FEDERATED LEARNING FOR MOBILE KEYBOARD PREDICTION

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays
Sean Augenstein, Hubert Eichner, Chloé Kiddon, Daniel Ramage

Google LLC,
Mountain View, CA, U.S.A.
{harda, kanishkarao, mathews, swaroopram, fsb
saugenst, huberte, loeki, dramage}@google.com

ABSTRACT

We train a recurrent neural network language model using a distributed, on-device learning framework called federated learning for the purpose of next-word prediction in a virtual keyboard for smartphones. Server-based training using stochastic gradient descent is compared with training on client devices using the FederatedAveraging algorithm. The federated algorithm, which enables training on a higher-quality dataset for this use case, is shown to achieve better prediction recall. This work demonstrates the feasibility and benefit of training language models on client devices without exporting sensitive user data to servers. The federated learning environment gives users greater control over the use of their data and simplifies the task of incorporating privacy by default with distributed training and aggregation across a population of client devices.

Index Terms— Federated learning, keyboard, language modeling, NLP, CIFG.

1. INTRODUCTION

Gboard — the Google keyboard[1]— is a virtual keyboard for touchscreen mobile devices with support for more than 600 language varieties and over 1 billion installs as of 2019. In addition to decoding noisy signals from input modalities including tap and word-gesture typing, Gboard provides auto-correction, word completion, and next-word prediction features.

As users increasingly shift to mobile devices [1], reliable and fast mobile input methods become more important. Next-word predictions provide a tool for facilitating text entry. Based on a small amount of user-generated preceding text, language models (LMs) can predict the most probable next word or phrase. Figure 1 provides an example: given the text, "I love you", Gboard predicts the user is likely to type "and", "too", or "so much" next. The center position in the suggestion strip is reserved for the highest-probability candidate, while the second and third most likely candidates occupy the left and right positions, respectively.

[1]gboard.app.goo.gl/get

Language models help in the
next word prediction task.

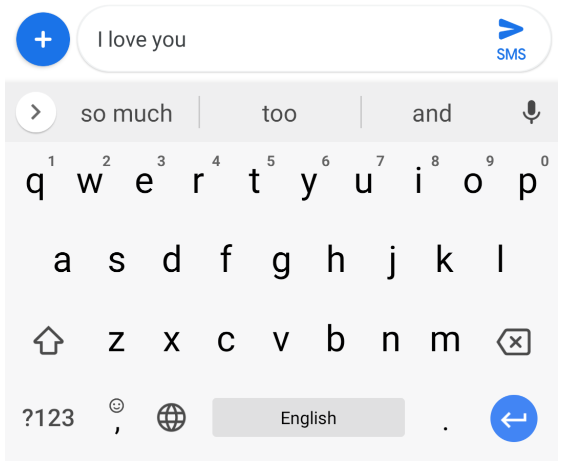

Fig. 1. Next word predictions in Gboard. Based on the context "I love you", the keyboard predicts "and", "too", and "so much".

Prior to this work, predictions were generated with a word n-gram finite state transducer (FST) [2]. The mechanics of the FST decoder in Gboard — including the role of the FST in literal decoding, corrections, and completions — are described in Ref. [3]. Next word predictions are built by searching for the highest-order n-gram state that matches the preceding text. The n-best output labels from this state are returned. Paths containing back-off transitions to lower-orders are also considered. The primary (static) language model for the English language in Gboard is a Katz smoothed Bayesian interpolated [4] 5-gram LM containing 1.25 million n-grams, including 164,000 unigrams. Personalized user history, contacts, and email n-gram models augment the primary LM.

Mobile keyboard models are constrained in multiple ways. In order to run on both low and high-end devices, models should be small and inference-time latency should be low. Users typically expect a visible keyboard response within 20 milliseconds of an input event. Given the frequency with which mobile keyboard apps are used, client device batteries could be quickly depleted if CPU consumption were not constrained. As a result, language models are usually limited to tens of megabytes in size with vocabularies of hundreds of thousands of words.

Neural models — in particular word and character-level recurrent neural networks (RNNs) [5] — have been shown to perform well on language modeling tasks [6, 7, 8]. Unlike n-gram models and feed-forward neural networks that rely on a fixed historical context window, RNNs utilize an arbitrary and dynamically-sized context window. Exploding and vanishing gradients in the back-propagation through time algorithm can be resolved with the Long Short-Term Memory (LSTM) [6]. As of writing, state-of-the art perplexities on the 1 billion word benchmark [9] have been achieved with LSTM variants [10, 11].

