Fast Word Predictor for On-Device Application

Huy Tien Nguyen[1,2,3] **Khoi Tuan Nguyen**[1] **Anh Tuan Nguyen**[1] **Thanh Lac Thi Tran**[1]

[1] Zalo Research Center, Ho Chi Minh, Vietnam
[2] Faculty of Information Technology, University of Science, Ho Chi Minh city, Vietnam
[3] Vietnam National University, Ho Chi Minh city, Vietnam
{huynt,khoint3,anhnt7,thanhttl}@zalo.me

Abstract

Learning on large text corpora, deep neural networks achieve promising results in the next word prediction task. However, deploying these huge models on devices has to deal with constraints of low latency and a small binary size. To address these challenges, we propose a fast word predictor performing efficiently on mobile devices. Compared with a standard neural network which has a similar word prediction rate, the proposed model obtains 60% reduction in memory size and 100X faster inference time on a middle-end mobile device. The method is developed as a feature for a chat application which serves more than 100 million users.

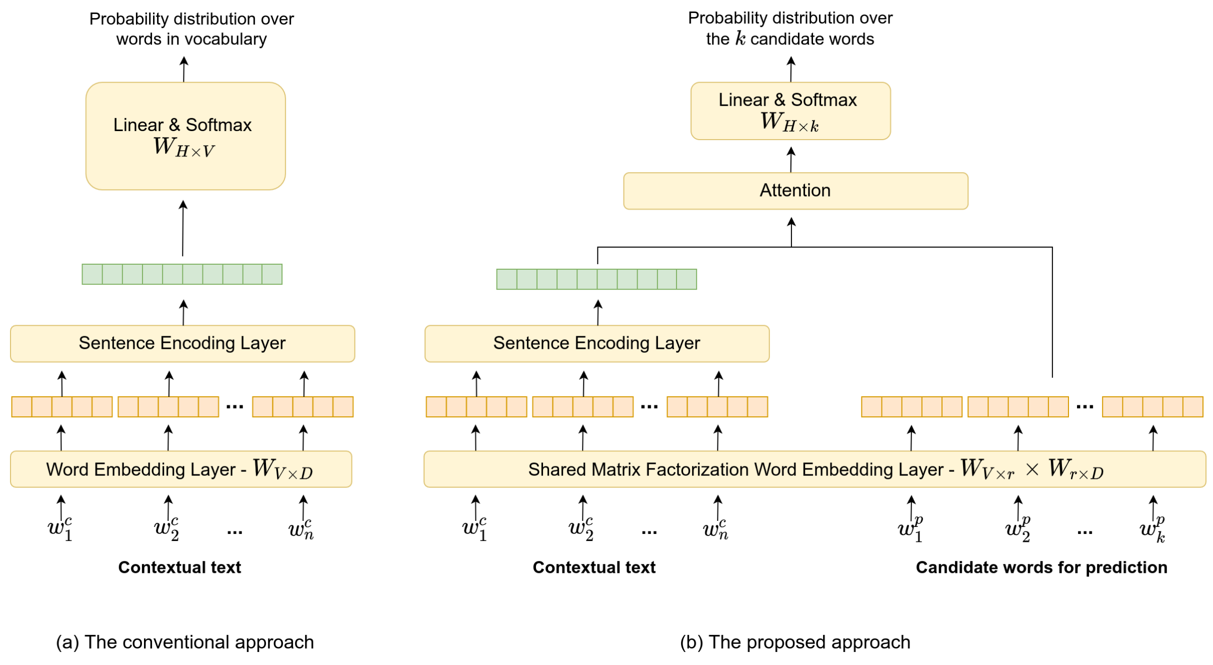
1 Introduction

As a self-supervised learning task, next word prediction based on deep neural networks obtains robust performance by learning on large text corpora. Given a textual context as input, these models shown in Figure 1(a) encode the text to generate a probability distribution over a vocabulary for the next word. Although various neural networks have been developed for efficient computation and performance, the word embedding layer and softmax layer are still essential parts in these architectures. However, this approach faces a bottleneck for deploying on mobile devices that is the huge number of parameters in the word embedding matrix and softmax layer. For a vocabulary of N words and a word embedding of dimension D, the word embedding matrix takes $N \times D$ parameters and the softmax layer takes $H \times N$ where H is the dimension of encoded text. Yu (2018), for instance, proposed a recurrent neural network with $N = 15\text{K}$, and $D = H = 600$, so the word embedding matrix and softmax layer have 18M parameters in total. This cost has limited the applicability of deep neural networks on mobile devices for word prediction, word completion, and error correction tasks.

	Wikipedia	**Social text**
Vocabulary's size	1,364,714	897,846
Num of Bigrams	14,507,901	9,868,026
Avg num of accompanied words for one-grams	10.7	11
Avg num of accompanied words for bi-grams	5.1	4.9

Table 1: Statistics on 150 million tokens.

