Enabling Real-time Neural IME with Incremental Vocabulary Selection

Jiali Yao
Microsoft
jiayao@microsoft.com

Raphael Shu
The University of Tokyo
shu@nlab.ci.i.u-tokyo.ac.jp

Xinjian Li
Carnegie Mellon University
xinjianl@andrew.cmu.edu

Katsutoshi Ohtsuki
Microsoft
Katsutoshi.Ohtsuki@microsoft.com

Hideki Nakayama
The University of Tokyo
nakayama@ci.i.u-tokyo.ac.jp

Abstract

Input method editor (IME) converts sequential alphabet key inputs to words in a target language. It is an indispensable service for billions of Asian users. Although the neural-based language model is extensively studied and shows promising results in sequence-to-sequence tasks, applying a neural-based language model to IME was not considered feasible due to high latency when converting words on user devices. In this work, we articulate the bottleneck of neural IME decoding to be the heavy softmax computation over a large vocabulary. We propose an approach that incrementally builds a subset vocabulary from the word lattice. Our approach always computes the probability with a selected subset vocabulary. When the selected vocabulary is updated, the stale probabilities in previous steps are fixed by recomputing the missing logits. The experiments on Japanese IME benchmark shows an over 50x speedup for the softmax computations comparing to the baseline, reaching real-time speed even on commodity CPU without losing conversion accuracy[1]. The approach is potentially applicable to other incremental sequence-to-sequence decoding tasks such as real-time continuous speech recognition.

1 Introduction

Input Method Editors (IME) run on every desktop and mobile devices that allows users to type the scripts in their language. Though Latin users can type directly without conversion as a second step, some common languages such as Chinese and Japanese require users to convert the keyboard input sequence as there are thousands of characters in these languages. The conversion task of an IME takes a key sequence and converts it to a sequence of words in the target language. In the ideal case, the conversion results shall fit the intention of users. The accuracy of the conversion task directly affects the typing efficiency and user experiences.

The conversion task is a sequence decoding task similar to speech recognition, machine translation, and optical character recognition. Conventionally, an n-gram language model toolkit is used to evaluate the path probability during decoding (Stolcke, 2002; Heafield, 2011). Due to the ability of leveraging context information without hitting data sparsity issue, neural language models as an alternative option have been extensively studied in the past (Bengio et al., 2003; Schwenk, 2007; Mikolov et al., 2010; Mikolov and Zweig, 2012), which achieve state-of-the-art performance on many tasks (Sundermeyer et al., 2012; Luong et al., 2015; Jozefowicz et al., 2016). With emerging dedicated hardware processing unit such as custom ASIC (Jouppi and Young, 2017), neural-based models are promising to be even more widely applied to user devices.

However, neural-based language models were not considered feasible for the IME conversion task. The main reason is that an IME has to run interactively on various user devices, whereas speech recognition and machine translation services are normally provided on servers. Furthermore, the neural model has to meet following requirements in order to be adopted in practice: 1) **low-latency incremental conversion**; 2) **word lattice post-editing**. First, the conversion task for IME is an incremental process that needs to return the best paths immediately when receiving each key input. Existing speed optimization methods (Deoras et al., 2011; Hori et al., 2014) normally increase the speed of processing sequences in batch. Some methods incorporate prefix tree (Si et al., 2013), which doesn't ensure the worse

[1]The source code of the implementation is available from https://github.com/jiali-ms/JLM

case latency still meets the real-time requirement. Second, IME allows users to post-edit the converted results at word lattice by manually selecting candidates. It limits the choice of many end-to-end neural network architectures (Cho et al., 2014; Vaswani et al., 2017) as they do not provide a way for users to select partial conversion results.

In this work, we enhance a neural language model tailored for IMEs to meet real-time inference speed requirements on the conversion task. Our baseline model is composed of a LSTM-based language model (Hochreiter and Schmidhuber, 1997) with a Viterbi decoder (Forney, 1973) as Figure 1 shows. We articulate the bottleneck of run-time speed as the heavy linear transformation in the softmax layer. We propose an incremental vocabulary selection approach that builds a subset vocabulary V_t at each decoding step t. By only computing softmax over V_t, the cost of softmax significantly drops since $|V_t|$ is usually a small number that is less than 1% of the original vocabulary size. We evaluate the speedup comparing to other softmax optimizations on a Japanese benchmark. The contributions of this work can be summarized as:

1. We propose a novel incremental vocabulary selection approach, which significantly reduces the latency of lattice decoding in IME conversion task.

