# On-Device Neural Language Model based Word Prediction

**Seunghak Yu*** **Nilesh Kulkarni*** **Haejun Lee** **Jihie Kim**
Samsung Research, Seoul, Korea
{seunghak.yu, n93.kulkarni, haejun82.lee, jihie.kim}@samsung.com

## Abstract

Recent developments in deep learning with application to language modeling have led to success in tasks of text processing, summarizing and machine translation. However, deploying huge language models on mobile devices for on-device keyboards poses computation as a bottle-neck due to their puny computation capacities. In this work, we propose an on-device neural language model based word prediction method that optimizes run-time memory and also provides a real-time prediction environment. Our model size is 7.40MB and has average prediction time of 6.47 ms. The proposed model outperforms existing methods for word prediction in terms of keystroke savings and word prediction rate and has been successfully commercialized.

## 1 Introduction

Recurrent neural networks (RNNs) have delivered state of the art performance on language modeling (RNN-LM). A major advantage of RNN-LMs is that these models inherit the property of storing and accessing information over arbitrary context lengths from RNNs. The model takes as input a textual context and generates a probability distribution over the words in the vocabulary for the next word in the text. However, the state of the art RNN-LM requires over 50MB of memory (Zoph and Le (2016) contains over 25M parameters; quantized to 2 bytes). This has, in the past, hampered deployment of RNN-LM on mobile devices for word prediction, word completion, and error correction tasks. Even on high-end mobile devices, keyboards have constraints on memory (10MB) and response time (10ms), hence we cannot apply RNN-LM directly without compression.

Various deep model compression methods have been developed. Compression through matrix factorization (Sainath et al., 2013; Xue et al., 2013; Nakkiran et al., 2015; Prabhavalkar et al., 2016; Lu et al., 2016) has shown promising results in model compression but has been applied to the tasks of automatic speech recognition. Network pruning (Han et al., 2015a; Han et al., 2015b) keeps the most the relevant parameters while removing the rest. Weight sharing (Gong et al., 2014; Chen et al., 2015; Ullrich et al., 2017) attempts to quantize the parameters into clusters. Network pruning and weight sharing methods only consider memory constraints while compressing the models. They achieve high compression but do not meet the time constraints of mobile devices and hence none of them are suitable for our application.

To address the constraints of both memory size and computation we propose a word prediction method that optimizes for run-time, and memory to render a smooth performance on embedded devices. We propose shared matrix factorization to compress the model along with using knowledge distillation to compensate the loss in accuracy while compressing. The resulting model is approximately $8\times$ compressed with negligible loss in accuracy and has a response time of 6.47ms per prediction on a high-end mobile devices (e.g. Samsung Galaxy S7). To the best of our knowledge, this is the first approach to use RNN-LMs for word prediction on mobile devices whereas previous approaches used n-gram based statistical language models or unpublished. We achieve better performance than existing approaches in terms of Keystroke Savings (KS) (Fowler et al., 2015) and Word Prediction Rate (WPR). The proposed method has been successfully commercialized.

* Equal Contribution





*Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 128–131
Santa Fe, New Mexico, USA, August 20-26, 2018.

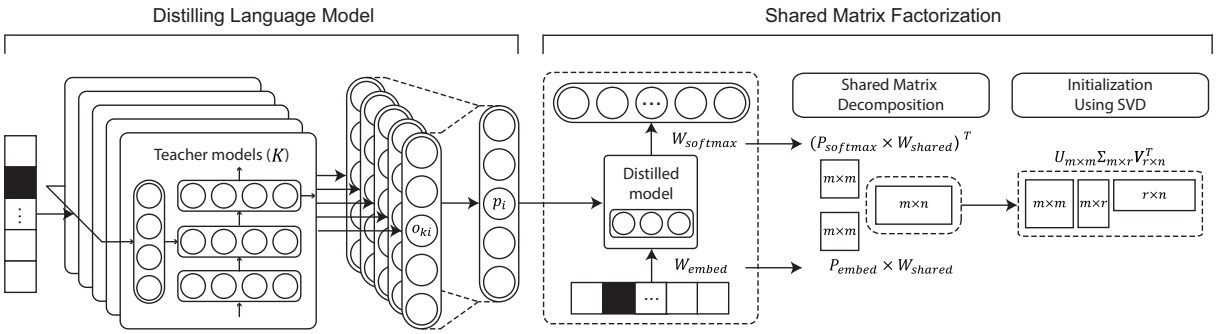


Figure 1: Overview of the proposed method. $\boldsymbol{o_{ki}}$: the $i_{th}$ logits of $k_{th}$ model, $\boldsymbol{p_i}$: the $i_{th}$ softened output of ensemble. $(P_{softmax} \times W_{shared})^T$ and $P_{embed} \times W_{shared}$ substitute $W_{softmax}$ and $W_{embed}$ in the proposed model respectively.