Various approaches have been proposed for deep neural compression. Matrix factorization (Nakkiran et al., 2015; Lu et al., 2016; Mehta et al., 2020) is applied to weight matrices to reduce model parameters while weight tying (Pappas et al., 2018; Pappas et al., 2018) shares the parameters of the embedding layer with those of the output classifier. In addition, network pruning and sharing are also efficient for reducing network complexity. For instance, Srinivas (2015) and Han (2015) explored the redundancy among neurons and propose to keep only the most the relevant parameters. Recently, Yu (2018) proposed an on-device word predictor employing shared matrix factorization and distillation to optimize memory and computation. These methods achieve promising results in memory reduction but limited efficiency in time constraint, especially on low-resource devices.



Probability distribution over words in vocabulary
Linear & Softmax $W_{H\times V}$
Sentence Encoding Layer
Word Embedding Layer - $W_{V\times D}$
w_1^c w_2^c ... w_n^c
Contextual text
(a) The conventional approach
Probability distribution over the k candidate words
Linear & Softmax $W_{H\times k}$
Attention
Sentence Encoding Layer
Shared Matrix Factorization Word Embedding Layer - $W_{V\times r} \times W_{r\times D}$
w_1^c w_2^c ... w_n^c
w_1^p w_2^p ... w_k^p
Contextual text
Candidate words for prediction
(b) The proposed approach

Figure 1: Approaches for word prediction

As we analysed the bottleneck, computation complexity primarily depends on vocabulary's size N, word embedding dimension D, and hidden dimension H related to the embedding matrix and softmax layer. Designed as a lookup table, the former have constant complexity, while the latter's operation is fully matrix multiplication. Therefore, our proposed approach is to reduce this softmax layer's computation. According to our analysis on 150 million tokens from Vietnamese Wikipedia text and social text listed in Table 1, the number of words frequently accompanied with each n-gram is quite small. This fact motivates us to design a model to consider candidate words for a given context instead of looking at all words in vocabulary for predicting the next word.

In this work, we propose an approach which takes a contextual text and list of candidate words as input, and output the probability distribution over these words via the attention mechanism (Vaswani et al., 2017). This architecture shown in Figure 1(b) decreases significantly the number of softmax layer's parameters and performs faster due to the attention operation's parallelization. In addition, our proposed model keeps the contextual words and potential words in the same embedding space which is recommended for learning language models (Press and Wolf, 2017). On middle-end devices (e.g., Samsung Galaxy A8 - 2016), the proposed model takes a smaller binary memory of 2.1MB (vs. 4.9MB) and faster inference time of 0.4ms (vs 41.2ms) while achieving a similar word prediction rate compared with a conventional neural model.

The next section describes in detail the model's architecture, how to obtain a list of candidate words for a given context, and the mechanism to construct training samples for efficiently learning and diminishing bias.

2 Proposed Method

2.1 N-gram potential words

Given a context, the way to obtain candidates for word prediction is a key factor to our model's performance. Through the statistics listed in Table 1, a tri-gram model is suitable for obtaining potential words. Experimental results shows that for a given bi-gram, next words belong to the top five accompanied words in 98% cases. Obviously, more grams (i.e., 4-grams, 5-grams) are employed, candidates are more fitted. However, storing these grams requires a bigger memory space which could exceed the conventional model's. For that reason, we choose the tri-gram model to query candidates for a given context. To optimize the tri-gram model's size, tri-grams with frequency less than a threshold T = 0.01% are filtered out; and candidates for these cases are randomly selected from the top frequent words in vocabulary. According to our evaluation, the difference in word prediction rate between the tri-gram model and the filtered one is minor.

2.2 Training samples

A straightforward approach of taking the n-gram potential words as candidates faces some drawbacks (i) bias in position since candidate lists are sorted by frequency; (ii) sensitive to uncommon words as the model learns local information (i.e., comparison between potential words) but lack of global information (i.e., comparison with other words). To address these challenges, some globally frequent words are added to the list of potential words to diversify comparison instead of only potential words. It also helps to avoid bias in the most frequent words. Then, the candidate lists are randomly permuted to remove position dependence in prediction. To enhance the ability of word comparison, negative samples, where the next word is not in the candidate list, are added.

2.3 Model architecture

Given a context $W^c = \{w_1^c, w_2^c, ..., w_n^c\}$ and candidate words $W^p = \{w_1^p, w_2^p, ..., w_k^p\}$, our model embeds each one-hot word $w_i \in \mathbb{R}^V$ into a low dimension vector $e_i \in \mathbb{R}^D$ via the shared matrix factorization (Yu et al., 2018) as follows:

$$e_i = w_i \times W_{V\times r} \times W_{r\times D} \tag{1}$$

Then, the context is encoded into a context embedding vector $e^C \in \mathbb{R}^D$ via a sentence embedding layer; and the attention $a \in \mathbb{R}^k$ (Vaswani et al., 2017) are computed between the query e^c and the keys e_i^p for generating a probability distribution $p(w_i^p|W^c)$ over the candidate words as follows:

$$e^c = Embed([e_1^c, e_2^c, ..., e_n^c]) \tag{2}$$

$$e^p = [e_1^p, e_2^p, ..., e_n^p] \tag{3}$$

$$a = Attention(e^c, e^p) \tag{4}$$

$$p = softmax(Wa + b) \tag{5}$$

where $W \in \mathbb{R}^{(k+1)\times k}$ and $b \in \mathbb{R}^{k+1}$ are a weight matrix and bias respectively. Instead of an output $p \in \mathbb{R}^k$, one slot is added to the output for negative sample prediction.

