the "price" of the performance gain paid by a higher time complexity, our model best deploys on a client-server architecture. Heavily-loaded RNN-LM deploys on the server while the n-gram model on the client. It consists of an input layer equipped with word embedding, an output layer, and hidden layers connected with LSTMs (Long Short-Term Memories). Training is done via BPTT (Back Propagation Through Time). For robust training, BPTT is elaborated by learning rate refinement and gradient norm scaling. To avoid overfitting, the dropout technique is applied except for LSTM. Due to synergetic elaboration, it shows 10% lower perplexity than Zaremba's excellent conventional models in our experiment. Our model has been incorporated into IME (Input Method Editor) we call Flick. In our experiment, Flick outperforms Mozc (Google Japanese Input) by 16% in time and 34% in the number of key strokes.**

Index Terms—**Input Method Editor, word prediction, hybrid language model, client-server model, Recurrent Neural Networks, Back Propagation Through Time**

I. Introduction

IN today's world, billions of users rely on smartphones and mobile communication technologies in their personal and work time. In Japan, the Japanese Ministry of Internal Affairs and Communications, 2017, has reported the number of smartphone users reached 60.48 million [25]. The widespread use of smartphones for text-based messaging has increased on-the-job flexibility, allowing people to decide when and where to exchange messages. It can also impact on productivity. On smartphones, a software keyboard is used to input texts. Software keyboards and their providing services such as next word prediction, shape the speed and accuracy with which smartphone users communicate.

As Komachi and Kida [32] reported, inputting texts via a software keyboard requires more time than by traditional keyboards. This negative impact on productivity is partly language-dependent. Recent findings suggest that inputting Japanese text requires an even larger amount of typing than other languages.

[46] shows there are 2.71 billion characters in 34.2 million English Twitter posts (tweets) and there are 625 million characters in 13.8 million Japanese tweets. In other words, the average length of English is 79.23 characters, while that of Japanese tweets is 45.28 characters. These results seem to suggest inputting Japanese is easier than English.

However, attention should be paid to the fact that inputting most Japanese characters requires typing multiple keys, since there is a large set of characters that represent full syllables, as opposed to the other writing systems, which approximately have one letter per phoneme. For instance, a Kanji character "東" (east) requires typing "higashi" (i.e. 7 keys).

More concretely, we calculated Japanese keystrokes per character (KSPC). KSPC is proposed in [37]. To measure the average Japanese KSPC, we used the core data of the Balanced Corpus of Contemporary Written Japanese [38]. This corpus contains equal amounts of text data from different domains (poems, magazine articles, blogs, government documents, and others). This result showed the average number of keystrokes to input a single Japanese character is 2.3, i.e. the average number of keystrokes to input Japanese tweets is (2.3 keystrokes * 45.28 character =) 104.14. It exceeds that one (79.23 keystrokes) of English tweets. According to [51], the average number of characters inputted by ordinary people is 5.4 per second. Based on the above estimate of the average size of Japanese tweets, we get 104.14 keystrokes / 5.4 seconds = 19.29 seconds on average for each Japanese tweet, while (79.23 keystrokes / 5.4 seconds =) 14.67 seconds are enough for an English tweet. Japanese requires around 25% larger amount of typing than English.

In detail, Japanese relies on multiple character sets such as Kanji (Chinese word symbols) as well as the two syllable sets Hiragana and Katakana. Katakana and Hiragana have 46 basic letters each and some modified forms. Totally, the three sets include over 6,000 characters, which obviously cannot be assigned to the keys of a physical keyboard and even less on a mobile device's software keyboard. Thus, Japanese is usually input using special software tools called Input Method Editors (IMEs). To reduce input time and key strokes, Japanese IMEs provide character sets conversion (e.g. alphabet-to-Kana, Kana-to-Kanji) and current/next word prediction. IMEs are used on personal computers as well as smartphones. In case of smartphones, IMEs are used via software keyboards instead of physical keyboards. Many software keyboards used on smartphones offer a predictive text input function. Such functions predict the current word as well as the next word based on the characters typed so far. To decrease the high (25% more than English) input cost of Japanese, it is very important to predict the next words. Inputting Kanji has remarkably high cost since Kanji consists of one or more syllables per character and has many homophone characters (e.g. over 50 Kanji

characters for “kai”: 解, 会, 回, 貝, 快, 戒, 階, …, 介, 櫂). It is especially important to decrease, by word prediction, the many keystrokes needed for Kanji conversion (e.g. 5-10 keystrokes to obtain “櫂” for “kai” even having a window of 5-10 character width). Our challenge is improving current and next word prediction to decrease the high input cost of Japanese, especially Kanji.

Using prefixes or previous words is effective to input prediction to reduce typing effort. To solve Japanese input problems mentioned above, *Mozc* [34] can predict both the current word and the next words, using a n-gram language model (n-gram LM). A drawback of the n-gram model is that it heavily depends on the training data used for learning. It is well-suited for extremely large amounts of training data [19]. In other words, Mozc has difficulty in accurate prediction of the next words since it depends on the availability of sufficiently long series of previous word inputs. Namely, the performance of Mozc has turned out to be insufficient for user-acceptable prediction of next words.

On the other hand, Recurrent Neural Networks (RNNs) have been applied successfully to process the sequential data in speech recognition by Sak et al. [55]. They use the backpropagation through time (BPTT) [61] to Long Short-Term Memory (LSTM) [22], which is a variant of RNNs, to address the vanishing/exploding gradient problem [6] of the original RNN paradigm. RNNs are widely applied to text input prediction. Alsharif et al. [3] proposed RNN combined with Finite State Transducer (FST) [45] for the challenging task of keyboard gesture decoding. Their work deals with a very large lexicon decoding on a synthetic large scale dataset. It combines LSTM with conventional FST decoding. However, unlike a conventional software keyboard layout (e.g. Qwerty), Japanese Kana layout (e.g. 12 keys layout, Godan layout) keyboard is not suitable for gesture inputting. Most recently, Hard et al. [20] proposed a technique for mobile keyboard prediction. It uses a federated decentralized learning technique to train an LSTM model. In spite of expected high cost-performance, it has security problems of due to potential poisoning attacks by mischievous users taking part to the federation. Yu et al. [59] proposed on-device LSTM based text input prediction. It approaches to decrease the size of the language model but sacrifices a word perplexity in exchange for resizing.

This paper tackles the problem of seriously inefficient Japanese text inputting on smartphones. Our novel method focuses on predicting the next words in Japanese text input. It relies on an elaborated Recurrent Neural Network based language model (RNN-LM) and improves over previous works such as Mozc [34] that uses just n-gram. Mikolov et al. [43] have shown that RNN-LM interpolated with n-gram LM is a better model than RNN-LM only. We take a distinct, though related, approach by loosely integrating RNN-LM with an n-gram LM. RNN-LM functions for next word prediction while n-gram LM for current word completion. Considering the cost-performance of different architectures, we opted for a simple yet effective client-server model. The heavily-loaded RNN-LM is deployed on servers. The n-gram language model is deployed on local clients. Due to security threats, we do not use federated learning [20].

In order to increase the next-word prediction performance, our RNN-LM works as follows. For dimensionality reduction, a continuous bag-of-words algorithm [40] condenses sparse information (>100K) in an input layer to dense one (<1K). This reduces the computational burden of huge vocabulary due to Japanese multiple characters such as Kanji and Kana. 3 hidden layers are included each having 650 units for decreasing perplexity. To create an elaborated Recurrent Neural Network (RNN) model, hidden layers are combined with Long Short-Term Memory (LSTM) [22]. They are used as input and projection (or output) layers of LSTM. Considering cost-performance, we have two LSTMs. LSTM is known to be capable of robust training by solving the vanishing gradient problem [4], [6], [21] as described in Section IV. Back Propagation Through Time (BPTT) [61] in LSTM is used to prevent the learning from vanishing/exploding gradient. Usually, BPTT is provided with an efficient weight updating mechanism called Adam [30] that automatically adjusts the learning rates. To avoid exploding gradients, our BPTT also uses a gradient norm scaling technique [48]. The dropout methodology advocated by Zaremba et al. [62], which randomly masking only some units in input of hidden layers, is adopted. This elaboration reduces overfitting by keeping LSTM’s valuable memorizing ability intact.