Training a prediction model requires a large data sample that is representative of the text that users will commit. Publicly available datasets can be used, though the training distribution often does not match the population's distribution. Another option is to sample user-generated text. This requires logging, infrastructure, dedicated storage on a server, and security. Even with data cleaning protocols and strict access controls, users might be uncomfortable with the collection and remote storage of their personal data [12].

Apart from privacy, the FL approach also improves recall w.r.t FST baseline.

In this paper, we show that federated learning provides an alternative to the server-based data collection and training paradigm in a commercial setting. We train an RNN model from scratch in the server and federated environments and achieve recall improvements with respect to the FST decoder baseline.

The paper is organized in the following manner. Section 2 summarizes prior work related to mobile input decoding, language modeling with RNNs, and federated learning. Coupled Input-Forget Gates (CIFG) — the RNN variant utilized for next-word prediction — are described in Section 3. Section 4 discusses the federated averaging algorithm in more depth. Section 5 summarizes experiments with federated and server-based training of the models. The results of the studies are presented in Section 6, followed by concluding remarks in Section 7.

2. RELATED WORK

FSTs have been explored in the context of mobile keyboard input decoding, correction, and prediction [3]. LSTMs have greatly improved the decoding of gestured inputs on mobile keyboards [13]. RNN language models optimized for word prediction rate and keystroke savings within inference-time latency and memory constraints have also been published [14, 15].

Research into distributed training for neural models has gained relevance with the recent increased focus on privacy and government regulation. In particular, federated learning has proved to be a useful extension of server-based distributed training to client device-based training using locally stored data [12, 16]. Language models have been trained using the federated algorithm combined with differential privacy [17, 18]. And Gboard has previously used federated learning to train a model to suggest search queries based on typing context [19], though the results have not been published yet. To the best of our knowledge, there are no existing publications that train a neural language model for a mobile keyboard with federated learning.

3. MODEL ARCHITECTURE

Variant of LSTM

The next-word prediction model uses a variant of the Long Short-Term Memory (LSTM) [6] recurrent neural network called the Coupled Input and Forget Gate (CIFG) [20]. As with Gated Recurrent Units [21], the CIFG uses a single gate to control both the input and recurrent cell self-connections, reducing the number of parameters per cell by 25%. For timestep t, the input gate i_t and forget gate f_t have the relation:

$$f_t = 1 - i_t. \quad (1)$$

The CIFG architecture is advantageous for the mobile device environment because the number of computations and the parameter set size are reduced with no impact on model performance. The model is trained using TensorFlow [22] without peephole connections. On-device inference is supported by TensorFlow Lite[2].

Tied input embedding and output projection matrices are used to reduce the model size and accelerate training [23, 24]. Given a vocabulary of size V, a one-hot encoding $v \in \mathbb{R}^V$ is mapped to a dense embedding vector $d \in \mathbb{R}^D$ by $d = Wv$ with an embedding matrix $W \in \mathbb{R}^{D \times V}$. The output projection of the CIFG, also in $\mathbb{R}^D$, is mapped to the output vector $W^{\intercal}h \in \mathbb{R}^V$. A softmax function over the output vector converts the raw logits into normalized probabilities. Cross-entropy loss over the output and target labels is used for training.

The client device requirements alluded to in Section 1 limit the vocabulary and model sizes. A dictionary of $V =$ 10,000 words is used for the input and output vocabularies. Input tokens include special beginning of sentence, end of sentence, and out-of-vocabulary tokens. During network evaluation and inference, the logits corresponding to these special tokens are ignored. The input embedding and CIFG output projection dimension D is set to 96. A single layer CIFG with 670 units is used. Overall, 1.4 million parameters comprise the network — more than two thirds of which are associated with the embedding matrix W. After weight quantization, the model shipped to Gboard devices is 1.4 megabytes in size.