2. We provide an extensive comparison among different approaches for speeding up the softmax computation in lattice decoding.

3. We demonstrate that with our proposed acceleration method helps the neural models to meet the requirement for real-world applications.

2 Related Work

Applying a neural-based model for input method is studied in a few previous works. Chen et al. (2015) proposed a MLP architecture for Chinese Pinyin input method to re-score the n-best list of n-gram decoded results similar to speech recognition solutions (Deoras et al., 2011; Si et al., 2013). Though not in literature, we have found the implementation of RNN-based input method[2]. The decoding results on Japanese corpus show promising accuracy improvement comparing to the n-gram model. Huang et al. (2018) treats the Chinese input conversion as machine translation and apply sequence-to-sequence models with attention mechanism. The conversion and other features are served as cloud services. Our work focus on enabling real-time neural-based models on devices with limited computation resources.

The softmax layer with a large vocabulary size is the most computational-heavy operation of a neural-based language model. Differentiated softmax (Chen et al., 2016) and its variation (Joulin et al., 2017) decrease the amount of computation by reducing the embedding size of long tail words in the vocabulary with frequency based segmentation. For prediction tasks where only top-k words are necessary, softmax approximation such as SVD softmax (Shim et al., 2017) uses a low-rank matrix in the softmax layer for a first pass fast ranking. Zhang et al. (2018) proposes a word graph traverse approach to find top-k hypothesis in logarithmic complexity. In a word lattice decoder, where the target words are given, structured output layer (Mnih and Hinton, 2009; Le et al., 2011; Shi et al., 2013) is often applied to avoid calculating probability distributions from the full vocabulary. Character-based RNN (Mikolov et al., 2010) is a good alternative for word-based language models that can significantly reduce the vocabulary size. However, there are still tens of thousands of Chinese characters.

Another approach to solving the softmax bottleneck is vocabulary manipulation. In machine translation, given a source sentence, the vocabulary of target words can be largely limited before translation. It is possible to compute the softmax on a subset of the full vocabulary (Jean et al., 2015; Mi et al., 2016). However, for the input method, the conversion task takes user input incrementally. Our proposed incremental vocabulary selection can work without predicting the full vocabulary beforehand, which is designed for reducing the latency for incremental sequence decoding tasks with a large vocabulary.

3 Neural-based Input Method Editor

Our proposed approach for neural-based IME is illustrated in Figure 1. We use a neural-based language model to predict the word probabilities in the lattice decoder. Although there are

[2]Yoh Okuno. Neural ime: Neural input method engine. https://github.com/yohokuno/neural_ime, 2016

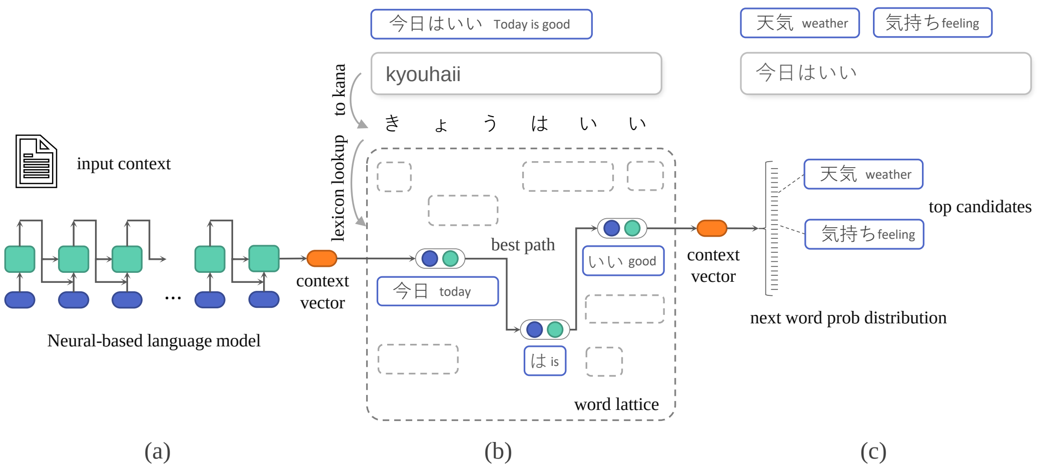

Figure 1: Illustration of a neural-based IME with LSTM language model and word lattice decoder. (a) Input context. (b) Conversion with a LSTM-based language model and Viterbi lattice decoder. (c) Word prediction.

various choices for the model architecture, we choose a single layer LSTM (Hochreiter and Schmidhuber, 1997) model considering the run-time speed constraint. Other model architectures such as sequence-to-sequence models (Cho et al., 2014) and the bi-directional LSTM-based models (Huang et al., 2015) are not considered as they cannot generate conversion results incrementally.