The effects of our elaboration model are shown by perplexity in Japanese text prediction experiment. It is compared with excellent conventional models such as Zaremba’s. Our model is incorporated in an IME we call *Flick*. The extensive evaluation of Flick and its comparison with Mozc proves that Flick outperforms Mozc in terms of both elapsed time of inputting Japanese texts and the number of keystrokes.

The paper is organized as follows: Section II introduces and reviews related works in detail. Section III describes the details of our method. Section IV shows the results of evaluation experiments. Section V draws the conclusion.

II. Related Work

A. Input Method

There are several approaches to predict input text on mobile phones. Using prefixes or previous words is effective to input prediction to reduce typing effort. Currently, phrase-based predictive input systems are widely used [32]. POBox [41] is a predictive input technology that allows users to enter just a part of a word and then searches for similar words by spelling, pronunciation, or shape. It relies on a static database of words.

[33] proposed a corpus-based predictor that also requires a document storage system. In [16], a predictive input engine is proposed, which uses the input string and the previous inputs’ history to generate a candidate list of inputs to be presented to the user. The engine generates a candidate list by finding all previous inputs whose prefix matches the current input string and ordering them (predicted input candidates) by frequency.

Some research works approach the input of multilingual text. Dictionary-based techniques have been proposed since long [27], [56]. However, they have limitations. Thesaurus-based

approaches are known to perform badly if unknown words and new or buzz words are input [50]. Furthermore, homographs in Japanese (Kanji characters that have multiple readings) cannot be addressed with Thesaurus-based approaches [28].

Chinese modeless input method [10] can automatically switch the input mode when a user changes language between Chinese and English. Character-surface *n*-grams are exploited to detect ASCII strings with high probability to discriminate Chinese from English. Nevertheless, their method suffers from the so-called "zero frequency problem" [49]. Also, there are many cases between which it is difficult to discriminate. Consequently, false positives can occur if the *n*-gram length *n* is small, and switching delay can occur if *n* is too high. Character feature based approaches (e.g., the distribution of upper/lowercase) have succeeded but just in single character recognition for point-wise natural language processing [47].

Other researchers focused ways to detect a change in the language used for input. [1] tried to find foreign inclusions within German text. This approach improves parsing accuracy [2]. However, it is specific to kindred alphabetic languages. A method called *TypeAny* for multilingual input has been introduced in [14]. However, TypeAny requires Japanese users to press at the end of each word a delimiter key which they would not normally need to press, as Japanese is (like Europe's Uralic languages Estonian, Finnish and Hungarian) an *agglutinative* language where different morphemes can be added to words to determine their meaning [18].

To solve all these problems, a method based on multiple *n*-gram was proposed [23]. This method uses character surface, character-type, and history sequence features. It exploits two abstract features (character-type and history sequence) to handle the so-called zero frequency problem [49], achieving the generation of a fully discriminative model, i.e. one capable of making a classification suggestion for any word. As the *n*-gram system focuses on binary classification (Japanese or non-Japanese), texts including different languages can be efficiently managed. As to the use of *n*-gram features, trigrams (*n*-grams where *n*=3) have also been used for detecting whether a text is spam, which is also a binary classification problem [11].

To improve over the "pure" *n*-gram approach, we previously proposed a hybrid method [24] that combines *n*-gram with non-Japanese word dictionary matching. Using the dictionary, false positives against non-Japanese words decrease [12]. This technique decides whether current user input should be converted into Kana by looking at its encoding as a set of enhanced *n*-grams and considering both characters and character-type features. Such encodings are then classified via a linear SVM [17], [26] to achieve high-speed learning with high-accuracy for Kana-to-Kanji conversion. Both *n*-gram discrimination and non-Japanese word dictionary matching are exploited before performing Kana-to-Kanji conversion. Our previous method [24] follows the line of an open source software, *Mozc*, derived from the original Google IME. Using *n*-gram based discrimination jointly with dictionary matching, the Mozc techniques make up for the shortcomings of the other.

Mozc considers the conditional probability $P(y|x)$ of sentence y, given the input Hiragana string x. The input prediction problem given a prefix x can be formulated as a problem of deriving word y to maximize $P(y|x)$. According to Bayes' theorem, this maximization problem can be expressed as maximization of the product of a language model $P(y)$ and a Kana-Kanji model $P(x|y)$. We write:

$$\hat{y} = \operatorname{argmax} P(y|x) = \operatorname{argmax} P(y)P(x|y) \quad (1)$$

$$P(y) = \prod_{i=1}^{n} P(w_i|c_i)P(c_i|c_{i-1}) \quad (2)$$

$$P(x|y) = \prod_{i=1}^{n} P(r_i|w_i) \quad (3)$$

where c_i is the language class (e.g. part-of-speech, inflected forms) of word w_i. $P(w_i|c_i)$ is the probability of word occurrence, $P(c_i|c_{i-1})$ is the probability of class transition, r_i are candidate readings of word w_i, and finally $P(r_i|w_i)$ is the probability of reading word w_i as r_i.

As we mentioned before, both (i) the identification of the word currently being typed and (ii) the prediction of the next words that follow the current one are important for efficient typing. Due to time/labor intensive input of Japanese, (ii), the prediction of the next words is crucial. Mozc can handle both problems. However, Mozc is weak in accurate prediction of next words since this prediction depends on a sufficiently long series of previous word inputs.

Alsharif et al. proposed a keyboard gesture decoding method [3] using Long Short-Term Memory (LSTM) [22] and Finite State Transducer [45]. While this method presupposes to use a software keyboard having a conventional PC-like layout (e.g. Qwerty), usually Japanese Kana layouts (e.g. 12 keys layout, Godan layout) keyboard is not designed to input a word by a single stroke. For this reason, their method is not suitable for Japanese Kana layout keyboard. On the other hand, our approach supports not only conventional PC-like layout but also Japanese Kana layout. Additionally, this task is limited to predict or complete the word currently input.

[20] proposed the mobile keyboard prediction using federated learning, a decentralized learning technique to train models such as neural networks on users' devices, uploading only ephemeral model updates to the server for aggregation, and leaving the users' raw data on their device. While this method is based on the ethical doctrine that human nature is fundamentally good, research like [5] has shown that inference as well as poisoning attacks (injecting incorrect data) problem by mischievous users taking part to the federation are possible. Additionally, detecting poisoning attack from NNs' parameter is difficult because it is too large parameters to detect. Thus, our approach does not consider the user's feedback while learning a language model.

[59] proposed the recurrent neural network based text input prediction designed for mobile devices. It compacts RNN-LM by knowledge distillation, shared matrix factorization, and quantization. However, it sacrifices a word perplexity in exchange for language model size. On the other hand, our approach overcomes this tradeoff by putting the RNN-LM to the server and communicating via WebSocket [60], which is an interface to enable Web applications to maintain bidirectional

communications with server-side processes.

Among commercial software tools, SwiftKey[1] seems to use a neural network based predictive input. Nevertheless, there is no technical paper describing its internals. To the best of our knowledge, SwiftKey's algorithm and performance data have not been made available to the research community.

B. *Language Models*

The *n*-gram model, which was first introduced in [52], has been applied successfully to natural language processing. Here, the conditional probability of the next word is calculated by a (*n*-1)-th order Markov model with the previous (*n*-1) words as states. If the total number of words in the vocabulary is *V*, the number of possible states is therefore of the order of V^{n-1}. When *n* increases, however, the total number of states explodes exponentially (*curse of dimensionality*).