[2]https://www.tensorflow.org/lite/

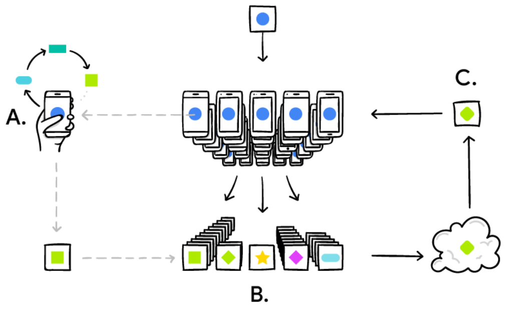

Fig. 2. An illustration of the federated learning process from Ref. [19]: (A) client devices compute SGD updates on locally-stored data, (B) a server aggregates the client updates to build a new global model, (C) the new model is sent back to clients, and the process is repeated.

4. FEDERATED LEARNING

Federated learning [12, 16] provides a decentralized computation strategy that can be employed to train a neural model. Mobile devices, referred to as clients, generate large volumes of personal data that can be used for training. Instead of uploading data to servers for centralized training, clients process their local data and share model updates with the server. Weights from a large population of clients are aggregated by the server and combined to create an improved global model. Figure 2 provides an illustration of the process. The distributed approach has been shown to work with unbalanced datasets and data that are not independent or identically distributed across clients.

The `FederatedAveraging` algorithm [12] is used on the server to combine client updates and produce a new global model. At training round t, a global model w_t is sent to a subset K of client devices. In the special case of $t = 0$, client devices start from the same global model that has either been randomly initialized or pre-trained on proxy data. Each of the clients participating in a given round has a local dataset consisting of n_k examples, where k is an index of participating clients. n_k varies from device to device. For studies in Gboard, n_k is related to the user's typing volume.

Every client computes the average gradient, g_k, on its local data with the current model w_t using one or more steps of stochastic gradient descent (SGD). For a client learning rate ϵ, the local client update, w_{t+1}^k, is given by:

$$w_t - \epsilon g_k \rightarrow w_{t+1}^k. \tag{2}$$

The server then does a weighted aggregation of the client models to obtain a new global model, w_{t+1}:

$$\sum_{k=1}^{K} \frac{n_k}{N} w_{t+1}^k \rightarrow w_{t+1}, \tag{3}$$

where $N = \sum_k n_k$. In essence, the clients compute SGD updates locally, which are communicated to the server and aggregated. Hyperparameters including the client batch size, the number of client epochs, and the number of clients per round (global batch size) are tuned to improve performance.

Decentralized on-device computation offers fewer security and privacy risks than server storage, even when the server-hosted data are anonymized. Keeping personal data on client devices gives users more direct and physical control of their own data. The model updates communicated to the server by each client are ephemeral, focused, and aggregated. Client updates are never stored on the server; updates are processed in memory and are immediately discarded after accumulation in a weight vector. Following the principle of data minimization [25], uploaded content is limited to model weights. Finally, the results are only used in aggregate: the global model is improved by combining updates from many client devices. The federated learning procedure discussed here requires users to trust that the aggregation server will not scrutinize individual weight uploads. This is still preferable to server training because the server is never entrusted with user data. Additional techniques are being explored to relax the trust requirement. Federated learning has previously been shown to be complementary to privacy-preserving techniques such as secure aggregation [26] and differential privacy [17].

5. EXPERIMENTS

Federated learning and server-based stochastic gradient descent are used to train the CIFG language model described in Section 3 starting from random weight initializations. The performance of both models is evaluated on server-hosted logs data, client-held data, and in live production experiments.

5.1. Server-based training with logs data

Server-based training of the CIFG next-word prediction model relies on data logged from Gboard users who have opted to share snippets of text while typing in Google apps. The text is truncated to contain short phrases of a few words, and snippets are only sporadically logged from individual users. Prior to training, logs are anonymized and stripped of personally identifiable information. Additionally, snippets are only used for training if they begin with a start of sentence token.

For this study, logs are collected from the English speaking population of Gboard users in the United States. Approximately 7.5 billion sentences are used for training, while the test and evaluation samples each contain 25,000 sentences. The average sentence length in the dataset is 4.1 words. A breakdown of the logs data by app type is provided in Table 1. Chat apps generate the majority of logged text.