Conversion task is illustrated in Figure 1(b). User key inputs are first converted into Japanese Kana (also known as “fifty sounds”) with pre-defined rules. Given a partial observed Kana character input sequence $(x_1, ..., x_t)$, we search for a set of words in the dictionary that match any suffix of the observation as lattice words at step t:

$$D_t = \bigcup_{i=1}^{t} \mathrm{match}(x_i, ..., x_t), \qquad (1)$$

where the function $\mathrm{match}(\cdot)$ returns all lexicon items matching the partial sequence. For example, given the observation “ha ru ka”, the lexicon set D_t contains all words matching “ha ru ka” or “ru ka” or “ka”.

Given a word $w_t \in D_t$, to construct a hypothesis π_t ending with w_t, previous hypotheses π_{t-l} are used as the context to evaluate $p(w_t|\pi_{t-l})$, where l is the length of the Kana representation of w_t. Since l is a variable for w_t, only aligned hypotheses can be connected. To find the best hypotheses, we use a Viterbi decoder to decode with a beam size B.

We use a LSTM-based language model directly here to evaluate $p(w_t|\pi_{t-l})$. The conversion task has to compute $B \times |V| \times T$ steps of LSTM computation for one input sequence, where $|V|$ is the vocabulary size, T is the sequence length. Heavy computation cost of LSTM model comes from two operations: the matrix operations inside the LSTM cell and the matrix projection at softmax. In the case of a LSTM model with a vocabulary size of 100K, a hidden size of 512, and an embedding size of 256, estimated by simple matrix multiplication, the number of multiplication operations for LSTM cell is about 1.5M, while the softmax has 50M operations. In practice, the softmax occupies 97% of the total computation cost, which is the bottleneck for the conversion task with large vocabulary size.

4 Incremental Vocabulary Selection

To solve the challenges, we propose an incremental vocabulary selection algorithm, which significantly reduces the amount of softmax computation during decoding. The algorithm of *incremental vocabulary selection* is given in Alg. 1. Let Π be the hypothesis dictionary that stores the best hypotheses at each step with LSTM states. We define $\Pi[\cdot]$ as the dictionary lookup operation to get the best hypotheses at a specified step. Π is initialized with a start hypothesis that contains the LSTM state carried over from the previously converted sequence.

In current conversion step, when a new input character x_t arrives, we construct a subset vocabulary V_t that covers all possible output words for

Algorithm 1 Incremental vocabulary selection

```
 1: Initialize:
        $B \leftarrow$ beam size
        $\Pi \leftarrow$ hypothesis dictionary
        $\tilde{V} \leftarrow$ samples from vocabulary
        $t \leftarrow$ current step t
 2: $V_t \leftarrow$ sub vocabulary with Eq. 2
 3: $V_t \leftarrow V_t \cup \tilde{V}$
 4: $V_{\text{fix}} \leftarrow$ empty set
 5: for $k \leftarrow t-1$ to 1 do
 6:     $V_{\text{fix}} \leftarrow V_{\text{fix}} \cup (V_t \setminus V_k)$
 7:     $V_k \leftarrow V_t$
 8: Re-compute the logits on $V_{\text{fix}}$ for all past hyps
 9: Evaluate $\Pi[t]$
10: $\Pi[t] \leftarrow$ best $B$ hyps in $\Pi[t]$
11: Update LSTM state for $\Pi[t]$
12: output $\Pi[t]$
```

the sequence until step t as:

$$V_t = \bigcup_{i=1}^{t} D_i. \tag{2}$$

To evaluate a newly formed hypothesis (π_{t-l}, w_t), we need to query the probability $p(w_t|\pi_{t-l})$, which is already calculated and cached in $\Pi[t-l]$. However, as Figure 2 illustrates, since the V_t is built incrementally, the softmax distribution calculated in previous steps may not contain the word w_t. To evaluate $\Pi[t]$, we need to fix the missing vocabulary items in previously calculated softmax distributions. In principle, we can correct a stale probability in step k by adding the logits of missing vocabulary to the denominator as:

$$P(y_k = i|h_k) = \frac{\exp(h_k^\top W_i)}{\sum\limits_{j \in V_k} \exp(h_k^\top W_j) + \sum\limits_{j \in d} \exp(h_k^\top W_j)}. \tag{3}$$

where W is the projection matrix of the output layer. h_k is the cached LSTM state for a hypothesis π_k that can be re-used to calculate missing logits. d is the difference of vocabulary between steps $V_t \setminus V_k$. In practice, as each path has different missing vocabularies, we compute a union of all missing vocabularies V_{fix} and then re-compute the logits of them in one batch.