For this reason, even for small vocabularies, practical values of *n* range from 3 (trigrams) to about 5, and longer sequences cannot be practically handled. Simple *n*-gram models also suffer from the so-called *zero frequency* problem [49] regarding the estimation of the likelihood of a word occurring for the first time. Although several methods have been proposed, their suitability has been based on empirical evaluation rather than a well-founded model. [36] proposed a *cache model*, which assumes that most recently used words are often used again.

The application of Machine Learning to natural language processing has been attempted many times.

For example, [44] proposed language models based on recurrent neural networks (RNNs) [15]. The RNN language model overcomes the curse of dimensionality problem by treating word sequences as attribute vectors. This way, we do not need one bit per word of vocabulary (multiplied by *n)* to represent the *n*-gram sequence. Rather, with the vector representation, we need one bit per attribute, where attributes describe character sequences independently from the words they contain (i.e., two different sequences may share part of the attributes). Computing attribute vectors amounts to computing a projection from the sparse space of word sequences to a denser data space. This projection is adjusting itself during the neural network training. It has been shown that it can capture semantic information about words [42]. For example, words represented by similar vector to the word "France" are "Spain", "Belgium", "Netherlands", "Italy" and so on.

[8] introduces an algorithm using machine learning which trains itself using Neural Networks. Moreover, it proposes using it for the generation of replies, which is quite similar to Google's EMail reply by single tap suggestions.

[29] addresses the inefficiency issue by using long-term memory-recursive neural networks to generate text sequences in real-time conditions. It predicts a single data point at a time by considering a likelihood for the many possible word suggestions. Their system is able to generate the next real-time word in a wide variety of styles.

Quite good results can be reached by narrowing the text content subject as [7] shows. Moreover, this approach tries to address the synonym problem, but in a narrowed text content field of anamneses language.

However, current techniques focus on western languages. Attempts to apply them one by one to Japanese language ended up in the current poor text input systems for Japanese language of today's smartphones. In addition to the previously mentioned problem with the non-isomorphism of "painted" Kanji and their pronunciation and, moreover, with their (context-dependent) meaning, Japanese language analysis requests the consideration of (1) word separation (which does not appear in the word sequence), (2) its above mentioned mixture of Kanji, Hiragana, and Katakana symbols, and (3) much longer phrases to analyze the nominal meaning of a sentence due to its grammatical structure and, moreover, topical and cultural context knowledge on how to express intended meanings by nominal words, which is a bit different to the western way of expressing meanings in a more direct way of choosing words.

III. Proposed Model

A. *Predictive Text Input by Hybrid Language Models*

Predictive text input functions predict the next word based on the context. For example, as shown in Fig.1., if "カレーを" (curry wo) is input, then IME suggests "食べる" (eat), "作る" (cook) and the like.

This feature can decrease typing effort, and most IMEs for smartphones implement it. However, as discussed before, it is difficult to accurately predict next words using an *n*-gram approach, since the accuracy of this prediction depends on considering a large n, i.e. a sufficiently long sequence of previous inputs. To cope with this problem, our IME, which we call *Flick*, uses a hybrid language model. Basically, our hybrid model adds an RNN-based language model [15] (generally called RNN-LM [44]) for the next word prediction to the Mozc [34] approach (i.e. integration of a unigram and bigram language model) for the current input word conversion and completion. The Japanese input method editor Mozc was designed for the platform independent use and is adopted from Google Japanese Input, but does not suffer from extensive word



Fig. 1. Text input prediction after inputting "カレーを" (curry wo)

[1] https://swiftkey.com/

conversion tables, which means, of course a slight loss of performance, which is partly compensated by custom directories.

While RNN-LM provides effective prediction considering long contexts, dealing with large vocabulary (especially proper nouns) by RNN-LM is difficult due to computational cost. By contrast, while n-gram LMs are not suited for prediction considering long context due to the exponentially computational cost of building the language model, n-gram LM can deal with large vocabulary via a lossless compression of dictionary and language models [34]. In addition, manually parameter adjustment to n-gram LM is relatively easier than to RNN-LM. For compensating each weak point by each advantage, we integrated RNN-LM (140K words vocabulary) with n-gram LM (1.16M words vocabulary), using the client-server model. The n-gram LM is used on mobile devices since this LM requires a relatively small amount of computational cost by limiting a word frequency model to $n = 1$ (unigram) and a part-of-speech transition model to $n = 2$ (bigram). By contrast, the RNN-LM is deployed on the server due to the puny computational capacities of mobile devices.

Zaremba et al. [62] compared word perplexity (see Section IV for details) among various language models by using Penn Treebank corpus [40]. These results show that models combined with several models are better than single models. However, rich models (e.g. a model taking average from 38 models) need high computational capacity. For this reason, we need to consider the cost-performance of our model in practice. In other words, considering the trade-off between quality of prediction and cost of prediction (e.g. amount of numerical computation and memory usage) is important. Accordingly, we chose the hybrid model of our elaborated RNN-LM (details are described in Section III-B) and n-gram LM.

B. Recurrent Neural Networks Language Model

RNNs are a variety of neural networks that make it possible to model long-distance dependencies. Unlike feed-forward neural networks, RNNs are effectively applicable to the prediction of (usually long) sequences of words. As shown in Fig. 2, we elaborate RNN-LM as follows:

1. The word of current time step is converted into a one-hot-encoded sparse vector ($\boldsymbol{x}_t$). By using a continuous bag-of-words model, the word-embedding layer converts a one-hot-encoded sparse vector ($\boldsymbol{x}_t$) into dense vector ($\boldsymbol{h}_t^0$) for dimensionality reduction aiming at an efficient processing of RNN.
2. During training, Zaremba's dropout masks randomly units of hidden layers for LSTM's input ($\boldsymbol{h}_t^0, \boldsymbol{h}_t^1$) and linear transformation's input $\boldsymbol{h}_t^2$ (see Section III, B, 1, for details).
3. In LSTM block, there are the *memory cell* state and various gates. The memory cell state act as a transport highway that transfers information all the way down the sequence chain. The memory cell state, in theory, can carry relevant information to any stage of sequence processing. So information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory.

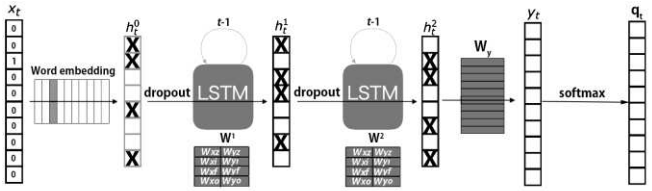


Fig.2. Structure of our RNN-LM (**X** indicates a unit masked by dropout)

As the cell state goes on its procedure, information is added or removed to the memory cell state via gates. The gates are small neural networks that decide which information is allowed on the memory cell state. The gates can learn what information is relevant to keep or forget during training.

4. For linear transformation to dimensions of the number of vocabulary, the output of last LSTM layer ($\boldsymbol{h}_t^2$) is multiplied by $\boldsymbol{W}_{yh}$, which is a weight matrix shaped by the number of words in vocabulary × the number of hidden units.
5. The softmax function converts the output vector of arbitrary real values ($\boldsymbol{y}_t$) to a vector of real values ($\boldsymbol{q}_t$) in the range [0, 1] like a discrete probability distribution ranging over different possible categories.
6. During training, Back Propagation Through Time (BPTT; details are described in Section III, B, 1) updates the weights ($\boldsymbol{W}^1, \boldsymbol{W}^2, \boldsymbol{W}_y$).