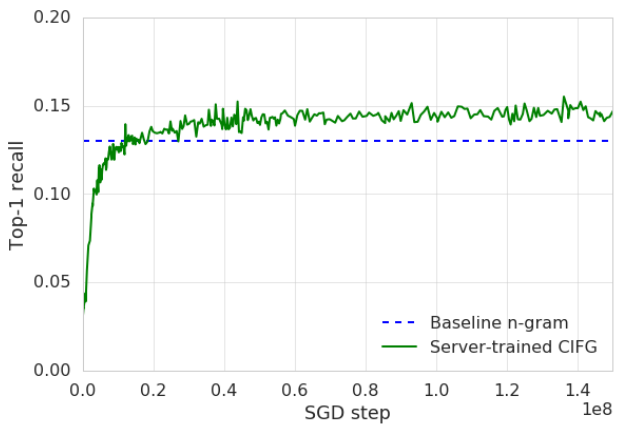

Fig. 3. Top-1 recall of the CIFG as a function of SGD step during server training. The recall of the n-gram FST baseline model is shown for comparison, but the FST model is not trained in this study.

Adam and AdaGrad don't help much. SGD is fine.

Asynchronous stochastic gradient descent with a learning rate equal to 10^{-3} and no weight decay or momentum is used to train the server CIFG. Adaptive gradient methods including Adam [27] and AdaGrad [28] are not found to improve the convergence. Sentences are processed in batches of 50. The network converges after 150 million steps of SGD. Figure 3 shows the top-1 recall of the CIFG during network training, compared with the performance of the n-gram baseline model.

App type	Share of data
Chat	60%
Web input	35%
Long form text	5%

Table 1. The composition of logs data by mobile app type.

5.2. Federated training with client caches

Data for the federated training of the CIFG next-word prediction model are stored in local caches on Gboard client devices. As with the logs data, each client cache stores text belonging to the device owner, as well as prediction candidates generated by the decoder.

Client devices must meet a number of requirements in order to be eligible for federated training participation. In terms of hardware requirements, the devices must have at least 2 gigabytes of memory available. Additionally, the clients are only allowed to participate if they are charging, connected to an un-metered network, and idle. These criteria are chosen specifically for the Gboard implementation of federated learning and are not inherent to the federated learning platform. Clients for this study are also required to be located in North America while running Gboard release 7.3 or greater with the US English language model enabled.

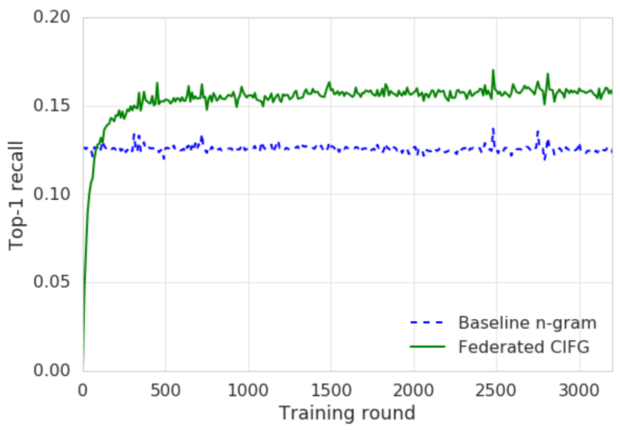

Fig. 4. Top-1 recall of the CIFG as a function of training round during federated training. The performance of the n-gram FST baseline model is evaluated on the client caches along with the CIFG, but it is not trained in this study.

Unlike server-based training, where train, test, and eval samples are obtained via explicit splits of the data, the federated train, test, and eval samples are obtained by defining separate computation tasks. While there is no explicit separation of client devices into three distinct populations, the probability of client reuse in both the training and test or eval tasks is minimal in a sufficiently large client population. The composition of the client cache data by app type is shown in Table 2. As with the logs data, the client caches are also dominated by chat apps. Social media apps have an increased presence in the client cache sample, while long-form communication is represented less.

App type	Share of data
Chat	66%
Social	16%
Web input	5%
Other	12%

Table 2. The composition of client cache data by mobile app type.

The `FederatedAveraging` algorithm described in Section 4 is used to aggregate distributed client SGD updates. Between 100 and 500 client updates are required to close each round of federated training in Gboard. The server update in Equation 3 is achieved via the Momentum optimizer, using Nesterov accelerated gradient [29], a momentum hyperparameter of 0.9, and a server learning rate of 1.0. This technique is found to reduce training time with respect to alternatives including pure SGD. On average, each client processes approximately 400 example sentences during a single training epoch. The federated CIFG converges after 3000 training rounds, over the course of which 600 million sentences are processed by 1.5 million clients. Training typically takes 4-5 days. The top-1 recall of the federated CIFG is shown as a function of training round in Figure 4. The performance of the n-gram baseline model is also measured in the federated eval tasks to provide a comparison for the CIFG, though the decoder is not trained in this study. N-gram model recall is measured by comparing the decoder candidates stored in the on-device training cache to the actual user-entered text.