As the vocabulary in initial steps is fairly small, we introduce $\tilde{V}$ as a subset vocabulary sampled from the full vocabulary. In contrast to the beam search in machine translation, we have to rank paths that contain a different number of words. We use V_t jointly with $\tilde{V}$ to make the word probabilities closer to their original probabilities.

After the best hypotheses are decided, we immediately update the LSTM state for B hypotheses in $\Pi[t]$. The single best hypothesis in $\Pi[t]$ is finally returned to the IME engine as the output of the conversion task.

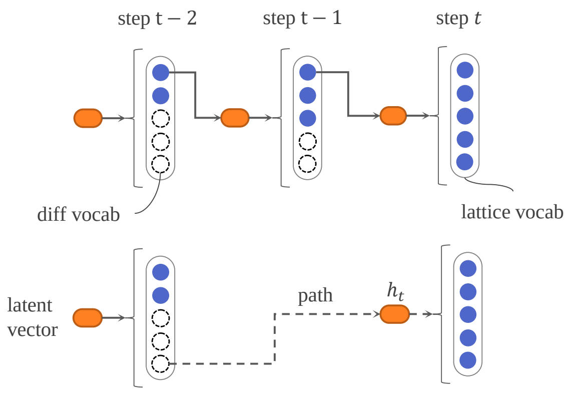

Figure 2: Previously aligned hypotheses have missing vocabulary to evaluate words in current step.

5 Experiments

5.1 Dataset

We use BCCWJ (Balanced Corpus of Contemporary Written Japanese) corpus (Maekawa et al., 2014) for evaluating our model. The corpus is well balanced with various sources of text representing contemporary written Japanese. This corpus contains 5.8M sentences, which are segmented into 127M tokens. In our experiments, all words are further segmented into short unit words. Each word has a format of “display/reading/POS”. The reading and part-of-speech (POS) attributes are attached to indicate different usages of the same word. Among the 611K unique words, we choose top 50K frequent ones which cover 97.3% of the token appearances as an appropriate vocabulary size for the IME task. The words in the vocabulary are ranked with frequency. Most frequent words are at the top.

5.2 Experiment Settings

The BCCWJ dataset is split into training, valid and test set. The ratio is 70%, 20%, and 10% respectively. We randomly sample 2000 sentences from the test set for evaluating the conversion accuracy.

For input method task, we evaluate the conversion accuracy using a Viterbi decoder with a beam size of 10. In Japanese, there are often more than one correct conversion results. For instance, a verb may have two acceptable styles, one in original Japanese Kana, the other in Chinese characters. To better evaluate the model performance, we also report the top-10 conversion accuracy in addition to top-1 conversion accuracy.

We implement using TensorFlow[3]. We use a batch size of 384. A dropout with a drop rate of 0.9 is applied before the LSTM layer. Adam is used with a fixed learning rate of 0.001. The hyper-parameters are shared for all experiments.

A replica of the same model is written in numpy to work with a Viterbi decoder in python. It uses the weights learned with TensorFlow model. The inference performance is measure with numpy on a single Intel E5 CPU. We also apply the underline BLAS library to accelerate matrix operation.

5.3 Evaluation of Neural-based Input Method Editor

We first compared the neural model performance with a conventional n-gram model. We evaluate the perplexity of the n-gram model with SRILM package (Stolcke, 2002). We choose modified Kneser Ney (Kneser and Ney, 1995; James, 2000) as the smoothing algorithm when learning the n-gram model. No cut-offs or pruning is applied. The learned language model is plugged into our input method pipeline for evaluating the sentence conversion accuracy. The prediction accuracy is not provided as the perplexity directly reflects it.

The LSTM baseline model has a standard architecture. The embedding size is 256, and the hidden size is 256. The size of LSTM cells is selected empirically on a validation dataset. In practice, using a network size bigger than 256 cannot gain significant improvement over perplexity. In all the following experiments, we bind the input embedding and output embedding according to (Press and Wolf, 2017). The idea is proven to save space and almost loss-less. We treat it as the baseline model in the following experiments.