In classic feed-forward Neural Networks (NNs), each layer feeds into the next layer in a chain connecting the inputs to the outputs. At each iteration t, a new example x_t is fed to the network by setting the values of the input nodes. The NN feeds the activation forward by successively calculating the activations of each hidden layer. Finally, outputs can be read from the network's top-most layer. When the next example x_{t+1} is fed to the NN, new activations are computed, and all previous activations get overwritten. If consecutive inputs to the NN are unrelated to each other (say, images uploaded by different users), this is a perfectly acceptable. However, in our case, dependence between consecutive words must be considered.

We recall that predict text inputs, we use a Language Model (LM) which decomposes the probability over input words by:

$$P(w_1, \cdots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \cdots, w_{i-1}) \quad (4)$$

As outlined in the Introduction, LMs use known frequencies of word co-occurrences to predict the probability of observing a sentence. The probability of a sentence is computed as the product of probabilities of each word, given the words that came before it. For example, the probability of the sentence "Let's have coffee" is computed as the probability of finding in a reference dataset of sentences the word "coffee" given "Let's have", multiplied by the probability of finding "have" given "Let's".

LMs have been successfully used as *scoring mechanisms*, e.g. in speech recognition systems, which generating multiple candidate sentences for a given utterance and then use the language model to pick the most probable one [39]. Given an existing sequence of words, we generate candidate next words and their predicted probabilities, and repeat the process until we have a full sentence. On the other hand, RNNs provide a way to implement sequential structures showing long-term

dependencies, such as LMs. At each time step t, each hidden layer $\boldsymbol{h}_t^l$ of an RNN receives input from the previous hidden layer $\boldsymbol{h}_t^{l-1}$and from the same hidden layer at the previous time step ($\boldsymbol{h}_{t-1}^l$). While the basic idea looks simple enough, applying it to predicting the next word(s) that will be input by a user typing a text message is not straightforward.

To take a closer look to basic RNN operation, we start from the hidden layer update of an ordinary NN, namely

$$\boldsymbol{h} = \phi(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}) \quad (5)$$

where $\boldsymbol{x}$ is the input vector, $\boldsymbol{W}$ is the weight array and $\boldsymbol{b}$ is the bias, while ϕ is a standard activation function like *tanh( )*. The output of the RNN hidden layer has the form

$$\boldsymbol{h}_t^l = \phi(\boldsymbol{h}_t^{l-1}\boldsymbol{W}_h^l + \boldsymbol{h}_{t-1}^l\boldsymbol{W}_h^l + \boldsymbol{b}^l) \quad (6)$$

The estimated probability distribution is computed by

$$\begin{aligned} \widehat{\boldsymbol{y}_t} &= \boldsymbol{h}_t^l\boldsymbol{W}_y + \boldsymbol{b}_y \\ \widehat{\boldsymbol{q}_t} &= \mathrm{softmax}(\widehat{\boldsymbol{y}_t}) \end{aligned} \quad (7)$$

Here, L is the number of hidden layers, *softmax()* is the *normalized exponential function* [4], [21], i.e. a generalization of the logistic function that converts the output vector of arbitrary real values to a vector of real values in the range [0, 1]. The output of the *softmax()* function is used to represent a categorical distribution - that is, a discrete probability distribution ranging over different possible categories.

When our RNN predicts the upcoming word, the hidden layer has access to the current word and the activation vector of the previous one, and by extension, to the previous input(s). Also, even though the RNN in Fig.2 technically contains a loop (the hidden layer is connected to itself), this connection spans a time step. Thus, our RNN is still a feedforward network like the Locally Recurrent Globally Feedforward networks [58] and we can train it by back-propagation similar to classic NNs.

For our RNN, we used the Long Short-Term Memory (LSTM) technique for updating the hidden layer in lieu of direct update, because the LSTM is known to be more robust in training w.r.t. the *vanishing gradient* problem [4], [6], [21]. We will discuss the problem of RNN-LM training in the next section. The LSTM technique "memorizes" information from multiple time steps. As shown in Fig. 3, we use an LSTM block with the following transformations that map inputs to outputs across blocks at consecutive layers and consecutive time steps:

$$\begin{aligned} \boldsymbol{z}_t &= \tanh\left(\boldsymbol{h}_t^{l-1}\boldsymbol{W}_{xz}^l + \boldsymbol{h}_{t-1}^l\boldsymbol{W}_{yz}^l + \boldsymbol{b}_z^l\right), \\ \boldsymbol{i}_t &= \sigma\left(\boldsymbol{h}_t^{l-1}\boldsymbol{W}_{xi}^l + \boldsymbol{h}_{t-1}^l\boldsymbol{W}_{yi}^l + \boldsymbol{b}_i^l\right), \\ \boldsymbol{f}_t &= \sigma\left(\boldsymbol{h}_t^{l-1}\boldsymbol{W}_{xf}^l + \boldsymbol{h}_{t-1}^l\boldsymbol{W}_{yf}^l + \boldsymbol{b}_f^l\right), \\ \boldsymbol{o}_t &= \sigma\left(\boldsymbol{h}_t^{l-1}\boldsymbol{W}_{xo}^l + \boldsymbol{h}_{t-1}^l\boldsymbol{W}_{yo}^l + \boldsymbol{b}_o^l\right), \\ \boldsymbol{c}_t^l &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1}^l + \boldsymbol{i}_t \odot \boldsymbol{z}_t, \\ \boldsymbol{h}_t^l &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t^l), \end{aligned} \quad (8)$$

where $\boldsymbol{h}_t^l$ is a vector of l-th hidden layer at time step t, $\boldsymbol{z}$ is *block input*, $\boldsymbol{i}$ is *input gate*, $\boldsymbol{y}$ is *block output*, $\boldsymbol{f}$ is *forget gate*, $\mathbf{o}$ is *output gate*, $\boldsymbol{c}$ is *memory cell*, $\odot$ is the element-wise multiplication operator, and σ is a short-hand for the *sigmoid()* function introduced above.

LSTM has the input gate to update the memory cell state. First, the previous hidden state and current input are passed into a sigmoid function to decide which values will be updated by transforming the values to be in the range [0, 1]. A value nearly 0 means not important and nearly 1 means important. The hidden state and current input are also passed into the tanh function to squish values in the range [-1, 1] to help regulating the network. Then the tanh output is multiplied with the sigmoid output. The sigmoid output will decide which information is

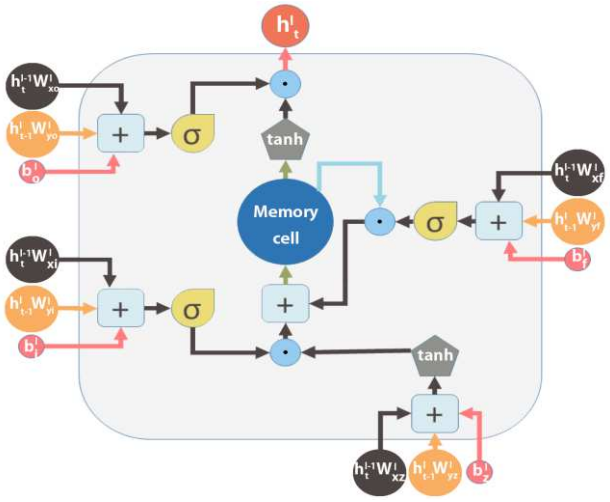


Fig. 3. Structure of LSTM block

important to keep from the tanh output. LSTM should have enough information to calculate the memory cell state. First, the memory cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the memory cell state if it gets multiplied by values near 0. Then LSTM takes the output from the input gate and do a pointwise addition which updates the memory cell state to new values that the neural network finds relevant. That gives LSTM to new memory cell state. The output gate decides the next hidden state. The hidden state contains information on previous inputs. The hidden state is also used for predictions. First, LSTM passes the previous hidden state and the current input into a sigmoid function. Then the newly modified memory cell state is passed to the tanh function. The tanh output is multiplied with the sigmoid output to decide what information the hidden state should carry. The output is the new hidden state. Then the new memory cell state and the new hidden state are carried over to the next time step. And also, the block output is recurrently connected back to the block input and all of the gates.