6. RESULTS

The performance of each model is evaluated using the recall metric, defined as the ratio of the number of correct predictions to the total number of tokens. Recall for the highest-likelihood candidate is important for Gboard because users are more prone to read and utilize predictions in the center suggestion spot. Since Gboard includes three candidates in the suggestion strip, top-3 recall is also of interest.

Model	Top-1 recall	Top-3 recall
N-gram	13.0%	22.1%
Server CIFG	16.5%	27.1%
Federated CIFG	16.4%	27.0%

Table 3. Prediction recall for the server and federated CIFG models compared with the n-gram baseline, evaluated on server-hosted logs data.

Server-hosted logs data and client device-owned caches are used to measure prediction recall. Although each contain snippets of data from actual users, the client caches are believed to more accurately represent the true typing data distribution. Cache data, unlike logs, are not truncated in length and are not restricted to keyboard usage in Google-owned apps. Thus, federated learning enables the use of higher-quality training data in the case of Gboard. Table 3 summarizes the recall performance as measured on server-hosted logs data, while Table 4 shows the performance evaluated with client-owned caches. The quoted errors are directly related to the number of clients used for federated evaluation.

Model	Top-1 recall [%]
N-gram	12.5 ± 0.2
Server CIFG	15.0 ± 0.5
Federated CIFG	15.8 ± 0.3

Table 4. Prediction recall for the server and federated CIFG models compared with the n-gram baseline, evaluated on client-owned data caches.

Model performance is also measured in live production experiments with a subset of Gboard users. Similar to top-1 recall, prediction impression recall is measured by dividing the number of predictions that match the user-entered text by the number of times users are shown prediction candidates. The prediction impression recall metric is typically lower than the standard recall metric. Zero-state prediction events (in which users open the Gboard app but do not commit any text) increase the number of impressions but not matches. Table 5 summarizes the impression recall performance in live experiments. The prediction click-through rate (CTR), defined as the ratio of the number of clicks on prediction candidates to the number of proposed prediction candidates, is also provided in Table 6. Quoted 95% CI errors for all results are derived using the jackknife method with user buckets.

The prediction of the tokens is the same as what was written by the user.

Model	Top-1 recall [%]	Top-3 recall [%]
N-gram	5.24 ± 0.02	11.05 ± 0.03
Server CIFG	5.76 ± 0.03	13.63 ± 0.04
Federated CIFG	5.82 ± 0.03	13.75 ± 0.03

Table 5. Prediction impression recall for the server and federated CIFG models compared with the n-gram baseline, evaluated in experiments on live user traffic.

Model	Prediction CTR [%]
N-gram	2.13 ± 0.03
Server CIFG	2.36 ± 0.03
Federated CIFG	2.35 ± 0.03

Table 6. Prediction CTR for the server and federated CIFG models compared with the n-gram baseline, evaluated in experiments on live user traffic.

For both server training and federated training, the CIFG model improves the top-1 and top-3 recall with respect to the baseline n-gram FST model. These gains are impressive given that the n-gram model uses an order of magnitude larger vocabulary and includes personalized components such as user history and contacts LMs. Live user experiments show that the CIFG model also generates predictions that are 10% more likely to be clicked than n-gram predictions.

The results also demonstrate that the federated CIFG performs better on recall metrics than the server-trained CIFG. Table 4 shows that, when evaluating on client cache data, the federated CIFG improves the top-1 recall by a relative 5% (0.8% absolute) with respect to the server-trained CIFG. Comparisons on server-hosted logs data show the recall of the two models is comparable, though the logs are not as representative of the true typing distribution. Most importantly, Table 5 shows that the federated CIFG improves the top-1 and top-3 prediction impression recall by 1% relative to the server

CIFG for real Gboard users. While the comparison is not exactly apples to apples — different flavors of SGD are used in each training context — the results show that federated learning provides a preferable alternative to server-based training of neural language models.