As shown in Table 1, the LSTM baseline achieved significant improvement on perplexity, comparing to conventional n-gram based models. The top-1 and top-10 path conversion accuracy were increased by 5.6% and 8.65% respectively comparing to tri-gram KN. In real products, bi-gram is often used for decoding while tri-gram is only used for re-ranking the best paths. Please note that in this evaluation, we did not apply any pruning for n-gram. In practice, the n-gram model takes over 1GB storage size.

[3]We implement a standard LSTM cell to avoid any unexpected customization from published version

model	**pp**	**top1 hit %**	**top10 hit %**
uni-gram	833.55	26.95	45.85
bi-gram KN	99.30	51.15	78.10
tri-gram KN	68.11	55.60	79.65
LSTM baseline	**41.39**	**61.20**	**88.30**

Table 1: Performance comparison of baseline LSTM model with conventional n-gram model.

5.4 Evaluation of Run-time Speed

In this section, we compare the inference speed of various methods for accelerating the computation. We measure the execution speed only for the component that computes the language model probabilities. For neural-based methods, the component includes the LSTM and softmax layers. For the n-gram model, the computation of probability is only a lookup in the hash tables. Other components such as lattice construction are not included as they heavily depend on implementation.

We report the decoding time in each step receiving a key input, as the per-step latency is critical for the real-time user experience. The computation cost for decoding the whole sequence is linear to the number of steps. For comparison, we also report the computation time of the softmax alone.

As Table 2 shows, our proposed incremental vocabulary selection (IVS) achieves an 84x speedup for softmax computation comparing to the LSTM baseline. The lattice vocabulary in our experiments contains only a few hundred words, while the full vocabulary has 50k words. IVS only takes 3 ms to handle a new coming key. Such a high processing speed meets the real-time latency requirement even on low-end devices. In contrast, the non-incremental vocabulary selection is less efficient since it recalculates from the beginning of each step. After solving the speed bottleneck of computing the softmax layer, the majority of the computational cost comes from the LSTM cells.

We also evaluate various sampling methods with IVS. We find top sampling help close the ac-

model	total time (ms)	softmax time (ms)	top-1 hit (%)	top-10 hit (%)
tri-gram Kneser-Ney	0.0025	-	55.6	79.6
LSTM baseline w/o batch	526	513	61.2	88.3
LSTM baseline w/ batch	87	84	61.2	88.3
Char LSTM	63	21	53.2	79.8
Char LSTM w/ large beam[4]	152	55	54.4	85.2
D-Softmax	38	35	60.8	86.8
D-Softmax*	37	35	60.9	87.5
IVS	3	1	58.8	86.9
w/ top sampling	**4**	**2**	**60.1**	**88.2**
w/ uniform sampling	4	2	58.9	87.2
w/ self-norm	3	1	60.1	88.2
non-incremental VS	11	2	58.8	86.9

Table 2: Evaluation of different model acceleration approaches in terms of computation time and resultant accuracy. The computation time is reported for each step.

curacy gap between the baseline and softmax approximation. In our experiments, the number of samples used in top sampling and uniform sampling methods is both 400.

If we are able to train the neural language model from scratch, then the self-normalization (Devlin et al., 2014) approach can be applied as a softmax approximation. When applying self-normalization, the model is trained with additional normalization terms, which force the exponential logits to sum up to 1. Therefore, it eliminates the necessity of performing vocabulary sampling. However, if one only has access to a pre-trained language model, then applying self-normalization is not an option.

Differentiated softmax (D-Softmax) (Chen et al., 2016) and its variation (D-softmax*) (Joulin et al., 2017) can also reduce the amount of softmax computation by over a half depending on the segmentation strategy. However, since the vocabulary size is still large, the room for speedup is limited. For character-based LSTM models, we use a hidden size of 1024 and embedding size of 512. Since there is no direct mapping between a single Kana and a single Chinese character, we use the word lattice and evaluate the path probability by character-based LSTM. It reduces the vocabulary size from 50K to 3717 in this experiment. However, the amount of vocabulary is still non-trivial. Furthermore, the integration of character-based models is not as efficient as word-based models in this task. Consequently, we have to use a large beam size to improve accuracy.

The cost to loop previous steps during fixing the vocabulary is not given here due to the implementation detail. In a real scenario, users do not type the full sentence in one effort. Instead, they type and convert in fragments, where each fragment contains a few words. It is trivial compared to softmax computation.