1) Training RNN-LM

Training RNN-LM for our specific problem requires caution. At every time step, our task is to predict the next word, given the word sequence up to that point. Classically, in the RNN training phase, a *backpropagation technique* (BP) is used to update the weights based on a loss function, which is an average of the losses at each time step. We rely on a standard loss function, *cross-entropy*, is computed as following:

$$H(p,q) = -\sum_{n}^{|V|} p(x_{t+1}^{(n)}) \log q(x_{t+1}^{(n)} | x_t^{(n)}) \quad (9)$$

where p denotes empirical distribution of the test sample, q denotes estimated probability distribution by language model, V denotes the vocabulary, $x_t^{(n)}$ denotes n-th word in the

vocabulary at the position t in the training data sequence.

When trying to back-propagate across many steps, the following problems arise: (1) The time it takes to compute a single update may be long (2) The gradient across many recurrent steps may vanish.

To prevent these problems, we feed data to our RNN in short sequences so that at every time step, our input is a single vector. Because our RNN must provide multiple outputs (one at each time step) we calculate the loss at every time step. The weights $\boldsymbol{W}$ of the RNN at time step t influence both the loss at time step t and the loss at time step $t+1$. To combine our losses into a single global loss, we take the average of the losses at each time step. This technique for accurate training is known as BPTT (*Back Propagation Through Time*) [61]. Different from BP, procedure of BPTT is as follows: (1) Present a sequence of time steps of input-output pairs to the network, (2) Unfold a recurrent layer, (3) Calculate losses across each time step by loss function, (4) Accumulate losses into a single global loss, (5) Take the average of losses at each time step, (6) Update all weights of recurrent layers and feed-forward layers, and (7) The state of memory cells after processing each time step is saved for the next time step.

This way, the training of the hidden layer consists of matrix-vector multiplications, which can be done efficiently off-line.

To update weights in BPTT, the *Adam* optimizer [30] is used. Adam combines the advantages of *AdaGrad* [13] and *RMSProp* [57], which are extensions of stochastic gradient descent. It maintains learning rates per parameter that improves performance on problems with sparse gradients. RMSProp also maintains learning rates per parameter that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g., how quickly it is changing). This means the algorithm works well on online and noisy gradients (e.g., applying dropout). Adam uses learning rates based on the average first moment (the mean) as in RMSProp, and uses also the average of the second moments of the gradients (the uncentered variance) as in AdaGrad. To determine the appropriate learning rate, Adam updates exponential moving averages of the gradient and the squared gradient. The moving averages themselves are estimates of the first and the second moments of the gradients. While these moving averages are initialized as (vectors of) zero, which leads to moment estimates that are biased towards 0, this initialization bias can be easily counteracted, resulting in bias-corrected estimates.

A flow of updating parameters by Adam is as follows:

1. Compute gradients g by loss function.
2. Update biased first moment estimate m and biased second raw moment estimate v with exponential decay rates β_1, β_2:
$$m = \beta_1 m + (1 - \beta_1) g \quad (10)$$
$$v = \beta_2 v + (1 - \beta_2) g \quad (11)$$
3. Compute bias-corrected first moment estimate $\hat{m}$ and bias-corrected second raw moment estimate $\hat{v}$:
$$\hat{m} = m / (1 - \beta_1) \quad (12)$$
$$\hat{v} = v / (1 - \beta_2) \quad (13)$$
4. Update parameter vector θ with step size α:
$$\theta = \theta - \alpha \hat{m} / (\sqrt{\hat{v}} + \varepsilon) \quad (14)$$

Kingma's work [30] shows that Adam is robust and well-suited to a wide range of non-convex optimization problems such as multi-layer neural network. Its performance gain outweighs the computational costs of building it in. Since the next word prediction task has the problems of sparse/noisy gradients and non-convex optimization, we applied Adam optimizer to update weights of our elaborated RNN-LM.

In addition, we exploit the gradient norm scaling technique, which has been proposed in [48] to prevent exploding gradient [6]. The gradient norm scaling involves changing the derivatives of the loss function to have a given vector norm when the L2 vector norm (sum of the squared values) of the gradient vector exceeds a threshold value. For example, a norm of 5 is specified, meaning that if the vector norm for a gradient exceeds 5, then the values in the vector will be rescaled so that the norm of the vector equals 5. This is formulated as follows:

$$g = \begin{cases} \dfrac{threshold}{\|g\|} g \ (\text{if } \|g\| \geq threshold) \\ g \ (\text{otherwise}) \end{cases} \quad (15)$$

where g indicates gradient.

Furthermore, as mentioned before, we applied Zaremba's dropout technique [62] to train the LSTM for reducing overfitting. The dropout operator corrupts or forgets the information carried by the units, forcing them to perform their intermediate computations more robustly and/or efficiently. At the same time, we do not want to erase all the information from the units. It is especially important that the units remember events that occurred many time steps in the past.

Standard dropout technique [54] temporarily and dynamically removes units from the network, along with all its incoming and outgoing connections. Amount of removing units is determined probabilistically or randomly. This technique works well as the Restricted Boltzmann Machine model [53].

Unfortunately, standard dropout as it is does not work well for RNNs. Standard dropout perturbs the recurrent connections by means of random determination to dynamically remove units, which makes it difficult for the LSTM to learn to store information for long periods of time. For this reason, our implementation exploits Zaremba's dropout technique, which does not use dropout on the recurrent connections. In order not to randomly remove/forget in LSTM block but to remember events that occurred on many significant time steps in the past, we used dropout outside LSTM. Moreover, according to our experiments, this particular dropout technology provides a reasonable trade-off between the additional complexity cost and the resulting performance gain.

As shown in Fig. 2, dropout is applied to only input of LSTM blocks ($\boldsymbol{h}_t^0, \boldsymbol{h}_t^1$) and linear transformation ($\boldsymbol{h}_t^2$). Therefore, our LSTM can benefit from dropout regularization without sacrificing its valuable memorization ability.

2) Generating the next word

We have just described an RNN model that takes sequences of words from our training data and tries to predict the next word at every time step. After training, RNN-LM can be deployed as a service to generate plausible sentences on behalf of applications. The generation procedure works as follows: say

TABLE IV
ELAPSED TIME FOR EACH PERSON ON THE REAL USER TEST (IN SEC.)

Person No.	1	2	3	4	5	6	7	8
Mozc	1795	1258	1418	1202	1416	1248	1260	1229
Flick (Mozc + RNN-LM)	1477	1208	1197	986	1217	933	1208	1137

our word sequence begins with the word "Reserve". We feed the word "Reserve" to the RNN-LM and get a conditional probability distribution over the next words P(x2|x1="Reserve"). We can then sample from this distribution, e.g. producing a token "table", and then assign x2="table", feeding this to the network at the next time step. We use a word encoding algorithm, continuous bag-of-words model [42] to represent words as vectors, making highly dimensional and sparse word vectors to become dense.

IV. IMPLEMENTATION AND EVALUATION

We implemented our hybrid approach as an IME called *Flick. Flick* integrates the RNN-LM-based predictive function described in the previous section with the *n*-gram processing function of the conventional tool Mozc. In this section, Flick is evaluated by comparing it to Mozc.

A. Flick Implementation

Flick is composed of three parts: a text input/output part, the *n*-gram LM part, and the RNN-LM part. The former two parts are deployed on the mobile application at the client side. The *n*-gram LM is used to predict/convert a current intended word being input with checking syntactical aspects such as a part-of-speech. The text input/output part and the *n*-gram LM part of *Flick* use *Mozc* and run at the client-side. The RNN-LM part of *Flick* is deployed and processed at the server-side due to client-side limitation of computational capacities. The RNN-LM part of *Flick* has input nodes and output nodes. Input nodes receive the past word sequence inputted by the user. Output nodes show a sequence of predicted words in the order of probability.