## 2 Proposed Method

### 2.1 Baseline Language Model

Figure 1 shows an overview of our approach. All language models in our pipeline mimic the conventional RNN-LM architecture. Each model consists of three parts: word embedding, recurrent hidden layers, and softmax layer. We use the architecture similar to the non-regularized LSTM model by (Zaremba et al., 2014). The hidden state of the LSTM unit $h_t$ is affine-transformed by the softmax function, which is a probability distribution over all the words in the $V$. We train the model with cross-entropy loss function using Adam optimizer. The initial learning rate is set to 0.001 and decays with roll-back after every epoch with no decrement in perplexity on the validation dataset.

### 2.2 Distilling Language Model

Knowledge Distillation (KD) (Hinton et al., 2015) uses an ensemble of pre-trained teacher models (typically deep and large) to train a distilled model (typically shallower). Knowledge Distillation helps provide global information to the distilled model, and hence regularizes and requires less iteration for parameter updates. We refer to 'hard targets' as true labels from the data which the baseline model uses, we adapt KD to learn a combined cost function from 'hard targets' and 'soft targets'. 'Soft targets' are generated by adding a temperature $T$ (Eq.1) to averaged logits of teachers' $z_i$ to train distilled model.

$$p_i = \frac{\exp(\frac{z_i}{T})}{\sum_j \exp(\frac{z_j}{T})} \quad (\text{where } z_i = \frac{1}{K}\sum_{k=1}^{K} o_{ki}) \tag{1}$$

### 2.3 Shared Matrix Factorization

We present a compression method using shared matrix factorization for embedding and softmax layers of the RNN-LM. We facilitate sharing by $W_{shared}$ for the softmax and embedding layers, allowing for more efficient parameterization of weight matrices. This reduces the total parameters in embedding and softmax layers by half. We introduce two trainable matrices $P_{embed}$ and $P_{softmax}$, called the projection matrices, that adapt the $W_{shared}$ for the individual tasks of embedding and softmax as $W_{embed} = P_{embed}W_{shared}$ and $W_{softmax} = (P_{softmax}W_{shared})^T$. Furthermore, in the layers parametrized by $W_{shared}$ only a few outputs are active for a given input, we suspect that they are probably correlated and the underlying weight matrix has low rank $r$. For such a weight matrix, $W$, there exists a factorization of $W_{m\times n} = W^A_{m\times r}W^B_{r\times n}$ where $W^A$ and $W^B$ are full rank. In our low-rank compression strategy, we expect rank of $W$ as $r'$ which leads to factorization as $W_{m\times n} \approx W^A_{m\times r'}W^B_{r'\times n}$.

Moreover, we compress by applying Singular Value Decomposition (SVD) to initialize the decomposed matrices. SVD has been proposed as a promising method to perform factorization for low rank matrices (Nakkiran et al., 2015; Prabhavalkar et al., 2016). We apply SVD on $W_{m\times n}$ to decompose it as $W_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$. $U, \Sigma, V$ are used to initialize $W^A$ and $W^B$ for the retraining process. We use the top $r'$ singular values from $\Sigma$ and corresponding $r'$ rows from $V^T$. Therefore,

| Model | PP | Size | CR |
|---|---|---|---|
| Baseline | 56.55 | 56.76 | - |
| + KD | 55.76 | 56.76 | - |
| + Shared Matrix | 55.07 | 33.87 | 1.68× |
| + SVD, Retrain | 59.78 | 14.80 | 3.84× |
| + Quantization | ∼**59.78** | **7.40** | **7.68**× |

Table 1: Evaluation of each model in our pipeline. Baseline uses 'hard targets' and Knowledge Distillation (KD) uses 'soft targets'. Size is in MB and 16-bit quantization is empirically selected for the final model. PP: Word Perplexity, CR: Compression Rate.