3 Experiment

According to the statistic shown in Table 1 and experimental results, we choose $k = 10$ including five words from the tri-gram model and five frequent words mentioned in Section 2.2. For the model's configuration, we empirically select $n = 15, r = 10, D = 100$ and a fully connected layer for embedding sentences. We collect 1.3B Vietnamese tokens for training and 130M Vietnamese tokens for evaluation from our social platform. We use top 50K frequent words as a vocabulary and replace out-of-vocabulary words with <UNK>. To improve the performance of models with matrix factorization, we employ the TA knowledge distillation (Mirzadeh et al., 2020) with $r = 50$ for the TA model. We use a Samsung Galaxy A8 version 2015 for evaluation.

3.1 Model evaluation

In Table 2, we report the comparison between the proposed approach and some baselines (i.e., N-gram, Tying weight (TW), and TW + Matrix factorization) in terms of inference speed per sample, model size and word prediction rate (WPR) which is a percentage of correct words prediction. Although compression methods help reduce model size, they show no improvement in computation time. Our Fast Prediction model achieves significant reduction for speed while remaining a competitive WPR. We observe that parameter reduction via matrix factorization hurts the conventional model's WPR (from 0.43

to 0.31). Our Fast Word Prediction, by contrast, lessens that effect via employing candidate words for narrowing learning space.

	WPR	**Speed (ms)**	**Model Size**
N-gram	0.36	0.05	1,5MB
Tying weight (Press and Wolf, 2017)	0.43	41.16	4,9MB
Tying weight + Matrix Factorization (Yu et al., 2018)	0.31	39.68	0.6MB
Fast Word Prediction	0.44	0.36	4.9MB* + 1.5MB$^+$
Fast Word Prediction + Matrix Factorization	0.41	0.4	0.6MB* + 1.5MB$^+$

Table 2: Performance comparison of our models and baselines. (*), and ($^+$) denote the size of neural networks and N-gram models respectively.

3.2 Candidate words evaluation

In this section, we evaluate various ways to construct a list of candidate words for a given context as follows:

- **Potential** (P): top 5 words from the tri-gram model. For context being out of the tri-gram model, the potential words are selected randomly from the vocabulary.
- **Potential** + **Random** (PR): add more 5 words randomly selected from the vocabulary.
- **Potential** + **Frequent** (PF): add more 5 words randomly selected from the top frequent words.

	P	PF
Potential	0.39	0.38
Potential + Random	0.4	0.39
Potential + Frequent	0.42	0.44

Table 3: Word Prediction Rate of three ways to obtain candidate words.

We evaluate these approaches on two validation sets: (i) samples constructed by P; (ii) samples constructed by PF. The experimental results in Table 3 support the claim mentioned in Section 1 that the approach PF prevents models from being biased towards frequent words compared with P and PR.

4 Conclusion

We have proposed an efficient approach for on-device word prediction. By making use of candidate words, TA distillation and matrix factorization, the model requires less parameters and computation while achieving a competitive performance. The model takes 2.1MB in memory and outputs a result in 0.4ms on the middle-end device. To optimize the model and keep it from being bias, we build a list of candidate words including potential words and frequent words. This approach of using candidate words is promising for enhancing user experience via personalizing the candidate list.

References

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143.

Zhiyun Lu, Vikas Sindhwani, and Tara N Sainath. 2016. Learning compact recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5960–5964. IEEE.

Sachin Mehta, Rik Koncel-Kedziorski, Mohammad Rastegari, and Hannaneh Hajishirzi. 2020. Define: Deep factorized input token embeddings for neural sequence modeling. In *International Conference on Learning Representations*.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. 2020. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. In *Proceedings of The Association for the Advancement of Artificial Intelligence*.

Preetum Nakkiran, Raziel Alvarez, Rohit Prabhavalkar, and Carolina Parada. 2015. Compressing deep neural networks using a rank-constrained topology. In *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, pages 1473–1477.

Nikolaos Pappas, Lesly Miculicich, and James Henderson. 2018. Beyond weight tying: Learning joint input-output embeddings for neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 73–83, Belgium, Brussels, October. Association for Computational Linguistics.

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*, pages 157–163, Valencia, Spain, April. Association for Computational Linguistics.

Suraj Srinivas and R. Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. In Mark W. Jones Xianghua Xie and Gary K. L. Tam, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 31.1–31.12. BMVA Press, September.

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 I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Seunghak Yu, Nilesh Kulkarni, Haejun Lee, and Jihie Kim. 2018. On-device neural language model based word prediction. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 128–131, Santa Fe, New Mexico, August. Association for Computational Linguistics.