To train our RNN-LM, about 4,000,000 sentences from Twitter's posts (tweets) were used. And 280,000 tweets, which do not contain in the training dataset, were used as the test dataset. We used MeCab [35] as Japanese word segmentation (token reader). MeCab is an open source word segmentation library for Japanese text. MeCab can analyze and segment a Japanese sentence into its parts of speech. In this work, the mecab-ipadic is used as MeCab's dictionary. The vocabulary size of our RNN-LM is about 140,000 words.

TABLE I
COMPARISON OF PPL AMONG DIFFERENT HYPER PARAMETERS ON JAPANESE TWITTER DATASET (LOWER IS BETTER)

Hyper parameters	PPL	Parameters to be trained
Hidden unit 100	226.98	28,300,800
Hidden unit 200	205.68	56,781,600
Hidden unit 600	176.83	173,904,800
Hidden unit 600 with dropout	159.9	173,904,800
Hidden unit 650	169.22	188,905,200
Hidden unit 650 with dropout	158.28	188,905,200
Hidden unit 700	166.85	203,985,600
Hidden unit 700 with dropout	157.91	203,985,600

Hyper-parametrization is an important issue. To address it, comparison about hyper-parameters for NN architectures was newly done. As criteria, neither the prediction accuracy nor error is appropriate to express the prediction capability since this task is that user choices the next word from some candidates and also there are too many classes (vocabulary size $\approx$ 140,000) as classification problem. Thus the prediction capability of our language model is expressed by its *word perplexity* (PPL). PPL is the common evaluation metric for a language model. Since we evaluate the language model and not just the correctness of the prediction, this metrics is more appropriate than the recall metrics. Generally, PPL measures how well the proposed probability model $q(\boldsymbol{X})$ represents the target data $q^*(\boldsymbol{X})$. Let a validation dataset be $\boldsymbol{D} = \{\boldsymbol{X}^{(n)}\}_{n=1}^{|D|}$, which is a set of sentences, where the n-th sentence length is $T^{(n)}$, and the vocabulary size of this dataset is $|V|$, the *word perplexity* is represented as follows:

$$word\ perplexity = -\frac{1}{|V|}\sum_{n=1}^{|D|}\sum_{t=1}^{T^{(n)}}\log_b q(\boldsymbol{X}_t^{(n)}) \qquad (16)$$

We usually assign base $b = 2$ or $b = e$. The PPL shows how much varied the predicted distribution for the next word is. When a language model represents the dataset well, it should show a high probability only for the correct next word, so that the entropy should be high. In the above equation, the sign is reversed, so that smaller perplexity means better model.

Finally, our model combines the advantages of several techniques within hybrid language model by integrating a Recurrent Neural Network (RNN) with a n-gram model. RNNs perform best for learning long sequences for next word prediction. On the other hand, n-gram models are best at current word completion. Not just the chosen combination of learning techniques, but also the architecture supports an efficient word prediction by running the cost intensive part of RNN-LM on the sever and the n-gram model on the client. Our RNN-LM consists of an input layer equipped with word embedding, an output layer, and hidden layers connected with LSTMs (Long Short-Term Memories). Training is done via BPTT (Back Propagation Through Time). For robust training, BPTT is elaborated by learning rate refinement and gradient norm scaling. Additionally, we avoid overfitting by a dropout technique. To limit the computational complexity, our model is compact. It consists of two LSTMs and 650 units per layer). Due to synergetic effects, it shows 10% lower perplexity than Zaremba's excellent conventional models in our Japanese text prediction experiment. Our model has been incorporated into IME (Input Method Editor). On the Japanese text input experiment, Flick outperforms Mozc (Google Japanese Input) by 16% in time and 34% in the number of key strokes.

Some of the particular experimentation results are as follows.

In comparison among hyper-parameters, the condition is fixed to learning epoch = 3, 2 LSTM, threshold for gradient norm scaling = 5 and dropout rate = 0.5 (if any). The result is shown in TABLE I.

This result shows PPLs of applied dropout models are lower than the others. Comparing hidden unit = 650 (with dropout) with hidden unit = 700 (with dropout), although the PPL differs by only 0.4, the difference in the number of parameters to be trained is large. Thus, hidden unit = 650 with dropout is used.

Though the language is not Japanese but English, TABLE II, which is based on Mikolov's work [43], shows the comparison among different neural network models/architectures on the Penn Treebank corpus (1M words) [40]. KN5 denotes the baseline: interpolated 5-gram model with modified Kneser-Ney smoothing [31] and no count cutoffs. While models applied modified Kneser-Ney smoothing is often used in n-gram language models, this evaluation result shows that NN-LMs (including RNN and LSTM) outperform the n-gram model in terms of PPL. This evaluation shows also that BPTT provides a more positive effect than BP because the word perplexity of BPTT is lower than BP.

TABLE II
COMPARISON OF PPL AMONG DIFFERENT LANGUAGE MODELS ON PENN TREEBANK (1M WORDS).

Model	PPL
KN5 (baseline) [31]	186
feedforward NN	141
RNN trained by BP	137
RNN trained by BPTT	123
Zaremba's medium regularized LSTM [62]	82
Our elaborated RNN	109

Coming to Japanese, TABLE III shows the comparison among our proposed RNN-LM and the other language models, using the same Japanese Twitter dataset as used in TABLE I. While TABLE II shows our elaborated RNN-LM is not the best model on the English small dataset, TABLE III shows our elaborated RNN-LM is the best model on the Japanese large dataset. And also these comparison results proved RNN-LM is over 20-30% better than the *n*-gram approach in the prediction capability expressed by PPL.

TABLE III
COMPARISON OF PPL AMONG PROPOSED RNN-LM AND THE OTHER MODELS ON THE SAME JAPANESE TWITTER DATASET AS IN TABLE I.

Model	PPL
KN5 (baseline) [31]	247
Feedforward NN	334
Zaremba's medium regularized LSTM [62]	174
Our elaborated RNN	158

As shown in TABLE I, increasing the hidden unit number from 100 to 650 decreased the word perplexity around 25% (from 227 to 169). Though this hyper parameterization effect is quite straightforward, this also shows that describing the practical engineering technique is important, due to its leading to practically beneficial methodologies such as dropout. Indeed, in TABLE I, hidden unit 650 without dropout and hidden unit 650 with dropout are close (only 6% different) each other in word perplexity. But, the quality of prediction is significantly different. The one without dropout is much inferior. Perplexity in the Japanese twitter dataset decreases just around 6% (from 169 to 158) due to the dropout over LSTM as TABLE I shows. But, as the example shows, the one with dropout clearly suggests the word much more related to the context than that without dropout.

Despite using BPTT, going back too many time steps leads the gradient to vanish/explode. However, the length of tweets used in this experiment is limited up to 140 characters. Additionally, the average number of characters in Japanese tweets is 45.28 characters, i.e. the average number of time steps to learn is less than 45.28 since this task is not character-level prediction but word-level prediction. Thus, we do not truncate the number of time steps in BPTT. In conclusion, by virtue of scientific experimental experience for increasing prediction performance, our RNN-LM was elaborated as follows: (1) having 650 hidden units to decrease word perplexity and Long Short-Term Memory (LSTM) for robust training (2) using Zaremba's dropout to reduce overfitting. This elaborated RNN-LM combined with Mozc's n-gram is evaluated as to the Japanese input time and the number of key strokes.

B. Evaluation Method

To evaluate the effectiveness of our method using the above-mentioned elaborated RNN-LM together with the n-gram technique, we performed tests involving users of texting applications. Mozc, which uses a word n-gram language model only, provided an ideal benchmark for this analysis. We compared the time, which required to enter texts using our Flick with the time needed when using Mozc, and the number of key strokes (finger actions) to enter texts. Since our system's language model need not to be frequently updated, training is done once-for-all and offline. Therefore, the comparison with the time required for training is not needed. For this reason, we took more attention to time and keystroke reduction for users. Eight people (5 females and 3 males) participated in our experiment. The order of using methods (Mozc and Flick) is randomly changed every experiment. Also, the order of example texts to input are shuffled every experiment.