$W^A = U_{m\times m}\Sigma_{m\times r'}$ and $W^B = V^T_{r'\times n}$, we replace all the linear transformations using $W_{m\times n}$ with $W^A \times W^B$. Approximation during factorization leads to degradation in model performance but when followed by fine-tuning through retraining it results in restoration of accuracy. This compression scheme without loss of generality is applied to $W_{shared}$.

## 3 Experimental Results

### 3.1 Evaluation of proposed approach

Train data[1] is extracted from resources on the social network services in a raw form it contains 8 billion words. We uniformly sample 10% (196 million) from the dataset. Then we split dataset as 60% for training, 10% for validation and 30% for test. We preprocess raw data to remove noise and filter phrases. We also replace numbers in the dataset with a special symbol, <NUM> and out-of-vocabulary (OOV) words with <UNK>. We append start of sentence token <s> and end of sentence token </s> to every sentence. We convert our dataset to lower-case to increase vocabulary coverage and use top 15K words as the vocabulary. Table 1 shows evaluation result of each step in our pipeline. We empirically select 600 dimensional embedding , a single hidden layer with 600 LSTM hidden units for the baseline model. Word Perplexity is used to evaluate and compare our models. Perplexity over the test set is computed as $\exp(-\frac{1}{N}\sum_{i=1}^{N}\log p(w_i|w_{<i}))$, where N is the number of words in the test set. Our final model is roughly $8\times$ smaller than the baseline (which is huge and slow) with 5% (3.16) loss in perplexity.

### 3.2 Performance Comparison

We compare our performance with existing word prediction methods using manually curated dataset[2], which covers general keyboard scenarios. Due to lack of access to language modeling engine used in other commercial solutions, we are unable to compare with them on word perplexity metric. To the best of our efforts we try to minimize all the personalization these solutions offer in their prediction engines while performing the human evaluation on the manually curated dataset. We employed three evaluators from the inspection group to cross-validate all the tests in Table 2 to eliminate human errors. We achieve the best performance compared to other solutions in terms of Keystroke Savings (KS) and Word Prediction Rate (WPR) as shown in Table 2. KS is a percentage of keystrokes *not* pressed compared to a keyboard without any prediction or completion capabilities. Every character the user types using the predictions of the language model counts as keystroke saving. WPR is a percentage of correct word predictions in the test set.

## 4 Conclusions

We have proposed a practical method for training and deploying RNN-LM for a mobile device which can satisfy memory and run-time constraints. Our method utilizes averaged output of teachers to train a distilled model and compresses its weight matrices by applying shared matrix factorization. Our memory

[1]The dataset is available at `https://github.com/Meinwerk/WordPrediction`

[2]The dataset consists of 102 sentences (926 words, 3,746 characters) which are the collection of formal and informal utterances from various sources. It is also available at `https://github.com/Meinwerk/WordPrediction`

| Developer | KS(%) | WPR(%) |
|---|---|---|
| Our model | **65.11** | **34.38** |
| iOS | 64.35 | 33.73 |
| Swiftkey | 62.39 | 31.14 |
| Samsung Galaxy S6 | 59.81 | 28.84 |
| G-board | 58.89 | 28.02 |

Table 2: Performance comparison of our method and other commercialized keyboard solutions by various developers. Higher the better.

footprint is 7.40MB and is well within the run-time constraint of 10ms per prediction (6.47ms). Also, we have compared proposed method to existing commercialized keyboards in terms of keystroke savings and word prediction rate. In our benchmark tests, our method out-performed the others.

## References


Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *ICML*, pages 2285–2294.

Andrew Fowler, Kurt Partridge, Ciprian Chelba, Xiaojun Bi, Tom Ouyang, and Shumin Zhai. 2015. Effects of language modeling and its personalization on touchscreen typing performance. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 649–658. ACM.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

Song Han, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015b. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

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

Preetum Nakkiran, Raziel Alvarez, Rohit Prabhavalkar, and Carolina Parada. 2015. Compressing deep neural networks using a rank-constrained topology. In *INTERSPEECH*, pages 1473–1477.

Rohit Prabhavalkar, Ouais Alsharif, Antoine Bruguier, and Lan McGraw. 2016. On the compression of recurrent neural networks with an application to lvcsr acoustic modeling for embedded speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5970–5974. IEEE.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659. IEEE.

Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*.

Jian Xue, Jinyu Li, and Yifan Gong. 2013. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, pages 2365–2369.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.