7. CONCLUSION

We show that a CIFG language model trained from scratch using federated learning can outperform an identical server-trained CIFG model and baseline n-gram model on the keyboard next-word prediction task. To our knowledge, this represents one of the first applications of federated language modeling in a commercial setting. Federated learning offers security and privacy advantages for users by training across a population of highly distributed computing devices while simultaneously improving language model quality.

8. ACKNOWLEDGEMENTS

The authors would like to thank colleagues on the Google AI team for providing the federated learning framework and for many helpful discussions.

9. REFERENCES

[1] Monica Anderson, "Technology device ownership: 2015," `http://www.pewinternet.org/2015/10/29/technology-device-ownership-2015/`, Accessed: 2018-10-02.

[2] Mehryar Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 2, pp. 269–311, June 1997.

[3] Tom Ouyang, David Rybach, Françoise Beaufays, and Michael Riley, "Mobile keyboard input decoding with finite-state transducers," *CoRR*, vol. abs/1704.03987, 2017.

[4] Cyril Allauzen and Michael Riley, "Bayesian language model interpolation for mobile speech input," in *Interspeech 2011*, 2011, pp. 1429–1432.

[5] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 270–280, June 1989.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov 1997.

[7] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Mar. 2003.

[8] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush, "Character-aware neural language models," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 2016, AAAI'16, pp. 2741–2749, AAAI Press.

[9] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson, "One billion word benchmark for measuring progress in statistical language modeling," in *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, Haizhou Li, Helen M. Meng, Bin Ma, Engsiong Chng, and Lei Xie, Eds. 2014, pp. 2635–2639, ISCA.

[10] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu, "Exploring the limits of language modeling," 2016.

[11] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *CoRR*, vol. abs/1701.06538, 2017.

[12] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, Aarti Singh and Xiaojin (Jerry) Zhu, Eds. 2017, vol. 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282, PMLR.

[13] Ouais Alsharif, Tom Ouyang, Franoise Beaufays, Shumin Zhai, Thomas Breuel, and Johan Schalkwyk, "Long short term memory neural network for keyboard gesture decoding," *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2076–2080, 2015.

[14] Seunghak Yu, Nilesh Kulkarni, Haejun Lee, and Jihie Kim, "On-device neural language model based word prediction," in *COLING 2018, The 27th International Conference on Computational Linguistics: System Demonstrations, Santa Fe, New Mexico, August 20-26, 2018*, Dongyan Zhao, Ed. 2018, pp. 128–131, Association for Computational Linguistics.

[15] Shaona Ghosh and Per Ola Kristensson, "Neural networks for text correction and completion in keyboard decoding," *CoRR*, vol. abs/1709.06429, 2017.

[16] Reza Shokri and Vitaly Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2015, CCS '15, pp. 1310–1321, ACM.

[17] Cynthia Dwork, "Differential privacy," in *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, Venice, Italy, July 2006, vol. 4052, pp. 1–12, Springer Verlag.

[18] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang, "Learning differentially private language models without losing accuracy," *CoRR*, vol. abs/1710.06963, 2017.

[19] Brendan McMahan and Daniel Ramage, "Federated learning: Collaborative machine learning without centralized training data," `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`, Accessed: 2018-10-04.

[20] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 28, no. 10, pp. 2222–2232, 2017.

[21] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "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 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, Alessandro Moschitti, Bo Pang, and Walter Daelemans, Eds. 2014, pp. 1724–1734, ACL.

[22] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, Kimberly Keeton and Timothy Roscoe, Eds. 2016, pp. 265–283, USENIX Association.

[23] Ofir Press and Lior Wolf, "Using the output embedding to improve language models," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, Mirella Lapata, Phil Blunsom, and Alexander Koller, Eds. 2017, pp. 157–163, Association for Computational Linguistics.

[24] Hakan Inan, Khashayar Khosravi, and Richard Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," *CoRR*, vol. abs/1611.01462, 2016.

[25] The White House, "Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy," 01 2013.

[26] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth, "Practical secure aggregation for federated learning on user-held data," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[27] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ArXiv e-prints*, Dec. 2014.

[28] John Duchi, Elad Hazan, and Yoram Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, July 2011.

[29] Yurii Nesterov, "A method for solving the convex programming problem with convergence rate $o(1/k^2)$," *Dokl. Akad. Nauk SSSR*, vol. 269, pp. 543–547, 1983.