C. Corpus

As the corpus for the user test and evaluation, we used the core data of the Balanced Corpus of Contemporary Written Japanese (BCCWJ) [38], which contains various domain text data and is manually annotated with word segmentation and pronunciations. BCCWJ consists of two types of data: the balanced sampling data and the single domain data. The BCCWJ balanced sampling data is a single file that contains equal amounts of text from various domains (poems, magazine articles, blogs, government documents, and others). On the assumption that people will use the proposed method in various scenarios, we selected sentences fitting to different contexts.

D. Results

TABLE IV shows the results of the user test. Every user using our method (*Flick: Mozc + elaborated RNN-LM)* shows faster text input compared with *Mozc*.

TABLE V
AVERAGE AND STANDARD DEVIATION OF ELAPSED TIME (IN SEC.)

	Average	Standard deviation
Mozc	1353.3	183.8
Flick (Mozc + RNN-LM)	1170.4	154.6

TABLE V shows the average and standard deviation of elapsed times for both methods. To test whether our result is statistically significant, we performed a t-test, which gave a significant result: $t\,(7) = 4.7$; $p < .01$. The results show that our predictive text input based on n-gram (Mozc) enhanced by our scientifically elaborated RNN-LM exploiting 2 layers LSTM, which have 650 hidden units per layer, and Zaremba's dropout is significantly effective for Japanese text input prediction. Especially, since such RNN-LM correctly suggests the next word reflecting the context, our elaborated RNN-LM contributes to reducing text input time. And also, TABLE VI shows the standard deviation of Flick is lower than Mozc. This means a difference of input time between peoples, which familiar with mobile devices, with peoples, unfamiliar with mobile devices, is reduced by using Flick. In addition, TABLE VI shows the results of the comparison of the numbers of keystrokes to enter texts, which same as user test, by Japanese Kana-layout keyboard such as Fig. 1. *Flick* is less than *Mozc* in terms of the number of keystrokes. Compared with the directly inputting, Flick saved about 42% of the number of keystrokes. Thus, it leads that the *Flick* user's fingers are less tired and the time to input text is decreased.

V. CONCLUSION

This paper proposed a novel hybrid approach to develop a smartphone IME for Japanese. It can predict the current input words by traditional *n*-gram based text prediction model on the client but the next word by an elaborated RNN-LM on the server. To increase prediction performance, our RNN-LM is equipped with 2 LSTMs among hidden layers each having 650 units. The LSTMs use BPTT elaborated for more robust training by learning rate refinement and gradient norm scaling. To avoid overfitting, Zaremba's dropout was applied only for hidden layers. Considering cost-performance for practical use on mobile devices, our novel model was rather compact (2 LSTMs, 650 units per layer), indeed. But, due to such synergetic elaboration, in predicting Japanese text hard to input, our model experimentally showed lower perplexity than the excellent conventional models such as Zaremba's. Our technique was applied to implement a novel IME, which we call Flick. It is really used as a smartphone application. Compared with conventional IME without RNN, Flick experimentally outperformed a conventional Japanese IME (Mozc) in the input time and the number (saved 34%) of keystrokes.

TABLE VI
NUMBER OF KEYSTROKE BY KANA-LAYOUT KEYBOARD

	Average	Sum (all sentences)
Direct input (no input prediction)	26.97	1160
Mozc	21.27	915
Flick (Mozc + RNN-LM)	15.87	682

DECLARATIONS

Ethical Approval
not applicable

Competing interests
not applicable

Authors' contributions
Introduction and related work: YI, ST, RK, ED
Proposed model: YI
Implementation and Evaluation: YI
Conclusion: ST, RK

Funding
This work was supported by

- the JSPS Fund KAKENHI, grant number 15K00349
- the European Commission Fund HORIZON under the project TOREADOR, grant number H2020-688797

Availability of data and materials
not applicable

REFERENCES

[1] Alex B (2005) An unsupervised system for identifying English inclusions in German text. *Proc. of the ACL Student Research Workshop*, *Association for Computational Linguistics*, pp. 133-138.

[2] Alex B, Dubey A, Keller F (2007) Using foreign inclusion detection to improve parsing performance. *Proc. of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, *EMNLP-CoNLL*, pp. 151-160.

[3] Alsharif O, Ouyang T, Beaufays F, Zhai S, Breuel T, Schalkwyk J (2015) Long short term memory neural network for keyboard gesture decoding. *Proc. of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing*, *ICASSP 2015*, pp. 2076-2080.

[4] Angeline PJ, Saunders GM Pollack JP (1994) An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54-65.

[5] Ateniese A, Mancini LV, Spognardi A, Villani A, Vitali D, Felici G (2015) Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks*, vol. 10, issue 3, pp. 137-150.

[6] Bengio Y, Simard P, Frasconi P (1994) Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, vol. 5, issue 2, pp. 157-166.

[7] PBarman PP, Boruaha A (2018) A RNN based Approach for next word prediction in Assamese Phonetic Transcription. *Procedia Computer Science*, vol. 143, 2018, pp. 117-123.

[8] Bindu KR, Aakash C, Orlando B, Parameswaran (2018) An Algorithm for Text Prediction Using Neural Networks. Hemanth D., Smys S. (eds): *Computational Vision and Bio Inspired Computing*. Lecture Notes in Computational Vision and Biomechanics, vol. 28, Springer, pp. 186-192.

[9] Cavnar WB Trenkle JM (1994) N-gram-based text categorization. *Proc. of the SDAIR'94*, pp. 161-175.

[10] Chen Z, and Lee K (2000) A new statistical approach to Chinese Pinyin input. *Proc. of the 38th annual meeting on association for computational linguistics,* pp. 241-247.

[11] Damiani E, Vimercati SC, Paraboschi S, Samarati P (2004) An Open Digest-based Technique for Spam Detection. *Proc. of the 2004 International Workshop on Security in Parallel and Distributed Systems*, pp. 559-564.

[12] Davies M (2009) The 385+ million word corpus of contemporary American English (1990-2008+): design, architecture, and linguistic insights. *Int'l journal of corpus linguistics*, vol. 14, no. 2, pp. 159-190.

[13] Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning*, vol. 12, pp. 2121-2159.

[14] Ehara Y Tanaka-Ishii K (2008) Multilingual text entry using automatic language detection. *Proc. of the Third International Joint Conference on Natural Language Processing: Volume I*, pp. 441-448.

[15] Elman JL (1990) Finding structure in time. *Cognitive Science*. vol. 14, no. 2, pp. 179-211.

[16] Elumeze N, Nishimoto K (2006) Intelligent Predictive Text Input System using Japanese Language. *Final Report for CSCI 5832: Natural Language Processing*.

[17] Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ (2008) LIBLINEAR: a library for large linear classification. *The Journal of Machine Learning Research*, vol 9. pp. 1871-1874.

[18] Hakkani-Tur DZ, Oflazer K, Tur G (2002) Statistical morphological disambiguation for agglutinative languages. *Computers and the Humanities*, vol. 36, no. 4, pp. 381-410.

[19] Halevy A, Norvig P, Pereira F (2009) The unreasonable effectiveness of data. *Intelligent Systems*, vol. 2, issue 24, IEEE, pp. 8-12.

[20] Hard A, Rao K, Mathews R, Ramaswamy S, Beaufays F, Augenstein S, Eichner H, Kiddon C, Ramage D (2018) Federated Learning for Mobile Keyboard Prediction. *arXiv preprint arXiv:1811.03604v2*.