We confirmed that batching is critical for accelerating matrix operations on CPU. The LSTM computation without batching is almost 7x slower with a beam size 10. Therefore, we batch all the softmax computations when fixing the vocabulary in the second pass to achieve the best speed.

6 Conclusion

IME plays an important role in improving typing efficiency and user experience. It has to work on various devices with extremely low latency. We study the key challenges to apply neural-based input method on real commodity devices. The proposed incremental vocabulary selection approach reduces the cost of computing the softmax layer without losing accuracy. Our proposed method sets a strong baseline in a real-time IME conversion task. More importantly, as the computation has low latency, the model is production-ready to be used on real devices.

[4]The large beam size is set to 50 in our experiments.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Shenyuan Chen, Hai Zhao, and Rui Wang. 2015. Neural network language model for chinese pinyin input method engine. In *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation*, pages 455–461.

Wenlin Chen, David Grangier, and Michael Auli. 2016. Strategies for training large vocabulary neural language models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1975–1985.

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 (EMNLP)*, pages 1724–1734.

Anoop Deoras, Tomas Mikolov, and Kenneth Church. 2011. A fast re-scoring strategy to capture long-distance dependencies. In *EMNLP*.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M. Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *ACL*.

G David Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Kenneth Heafield. 2011. Kenlm: Faster and smaller language model queries. In *WMT@EMNLP*.

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

Takaaki Hori, Yotaro Kubo, and Atsushi Nakamura. 2014. Real-time one-pass decoding with recurrent neural network language model for speech recognition. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6364–6368.

Yafang Huang, Zuchao Li, Zhuosheng Zhang, and Hai Zhao. 2018. Moon ime: Neural-based chinese pinyin aided input method with customizable association. In *Proceedings of ACL 2018, System Demonstrations*, pages 140–145. Association for Computational Linguistics.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Frankie James. 2000. Modified kneser-ney smoothing of n-gram models. Technical report.

Sebastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL-IJCNLP 2015*, pages 1–10. Association for Computational Linguistics (ACL).

Armand Joulin, Moustapha Cissé, David Grangier, Hervé Jégou, et al. 2017. Efficient softmax approximation for gpus. In *International Conference on Machine Learning*, pages 1302–1310.

Norman P. Jouppi and Cliff Young. 2017. In-datacenter performance analysis of a tensor processing unit. *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *icassp*, volume 1, page 181e4.

Hai Son Le, Ilya Oparin, Alexandre Allauzen, Jean-Luc Gauvain, and Franois Yvon. 2011. Structured output layer neural network language model. *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5524–5527.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Kikuo Maekawa, Makoto Yamazaki, Toshinobu Ogiso, Takehiko Maruyama, Hideki Ogura, Wakako Kashino, Hanae Koiso, Masaya Yamaguchi, Makiro Tanaka, and Yasuharu Den. 2014. Balanced corpus of contemporary written japanese. *Language resources and evaluation*, 48(2):345–371.

Haitao Mi, Zhiguo Wang, and Abe Ittycheriah. 2016. Vocabulary manipulation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 124–129.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Tomas Mikolov and Geoffrey Zweig. 2012. Context dependent recurrent neural network language model. *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239.

Andriy Mnih and Geoffrey E Hinton. 2009. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 157–163.

Holger Schwenk. 2007. Continuous space language models. *Computer Speech & Language*, 21:492–518.

Yongzhe Shi, Weiqiang Zhang, Jia Liu, and Michael T. Johnson. 2013. Rnn language model with word clustering and class-based output layer. *EURASIP J. Audio, Speech and Music Processing*, 2013:22.

Kyuhong Shim, Minjae Lee, Iksoo Choi, Yoonho Boo, and Wonyong Sung. 2017. Svd-softmax: Fast softmax approximation on large vocabulary neural networks. In *Advances in Neural Information Processing Systems*, pages 5463–5473.

Yujing Si, Qingqing Zhang, Ta Li, Jielin Pan, and Yonghong Yan. 2013. Prefix tree based n-best list re-scoring for recurrent neural network language model used in speech recognition system. In *INTERSPEECH*.

Andreas Stolcke. 2002. Srilm-an extensible language modeling toolkit. In *Seventh international conference on spoken language processing*.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. Lstm neural networks for language modeling. In *INTERSPEECH*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Minjia Zhang, Xiaodong Liu, Wenhan Wang, Jianfeng Gao, and Yuxiong He. 2018. Navigating with graph representations for fast and scalable decoding of neural language models. *CoRR*, abs/1806.04189.