[21] Hochreiter S, Bengio Y, Frasconi P, Schmidhuber J (2001) Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A field guide to dynamical recurrent neural networks*, *IEEE Press*, vol. 1, pp. 1-15.

[22] Hochreiter S, Schmidhuber J (1997) LSTM: Long Short-Term Memory. *Neural computation*, vol. 9, no. 8, MIT Press, pp. 1735-1780.

[23] Ikegami Y, Sakurai Y, Tsuruta S. (2012) Modeless Japanese input method using multiple character sequence features. *Proc. of the 8th Internat. conference on signal image technology and Internet based systems*, *IEEE Computer Society*, pp. 613-618.

[24] Ikegami Y, Tsuruta S (2015) Hybrid method for modeless Japanese input using N-gram based binary classification and dictionary. *Multimedia Tools and Applications*, vol. 74, no. 11, pp. 3933-3946.

[25] Japanese Ministry of Internal Affairs and Communications (2017) 2017 WHITE PAPER Information and Communications in Japan. http://www.soumu.go.jp/johotsusintokei/whitepaper/eng/WP2017/2017-index.html [Accessed 28 August 2019].

[26] Joachims T (2006) Training linear SVMs in linear time," *Proc. of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 217-226.

[27] Kasahara S, Komachi M, Nagata M, Matsumoto Y (2011) Error correcting Romaji-kana conversion for Japanese language education. *Proc. of the workshop on advances in text input methods*, *WTIM 2011*, pp. 38-42.

[28] Kerkhofs R, Dijkstra T, Chwilla DJ, Bruijn ER (2006) Testing a model for bilingual semantic priming with interlingual homographs: RT and N400 effects," *Brain research*, Elsevier, vol. 1068, pp. 170-813.

[29] Khare A, Gupta A, Mittal A, Jyoti A (2021) Text Sequence Prediction Using Recurrent Neuronal Network. *Advances and Applications in Mathematical Sciences*, vol. 20, issue 3, pp. 377-382.

[30] Kingma DP, Ba JL (2015) Adam: A Method for Stochastic Optimization. *3rd Internat. Conf. on Learning Representations*, *ICLR 2015*.

[31] Kneser R, Ney H (1995) Improved backing-off for M-gram language modelling. *International Conference on Acoustics, Speech, and Signal Processing*, *ICASSP 1995*, vol. 1, pp. 181-184.

[32] Komachi M, Kida Y (2011) Smartphone ni okeru nihongo nyuuryoku no genjou to kadai (in Japanese). *Proc. of the 17th Annual Meeting of the Association for Natural Language Processing*, pp. 1095-1098.

[33] Komatsu H, Takabayashi S, Masui T (2005) Corpus-based predictive text input. *Proc. of the 2005 International Joint Conference on Active Media Technology*, *AMT 2005*, pp. 75-80.

[34] Kudo T, Hanaoka T, Mukai J, Tabata Y, Komatsu H (2011) Efficient dictionary and language model compression for input method editors. *Proc. of the Workshop on Advances in Text Input Methods*, *WTIM 2011*, pp. 19-25.

[35] Kudo T, Komatsu H, Hanaoka T, Mukai A, Tabata Y, Yamamoto K, Matsumoto Y (2004) Applying Conditional Random Fields to Japanese Morphological Analysis. *Proc. of the 2004 conference on empirical methods in natural language processing*, *EMNLP 2004*, pp. 230-237.

[36] Kuhn R, Mori R (1990) A cache based natural language model for speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, vol. 12, no. 6, pp. 570-583.

[37] MacKenzie IS (2002) KSPC (keystrokes per character) as a characteristic of text entry techniques. *Proc. of the Internat. Conf. on Mobile Human-Computer Interaction*, pp. 195-210.

[38] Maekawa K, Yamazaki M, Ogiso T, Maruyama T, Ogura H, Kashino W, Koiso H, Yamaguchi M, Tanaka M, Den Y (2014) Balanced corpus of contemporary written Japanese. *Language Resources and Evaluation*, vol. 48, no. 2, pp. 345-371.

[39] Manning CD, Schütze H (1999) *Foundations of statistical natural language processing*, Springer.

[40] Marcus M, Santorini B, Marcinkiewicz MA (1993) Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, vol. 19, no. 2, pp. 313-330.

[41] Masui T (1999) POBox: An efficient text input method for handheld and ubiquitous computers. *Internat. Symposium on Handheld and Ubiquitous Computing*, pp. 289-300.

[42] Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781 v3 [cs.CL]*.

[43] Mikolov T, Kombrink S, Burget L, Černocký J, Khudanpur S (2011) Extensions of recurrent neural network language model. *Proc. of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing*, *ICASSP 2011*, pp. 5528-5531.

[44] Mikolov T, Karafiat M, Burget L, Cernocky JH and Khudanpur S (2010) Recurrent neural network based language model. *Proc. of the 11th annual conf. of the international speech communication association*, *INTERSPEECH 2010*, pp. 1045-1048.

[45] Mohri M (1997) Finite-state transducers in language and speech processing *Computational linguistics*, vol. 23, no. 2, pp. 269-311.

[46] Neubig G, Duh K (2013) How much is said in a tweet? A multilingual, in-formation-theoretic perspective. *AAAI'13 spring symposium on analyzing micro-text*.

[47] Neubig G, Nakata Y, Mori S (2011) Pointwise prediction for robust, adaptable Japanese morphological analysis. *Proc. of the 49th annual meeting of the association for computational linguistics: human language technologies: short papers*, vol. 2, pp. 529-533.

[48] Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. *Proc. of the Internat. Conf. on machine learning*, *ICML 2013*, pp. 1310-1318.

[49] Pigeon S (2014) The Zero Frequency Problem (Part I). https://hbfs.wordpress.com/2014/09/23/the-zero-frenquency-problem-part-i/ [Accessed 28 August 2019].

[50] Pouliquen B, Steinberger R, Ignat C (2006) Automatic annotation of multilingual text collections with a conceptual thesaurus *arXiv preprint cs/0609059*.

[51] Roeber H, Bacus J, Tomasi C (2003) Typing in thin air: the canesta projection keyboard - a new method of interaction with electronic devices *Proc. of the CHI '03 Extended Abstracts on Human Factors in Computing Systems*, pp. 712-713.

[52] Shannon CE (1948) A mathematical theory of communication. *Bell System Technical Journal*, vol. 27, no. 3, pp. 379-423.

[53] Smolensky P (1986) Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 194-281.

[54] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, vol. 15, pp. 1929-1958.

[55] Sak H, Senior A, Beaufays F (2014) Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128 [cs.NE]*.

[56] Suzumegano F, Amano J, Maruyama Y, Hayakawa F, Namiki M, Takahashi N (1995) The evaluation environment for a Kana to Kanji transliteration system and an evaluation of the modeless input method (in Japanese) *In IPSJ SIG technical report*, vol. 1995-HI-42, pp. 9-16.

[57] Tieleman T, Hinton G (2012) Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*. pp. 26-30.

[58] Tsoi AC, Back AD (1994) Locally recurrent globally feedforward networks: a critical review of architectures. *IEEE Transactions on Neural Networks*, vol. 5, issue 2, pp. 229-239.

[59] Yu S, Kulkarni N, Lee H, Kim J (2018) On-device neural language model based word prediction. *Proc. of the 27th Internat. Conf. on Computational Linguistics: System Demonstrations*, pp. 128-131.

[60] Web Hypertext Application Technology Working Group (2018) HTML Standard 9.3 Web sockets. 2018. https://html.spec.whatwg.org/multipage/web-sockets.html [Accessed 28 August 2019].

[61] Werbos PJ (1974) Beyond Regression: new tools for prediction and analysis in the behavioral sciences. Ph.D. dissertation, Harvard University.

[62] Zaremba W, Sutskever I, Vinyals O (2015) Recurrent Neural Network Regularization. *arXiv preprint arXiv:1409.2329v5 [cs.NE]*.